Part II

Inference

7 Inference algorithms: an overview

7.1 Introduction

In the probabilistic approach to machine learning, all unknown quantities — be they predictions about the future, hidden states of a system, or parameters of a model — are treated as random variables, and endowed with probability distributions. The process of **inference** corresponds to computing the posterior distribution over these quantities, conditioning on whatever data is available.

In more detail, let $\boldsymbol{\theta}$ represent the unknown variables, and $\mathcal{D}$ represent the known variables. Given a likelihood $p(\mathcal{D}|\boldsymbol{\theta})$ and a prior $p(\boldsymbol{\theta})$, we can compute the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ using Bayes' rule:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D})} \tag{7.1}$$

The main computational bottleneck is computing the normalization constant in the denominator, which requires solving the following high dimensional integral:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{7.2}$$

This is needed to convert the unnormalized joint probability of some parameter value, $p(\boldsymbol{\theta}, \mathcal{D})$, to a normalized probability, $p(\boldsymbol{\theta}|\mathcal{D})$, which takes into account all the other plausible values that $\boldsymbol{\theta}$ could have.

Once we have the posterior, we can use it to compute posterior expectations of some function of the unknown variables, i.e.,

$$\mathbb{E}\left[g(\boldsymbol{\theta})|\mathcal{D}\right] = \int g(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{7.3}$$

By defining g in the appropriate way, we can compute many quantities of interest, such as the following:

$$\text{mean: } g(\boldsymbol{\theta}) = \boldsymbol{\theta} \tag{7.4}$$

$$\text{covariance: } g(\boldsymbol{\theta}) = (\boldsymbol{\theta} - \mathbb{E}\left[\boldsymbol{\theta}|\mathcal{D}\right])(\boldsymbol{\theta} - \mathbb{E}\left[\boldsymbol{\theta}|\mathcal{D}\right])^{\mathsf{T}} \tag{7.5}$$

$$\text{marginals: } g(\boldsymbol{\theta}) = p(\theta_1 = \theta_1^*|\boldsymbol{\theta}_{2:D}) \tag{7.6}$$

$$\text{predictive: } g(\boldsymbol{\theta}) = p(\boldsymbol{y}_{N+1}|\boldsymbol{\theta}) \tag{7.7}$$

$$\text{expected loss: } g(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta}, a) \tag{7.8}$$

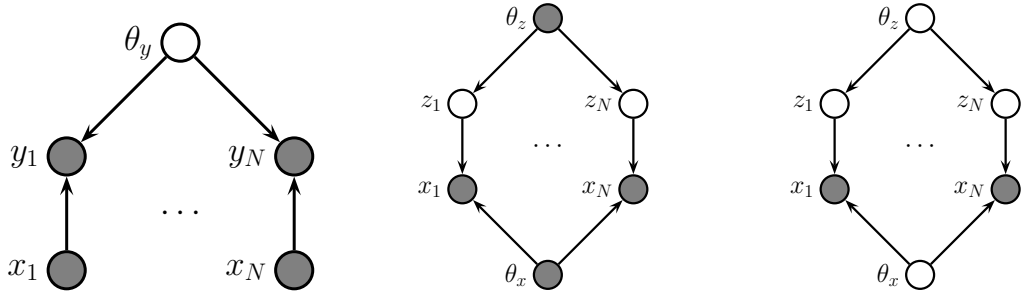


Figure 7.1: Graphical models with (a) global hidden variables for representing the Bayesian discriminative model $p(\boldsymbol{y}_{1:N}, \boldsymbol{\theta}_y|\boldsymbol{x}_{1:N}) = p(\boldsymbol{\theta}_y)\prod_{n=1}^N p(\boldsymbol{y}_n|\boldsymbol{x}_n;\boldsymbol{\theta}_y)$; (b) local hidden variables for representing the generative model $p(\boldsymbol{x}_{1:N}, \boldsymbol{z}_{1:N}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\boldsymbol{z}_n|\boldsymbol{\theta}_z)p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}_x)$; (c) local and global hidden variables for representing the Bayesian generative model $p(\boldsymbol{x}_{1:N}, \boldsymbol{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_z)p(\boldsymbol{\theta}_x)\prod_{n=1}^N p(\boldsymbol{z}_n|\boldsymbol{\theta}_z)p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}_x)$. Shaded nodes are assumed to be known (observed), unshaded nodes are hidden.

where $\boldsymbol{y}_{N+1}$ is the next observation after seeing the N examples in $\mathcal{D}$, and the posterior expected loss is computing using loss function ℓ and action a (see Section 34.1.3). Finally, if we define $g(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta}, M)$ for model M, we can also phrase the marginal likelihood (Section 3.8.3) as an expectation *wrt the prior*:

$$\mathbb{E}\left[g(\boldsymbol{\theta})|M\right] = \int g(\boldsymbol{\theta})p(\boldsymbol{\theta}|M)d\boldsymbol{\theta} = \int p(\mathcal{D}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)d\boldsymbol{\theta} = p(\mathcal{D}|M) \tag{7.9}$$

Thus we see that integration (and computing expectations) is at the heart of Bayesian inference, whereas differentiation is at the heart of optimization.

In this chapter, we give a high level summary of algorithmic techniques for computing (approximate) posteriors, and/or their corresponding expectations. We will give more details in the following chapters. Note that most of these methods are independent of the specific model. This allows problem solvers to focus on creating the best model possible for the task, and then relying on some inference engine to do the rest of the work — this latter process is sometimes called "**turning the Bayesian crank**". For more details on Bayesian computation, see e.g., [Gel+14a; MKL21; MFR20].

7.2 Common inference patterns

There are kinds of posterior we may want to compute, but we can identify 3 main patterns, as we discuss below. These give rise to different types of inference algorithm, as we will see in later chapters.

7.2.1 Global latents

The first pattern arises when we need to perform inference in models which have **global latent variables**, such as parameters of a model $\boldsymbol{\theta}$, which are shared across all N observed training cases. This is shown in Figure 7.1a, and corresponds to the usual setting for supervised or discriminative

learning, where the joint distribution has the form

$$p(\boldsymbol{y}_{1:N}, \boldsymbol{\theta}|\boldsymbol{x}_{1:N}) = p(\boldsymbol{\theta}) \left[\prod_{n=1}^{N} p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{\theta})\right] \tag{7.10}$$

The goal is to compute the posterior $p(\boldsymbol{\theta}|\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N})$. Most of the Bayesian supervised learning models discussed in Part III follow this pattern.

7.2.2 Local latents

The second pattern arises when we need to perform inference in models which have **local latent variables**, such as hidden states $\boldsymbol{z}_{1:N}$; we assume the model parameters $\boldsymbol{\theta}$ are known. This is shown in Figure 7.1b. Now the joint distribution has the form

$$p(\boldsymbol{x}_{1:N}, \boldsymbol{z}_{1:N}|\boldsymbol{\theta}) = \left[\prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}_x)p(\boldsymbol{z}_n|\boldsymbol{\theta}_z)\right] \tag{7.11}$$

The goal is to compute $p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})$ for each n. This is the setting we consider for most of the PGM inference methods in Chapter 9.

If the parameters are not known (which is the case for most latent variable models, such as mixture models), we may choose to estimate them by some method (e.g., maximum likelihood), and then plug in this point estimate. The advantage of this approach is that, conditional on $\boldsymbol{\theta}$, all the latent variables are conditionally independent, so we can perform inference in parallel across the data. This lets us use methods such as expectation maximization (Section 6.5.3), in which we infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta}_t)$ in the E step for all n simultaneously, and then update $\boldsymbol{\theta}_t$ in the M step. If the inference of $\boldsymbol{z}_n$ cannot be done exactly, we can use variational inference, a combination known as variational EM (Section 6.5.6.1).

Alternatively, we can use a minibatch approximation to the likelihood, marginalizing out $\boldsymbol{z}_n$ for each example in the minibatch to get

$$\log p(\mathcal{D}_t|\boldsymbol{\theta}_t) = \sum_{n \in \mathcal{D}_t} \log \left[\sum_{\boldsymbol{z}_n} p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\theta}_t)\right] \tag{7.12}$$

where $\mathcal{D}_t$ is the minibatch at step t. If the marginalization cannot be done exactly, we can use variational inference, a combination known as stochastic variational inference or SVI (Section 10.1.4). We can also learn an inference network $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta})$ to perform the inference for us, rather than running an inference engine for each example n in each batch t; the cost of learning $\boldsymbol{\phi}$ can be amortized across the batches. This is called amortized SVI (see Section 10.1.5).

7.2.3 Global and local latents

The third pattern arises when we need to perform inference in models which have **local and global latent variables**. This is shown in Figure 7.1c, and corresponds to the following joint distribution:

$$p(\boldsymbol{x}_{1:N}, \boldsymbol{z}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}_x)p(\boldsymbol{\theta}_z) \left[\prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}_x)p(\boldsymbol{z}_n|\boldsymbol{\theta}_z)\right] \tag{7.13}$$

This is essentially a Bayesian version of the latent variable model in Figure 7.1b, where now we model uncertainty in both the local variables $\boldsymbol{z}_n$ and the shared global variables $\boldsymbol{\theta}$. This approach is less common in the ML community, since it is often assumed that the uncertainty in the parameters $\boldsymbol{\theta}$ is negligible compared to the uncertainty in the local variables $\boldsymbol{z}_n$. The reason for this is that the parameters are "informed" by all N data cases, whereas each local latent $\boldsymbol{z}_n$ is only informed by a single datapoint, namely $\boldsymbol{x}_n$. Nevertheless, there are advantages to being "fully Bayesian", and modeling uncertainty in both local and global variables. We will see some examples of this later in the book.

7.3 Exact inference algorithms

In some cases, we can perform example posterior inference in a tractable manner. In particular, if the prior is **conjugate** to the likelihood, the posterior will be analytically tractable. In general, this will be the case when the prior and likelihood are from the same exponential family (Section 2.4). In particular, if the unknown variables are represented by $\boldsymbol{\theta}$, then we assume

$$p(\boldsymbol{\theta}) \propto \exp(\boldsymbol{\lambda}_0^\mathsf{T} \mathcal{T}(\boldsymbol{\theta})) \tag{7.14}$$

$$p(\boldsymbol{y}_i|\boldsymbol{\theta}) \propto \exp(\tilde{\boldsymbol{\lambda}}_i(\boldsymbol{y}_i)^\mathsf{T} \mathcal{T}(\boldsymbol{\theta})) \tag{7.15}$$

where $\mathcal{T}(\boldsymbol{\theta})$ are the sufficient statistics, and $\boldsymbol{\lambda}$ are the natural parameters. We can then compute the posterior by just adding the natural parameters:

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{1:N}) = \exp(\boldsymbol{\lambda}_*^\mathsf{T} \mathcal{T}(\boldsymbol{\theta})) \tag{7.16}$$

$$\boldsymbol{\lambda}_* = \boldsymbol{\lambda}_0 + \sum_{n=1}^{N} \tilde{\boldsymbol{\lambda}}_n(\boldsymbol{y}_n) \tag{7.17}$$

See Section 3.4 for details.

Another setting where we can compute the posterior exactly arises when the D unknown variables are all discrete, each with K states; in this case, the integral for the normalizing constant becomes a sum with K^D terms. In many cases, K^D will be too large to be tractable. However, if the distribution satisfies certain conditional independence properties, as expressed by a probabilistic graphical model (PGM), then we can write the joint as a product of local terms (see Chapter 4). This lets us use dynamic programming to make the computation tractable (see Chapter 9).

7.4 Approximate inference algorithms

For most probability models, we will not be able to compute marginals or posteriors exactly, so we must resort to using **approximate inference**. There are many different algorithms, which trade off speed, accuracy, simplicity, and generality. We briefly discuss some of these algorithms below, and give more detail in the following chapters. (See also [Alq22; MFR20] for a review of various methods.)

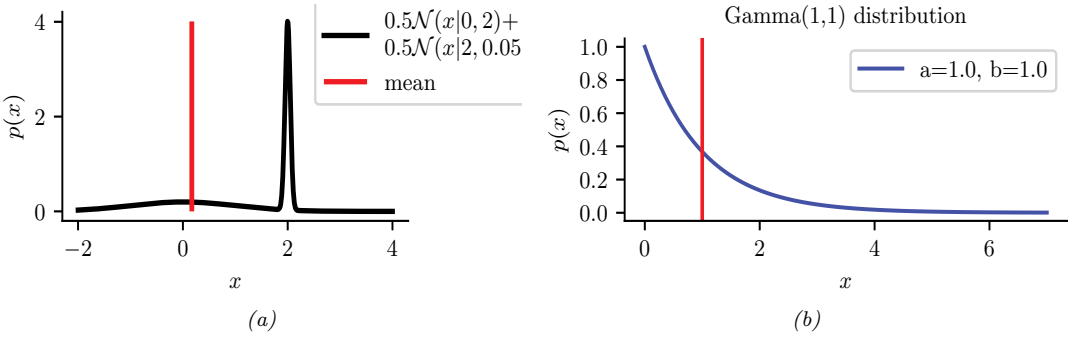


Figure 7.2: Two distributions in which the mode (highest point) is untypical of the distribution; the mean (vertical red line) is a better summary. (a) A bimodal distribution. Generated by bimodal_dist_plot.ipynb. (b) A skewed $\text{Ga}(1,1)$ *distribution. Generated by gamma_dist_plot.ipynb.*

7.4.1 The MAP approximation and its problems

The simplest approximate inference method is to compute the MAP estimate

$$\hat{\boldsymbol{\theta}} = \text{argmax}\, p(\boldsymbol{\theta}|\mathcal{D}) = \text{argmax} \log p(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta}) \tag{7.18}$$

and then to assume that the posterior puts 100% of its probability on this single value:

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \tag{7.19}$$

The advantage of this approach is that we can compute the MAP estimate using a variety of optimization algorithms, which we discuss in Chapter 6. However, the MAP estimate also has various drawbacks, some of which we discuss below.

7.4.1.1 The MAP estimate gives no measure of uncertainty

In many statistical applications (especially in science) it is important to know how much one can trust a given parameter estimate. Obviously a point estimate does not convey any notion of uncertainty. Although it is possible to derive frequentist notions of uncertainty from a point estimate (see Section 3.3.1), it is arguably much more natural to just compute the posterior, from which we can derive useful quantities such as the standard error (see Section 3.2.1.6) and credible regions (see Section 3.2.1.7).

In the context of prediction (which is the main focus in machine learning), we saw in Section 3.2.2 that plugging in a point estimate can underestimate the predictive uncertainty, which can result in predictions which are not just wrong, but confidently wrong. It is generally considered very important for a predictive model to "know what it does not know", and the Bayesian approach is a good strategy for achieving this goal.

7.4.1.2 The MAP estimate is often untypical of the posterior

In some cases, we may not be interested in uncertainty, and instead we just want a single summary of the posterior. However, the mode of a posterior distribution is often a very poor choice as a

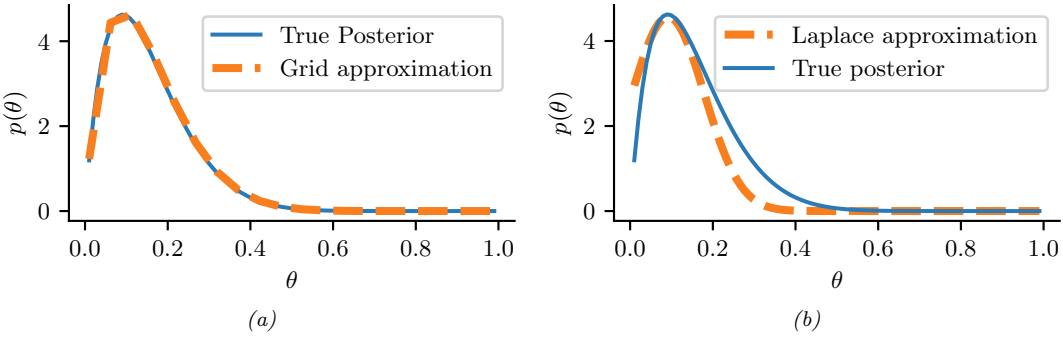


Figure 7.3: Approximating the posterior of a beta-Bernoulli model. (a) Grid approximation using 20 grid points. (b) Laplace approximation. Generated by laplace_approx_beta_binom.ipynb.

summary statistic, since the mode is usually quite untypical of the distribution, unlike the mean or median. This is illustrated in Figure 7.2(a) for a 1d continuous space, where we see that the mode is an isolated peak (black line), far from most of the probability mass. By contrast, the mean (red line) is near the middle of the distribution.

Another example is shown in Figure 7.2(b): here the mode is 0, but the mean is non-zero. Such skewed distributions often arise when inferring variance parameters, especially in hierarchical models. In such cases the MAP estimate (and hence the MLE) is obviously a very bad estimate.

7.4.1.3 The MAP estimate is not invariant to reparameterization

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

For example, let $\hat{x} = \text{argmax}_x \, p_x(x)$ be the MAP estimate for x. Now let $y = f(x)$ be a transformation of x. In general it is not the case that $\hat{y} = \text{argmax}_y \, p_y(y)$ is given by $f(\hat{x})$. For example, let $x \sim \mathcal{N}(6, 1)$ and $y = f(x)$, where $f(x) = \frac{1}{1+\exp(-x+5)}$. We can use the change of variables (Section 2.5.1) to conclude $p_y(y) = p_x(f^{-1}(y))|\frac{df^{-1}(y)}{dy}|$. Alternatively we can use a Monte Carlo approximation. The result is shown in Figure 2.12. We see that the original Gaussian for $p(x)$ has become "squashed" by the sigmoid nonlinearity. In particular, we see that the mode of the transformed distribution is not equal to the transform of the original mode.

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

7.4.2 Grid approximation

If we want to capture uncertainty, we need to allow for the fact that $\boldsymbol{\theta}$ may have a range of possible values, each with non-zero probability. The simplest way to capture this property is to partition

the space of possible values into a finite set of regions, call them $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_K$, each representing a region of parameter space of volume Δ centered on $\boldsymbol{\theta}_k$. This is called a **grid approximation**. The probability of being in each region is given by $p(\boldsymbol{\theta} \in \boldsymbol{r}_k|\mathcal{D}) \approx p_k \Delta$, where

$$p_k = \frac{\tilde{p}_k}{\sum_{k'=1}^{K} \tilde{p}_{k'}} \tag{7.20}$$

$$\tilde{p}_k = p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k) \tag{7.21}$$

As K increases, we decrease the size of each grid cell. Thus the denominator is just a simple numerical approximation of the integral

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \sum_{k=1}^{K} \Delta \tilde{p}_k \tag{7.22}$$

As a simple example, we will use the problem of approximating the posterior of a beta-Bernoulli model. Specifically, the goal is to approximate

$$p(\theta|\mathcal{D}) \propto \left[\prod_{n=1}^{N} \text{Ber}(y_n|\theta)\right] \text{Beta}(1,1) \tag{7.23}$$

where $\mathcal{D}$ consists of 10 heads and 1 tail (so the total number of observations is $N = 11$), with a uniform prior. Although we can compute this posterior exactly using the method discussed in Section 3.4.1, this serves as a useful pedagogical example since we can compare the approximation to the exact answer. Also, since the target distribution is just 1d, it is easy to visualize the results.

In Figure 7.3a, we illustrate the grid approximation applied to our 1d problem. We see that it is easily able to capture the skewed posterior (due to the use of an imbalanced sample of 10 heads and 1 tail). Unfortunately, this approach does not scale to problems in more than 2 or 3 dimensions, because the number of grid points grows exponentially with the number of dimensions.

7.4.3 Laplace (quadratic) approximation

In this section, we discuss a simple way to approximate the posterior using a multivariate Gaussian; this known as a **Laplace approximation** or **quadratic approximation** (see e.g., [TK86; RMC09]).

Suppose we write the posterior as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} e^{-\mathcal{E}(\boldsymbol{\theta})} \tag{7.24}$$

where $\mathcal{E}(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}, \mathcal{D})$ is called an energy function, and $Z = p(\mathcal{D})$ is the normalization constant. Performing a Taylor series expansion around the mode $\hat{\boldsymbol{\theta}}$ (i.e., the lowest energy state) we get

$$\mathcal{E}(\boldsymbol{\theta}) \approx \mathcal{E}(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^{\mathsf{T}} \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^{\mathsf{T}} \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \tag{7.25}$$

where $\boldsymbol{g}$ is the gradient at the mode, and $\mathbf{H}$ is the Hessian. Since $\hat{\boldsymbol{\theta}}$ is the mode, the gradient term is

zero. Hence

$$\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})} \exp\left[-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^{\mathsf{T}} \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right] \tag{7.26}$$

$$\hat{p}(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z}\hat{p}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}, \mathbf{H}^{-1}) \tag{7.27}$$

$$Z = e^{-\mathcal{E}(\hat{\boldsymbol{\theta}})}(2\pi)^{D/2}|\mathbf{H}|^{-\frac{1}{2}} \tag{7.28}$$

The last line follows from normalization constant of the multivariate Gaussian.

The Laplace approximation is easy to apply, since we can leverage existing optimization algorithms to compute the MAP estimate, and then we just have to compute the Hessian at the mode. (In high dimensional spaces, we can use a diagonal approximation.)

In Figure 7.3b, we illustrate this method applied to our 1d problem. Unfortunately we see that it is not a particularly good approximation. This is because the posterior is skewed, whereas a Gaussian is symmetric. In addition, the parameter of interest lies in the constrained interval $\theta \in [0, 1]$, whereas the Gaussian assumes an unconstrained space, $\boldsymbol{\theta} \in \mathbb{R}^D$. Fortunately, we can solve this latter problem by using a change of variable. For example, in this case we can apply the Laplace approximation to $\alpha = \text{logit}(\theta)$. This is a common trick to simplify the job of inference.

See Section 15.3.5 for an application of Laplace approximation to Bayesian logistic regression, and Section 17.3.2 for an application of Laplace approximation to Bayesian neural networks.

7.4.4 Variational inference

In Section 7.4.3, we discussed the Laplace approximation, which uses an optimization procedure to find the MAP estimate, and then approximates the curvature of the posterior at that point based on the Hessian. In this section, we discuss **variational inference** (**VI**), also called **variational Bayes** (**VB**). This is another optimization-based approach to posterior inference, but which has much more modeling flexibility (and thus can give a much more accurate approximation).

VI attempts to approximate an intractable probability distribution, such as $p(\boldsymbol{\theta}|\mathcal{D})$, with one that is tractable, $q(\boldsymbol{\theta})$, so as to minimize some discrepancy D between the distributions:

$$q^* = \underset{q \in \mathcal{Q}}{\text{argmin}}\, D(q, p) \tag{7.29}$$

where $\mathcal{Q}$ is some tractable family of distributions (e.g., fully factorized distributions). Rather than optimizing over functions q, we typically optimize over the parameters of the function q; we denote these **variational parameters** by $\boldsymbol{\psi}$.

It is common to use the KL divergence (Section 5.1) as the discrepancy measure, which is given by

$$D(q, p) = D_{\mathbb{KL}}\left(q(\boldsymbol{\theta}|\boldsymbol{\psi}) \parallel p(\boldsymbol{\theta}|\mathcal{D})\right) = \int q(\boldsymbol{\theta}|\boldsymbol{\psi}) \log \frac{q(\boldsymbol{\theta}|\boldsymbol{\psi})}{p(\boldsymbol{\theta}|\mathcal{D})} d\boldsymbol{\theta} \tag{7.30}$$

where $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. The inference problem then reduces to the following optimization

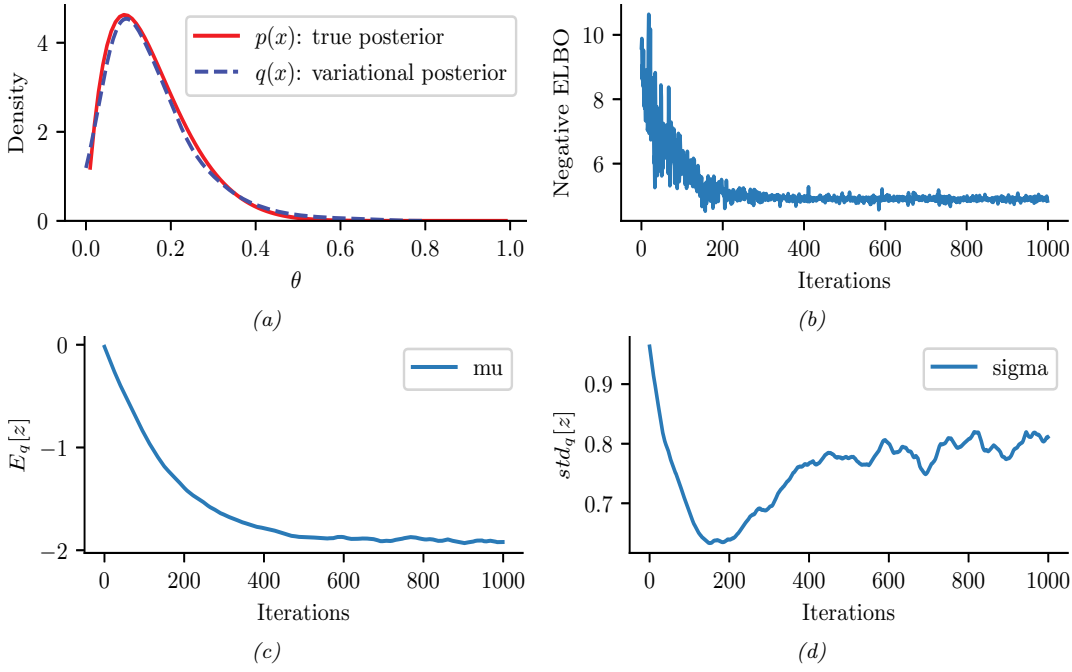


Figure 7.4: ADVI applied to the beta-Bernoulli model. (a) Approximate vs true posterior. (b) Negative ELBO over time. (c) Variational μ parameter over time. (d) Variational σ parameter over time. Generated by advi_beta_binom.ipynb.

problem:

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} D_{\mathbb{KL}}\left(q(\boldsymbol{\theta}|\boldsymbol{\psi}) \parallel p(\boldsymbol{\theta}|\mathcal{D})\right) \tag{7.31}$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})}\left[\log q(\boldsymbol{\theta}|\boldsymbol{\psi}) - \log\left(\frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}\right)\right] \tag{7.32}$$

$$= \underset{\boldsymbol{\psi}}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})}\left[-\log p(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{\psi})\right]}_{-Ł(\boldsymbol{\psi})} + \log p(\mathcal{D}) \tag{7.33}$$

Note that $\log p(\mathcal{D})$ is independent of $\boldsymbol{\psi}$, so we can ignore it when fitting the approximate posterior, and just focus on maximizing the term

$$Ł(\boldsymbol{\psi}) \triangleq \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})}\left[\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\psi})\right] \tag{7.34}$$

Since we have $D_{\mathbb{KL}}(q \parallel p) \geq 0$, we have $Ł(\boldsymbol{\psi}) \leq \log p(\mathcal{D})$. The quantity $\log p(\mathcal{D})$, which is the log marginal likelihood, is also called the **evidence**. Hence $Ł(\boldsymbol{\psi})$ is known as the **evidence lower bound** or **ELBO**. By maximizing this bound, we are making the variational posterior closer to the true posterior. (See Section 10.1 for details.)

We can choose any kind of approximate posterior that we like. For example, we may use a Gaussian, $q(\boldsymbol{\theta}|\boldsymbol{\psi}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. This is different from the Laplace approximation, since in VI, we optimize

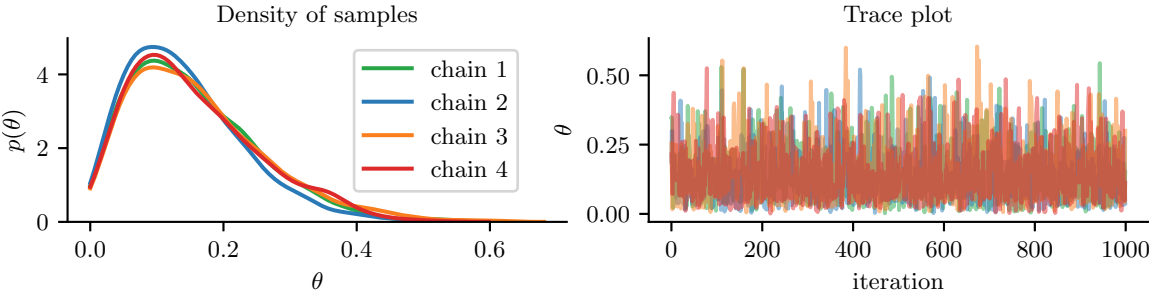


Figure 7.5: Approximating the posterior of a beta-Bernoulli model using MCMC. (a) Kernel density estimate derived from samples from 4 independent chains. (b) Trace plot of the chains as they generate posterior samples. Generated by hmc_beta_binom.ipynb.

$\boldsymbol{\Sigma}$, rather than equating it to the Hessian. If $\boldsymbol{\Sigma}$ is diagonal, we are assuming the posterior is fully factorized; this is called a **mean field** approximation.

A Gaussian approximation is not always suitable for all parameters. For example, in our 1d example we have the constraint that $\theta \in [0, 1]$. We could use a variational approximation of the form $q(\theta|\boldsymbol{\psi}) = \text{Beta}(\theta|a, b)$, where $\boldsymbol{\psi} = (a, b)$. However choosing a suitable form of variational distribution requires some level of expertise. To create a more easily applicable, or "turn-key", method, that works on a wide range of models, we can use a method called **automatic differentiation variational inference** or **ADVI** [Kuc+16]. This uses the change of variables method to convert the parameters to an unconstrained form, and then computes a Gaussian variational approximation. The method also uses automatic differentiation to derive the Jacobian term needed to compute the density of the transformed variables. See Section 10.2.2 for details.

We now apply ADVI to our 1d beta-Bernoulli model. Let $\theta = \sigma(z)$, where we replace $p(\theta|\mathcal{D})$ with $q(z|\boldsymbol{\psi}) = \mathcal{N}(z|\mu, \sigma)$, where $\boldsymbol{\psi} = (\mu, \sigma)$. We optimize a stochastic approximation to the ELBO using SGD. The results are shown in Figure 7.4 and seem reasonable.

7.4.5 Markov chain Monte Carlo (MCMC)

Although VI is fast, it can give a biased approximation to the posterior, since it is restricted to a specific function form $q \in \mathcal{Q}$. A more flexible approach is to use a non-parametric approximation in terms of a set of samples, $q(\boldsymbol{\theta}) \approx \frac{1}{S}\sum_{s=1}^{S} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^s)$. This is called a **Monte Carlo approximation**. The key issue is how to create the posterior samples $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$ efficiently, without having to evaluate the normalization constant $p(\mathcal{D}) = \int p(\boldsymbol{\theta}, \mathcal{D}) d\boldsymbol{\theta}$.

For low dimensional problems, we can use methods such as **importance sampling**, which we discuss in Section 11.5. However, for high dimensional problems, it is more common to use **Markov chain Monte Carlo** or **MCMC**. We give the details in Chapter 12, but give a brief introduction here.

The most common kind of MCMC is known as the **Metropolis-Hastings algorithm**. The basic idea behind MH is as follows: we start at a random point in parameter space, and then perform a random walk, by sampling new states (parameters) from a **proposal distribution** $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$. If q is chosen carefully, the resulting Markov chain distribution will satisfy the property that the fraction of

time we visit each point in space is proportional to the posterior probability. The key point is that to decide whether to move to a newly proposed point $\boldsymbol{\theta}'$ or to stay in the current point $\boldsymbol{\theta}$, we only need to evaluate the unnormalized density ratio

$$\frac{p(\boldsymbol{\theta}|\mathcal{D})}{p(\boldsymbol{\theta}'|\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}')p(\boldsymbol{\theta}')/p(\mathcal{D})} = \frac{p(\mathcal{D},\boldsymbol{\theta})}{p(\mathcal{D},\boldsymbol{\theta}')} \tag{7.35}$$

This avoids the need to compute the normalization constant $p(\mathcal{D})$. (In practice we usually work with log probabilities, instead of joint probabilities, to avoid numerical issues.)

We see that the input to the algorithm is just a function that computes the log joint density, $\log p(\boldsymbol{\theta}, \mathcal{D})$, as well as a proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ for deciding which states to visit next. It is common to use a Gaussian distribution for the proposal, $q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}'|\boldsymbol{\theta}, \sigma\mathbf{I})$; this is called the **random walk Metropolis** algorithm. However, this can be very inefficient, since it is blindly walking through the space, in the hopes of finding higher probability regions.

In models that have conditional independence structure, it is often easy to compute the **full conditionals** $p(\boldsymbol{\theta}_d|\boldsymbol{\theta}_{-d}, \mathcal{D})$ for each variable d, one at a time, and then sample from them. This is like a stochastic analog of coordinate ascent, and is called **Gibbs sampling** (see Section 12.3 for details).

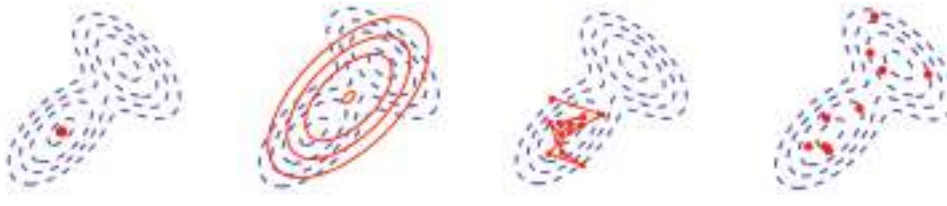
For models where all unknown variables are continuous, we can often compute the gradient of the log joint, $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D})$. We can use this gradient information to guide the proposals into regions of space with higher probability. This approach is called **Hamiltonian Monte Carlo** or **HMC**, and is one of the most widely used MCMC algorithms due to its speed. For details, see Section 12.5.

We apply HMC to our beta-Bernoulli model in Figure 7.5. (We use a logit transformation for the parameter.) In panel b, we show samples generated by the algorithm from 4 parallel Markov chains. We see that they oscillate around the true posterior, as desired. In panel a, we compute a kernel density estimate from the posterior samples from each chain; we see that the result is a good approximation to the true posterior in Figure 7.3.

7.4.6 Sequential Monte Carlo

MCMC is like a stochastic local search algorithm, in that it makes moves through the state space of the posterior distribution, comparing the current value to proposed neighboring values. An alternative approach is to use perform inference using a sequence of different distributions, from simpler to more complex, with the final distribution being equal to the target posterior. This is called **sequential Monte Carlo** or **SMC**. This approach, which is more similar to tree search than local search, has various advantages over MCMC, which we discuss in Chapter 13.

A common application of SMC is to **sequential Bayesian inference**, in which we recursively compute (i.e., in an online fashion) the posterior $p(\boldsymbol{\theta}_t|\mathcal{D}_{1:t})$, where $\mathcal{D}_{1:t} = \{(\boldsymbol{x}_n, y_n) : n = 1 : t\}$ is all the data we have seen so far. This sequence of distributions converges to the full batch posterior $p(\boldsymbol{\theta}|\mathcal{D})$ once all the data has been seen. However, the approach can also be used when the data is arriving in a continual, unending stream, as in state-space models (see Chapter 29). The application of SMC to such dynamical models is known as **particle filtering**. See Section 13.2 for details.



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

Figure 7.6: Different approximations to a bimodal 2d distribution. (a) Local MAP estimate. (b) Parametric Gaussian approximation. (c) Correlated samples from near one mode. (d) Independent samples from the distribution. Adapted from Figure 2 of [PY14]. Used with kind permission of George Panadreou.

7.4.7 Challenging posteriors

In many applications, the posterior can be high dimensional and multimodal. Approximating such distributions can be quite challenging. In Figure 7.6, we give a simple 2d example. We compare MAP estimation (which does not capture any uncertainty), a Gaussian parametric approximation such as the Laplace approximation or variational inference (see panel b), and a nonparametric approximation in terms of samples. If the samples are generated from MCMC, they are serially correlated, and may only explore a local model (see panel c). However, ideally we can draw independent samples from the entire support of the distribution, as shown in panel d. We may also be able to fit a local parametric approximation around each such sample (see Section 17.3.9.1), to get a semi-parametric approximation to the posterior.

7.5 Evaluating approximate inference algorithms

There are many different approximate inference algorithms, each of which make different tradeoffs between speed, accuracy, generality, simplicity, etc. This makes it hard to compare them on an equal footing.

One approach is to evaluate the accuracy of the approximation $q(\boldsymbol{\theta})$ by comparing to the "true" posterior $p(\boldsymbol{\theta}|\mathcal{D})$, computed offline with an "exact" method. We are usually interested in accuracy vs speed tradeoffs, which we can compute by evaluating $D_{\mathbb{KL}}\left(p(\boldsymbol{\theta}|\mathcal{D}) \parallel q_t(\boldsymbol{\theta})\right)$, where $q_t(\boldsymbol{\theta})$ is the approximate posterior after t units of compute time. Of course, we could use other measures of distributional similarity, such as Wasserstein distance.

Unfortunately, it is usually impossible to compute the true posterior $p(\boldsymbol{\theta}|\mathcal{D})$. A simple alternative is to evaluate the quality in terms of its prediction abilities on out of sample observed data, similar to cross validation. More generally, we can compare the expected loss or Bayesian risk (Section 34.1.3) of different posteriors, as proposed in [KPS98; KPS99]:

$$R = \mathbb{E}_{p^*(\boldsymbol{x},\boldsymbol{y})}\left[\ell(\boldsymbol{y}, q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}))\right] \text{ where } q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}) = \int p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}) q(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \tag{7.36}$$

where $\ell(\boldsymbol{y}, q(\boldsymbol{y}))$ is some loss function, such as log-loss. Alternatively, we can measure performance of the posterior when it is used in some downstream task, such as continual or active learning, as proposed in [Far22].

For some specialized methods for assessing variational inference, see [Yao+18b; Hug+20], and for Monte Carlo methods, see [CGR06; CTM17; GAR16].

8 Gaussian filtering and smoothing

8.1 Introduction

In this chapter, we consider the task of posterior inference in **state-space models** (SSMs). We discuss SSMs in more detail in Chapter 29, but we can think of them as latent variable sequence models with the conditional independencies shown by the chain-structured graphical model Figure 8.1. The corresponding joint distribution has the form

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{u}_{1:T}) = \left[p(\boldsymbol{z}_1|\boldsymbol{u}_1)\prod_{t=2}^{T} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{u}_t)\right]\left[\prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{z}_t, \boldsymbol{u}_t)\right] \tag{8.1}$$

where $\boldsymbol{z}_t$ are the hidden variables at time t, $\boldsymbol{y}_t$ are the observations (outputs), and $\boldsymbol{u}_t$ are the optional inputs. The term $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{u}_t)$ is called the **dynamics model** or **transition model**, $p(\boldsymbol{y}_t|\boldsymbol{z}_t, \boldsymbol{u}_t)$ is called the **observation model** or **measurement model**, and $p(\boldsymbol{z}_1|\boldsymbol{u}_1)$ is the prior or initial state distribution.[1]

8.1.1 Inferential goals

Given the sequence of observations, and a known model, one of the main tasks with SSMs is to perform posterior inference about the hidden states; this is also called **state estimation**.

For example, consider an airplane flying in the sky. (For simplicity, we assume the world is 2d, not 3d.) We would like to estimate its location and velocity $\boldsymbol{z}_t \in \mathbb{R}^4$ given noisy sensor measurements of its location $\boldsymbol{y}_t \in \mathbb{R}^2$, as illustrated in Figure 8.2(a). (We ignore the inputs $\boldsymbol{u}_t$ for simplicity.)

We discuss a suitable SSM for this problem, that embodies Newton's laws of motion, in Section 8.2.1.1. We can use the model to compute the **belief state** $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$; this is called Bayesian **filtering**. If we represent the belief state using a Gaussian, then we can use the **Kalman filter** to solve this task, as we discuss in Section 8.2.2. In Figure 8.2(b) we show the results of this algorithm. The green dots are the noisy observations, the red line shows the posterior mean estimate of the location, and the black circles show the posterior covariance. (The posterior over the velocity is not shown.) We see that the estimated trajectory is less noisy than the raw data, since it incorporates prior knowledge about how the data was generated.

Another task of interest is the **smoothing** problem where we want to compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})$ using an offline dataset. We can compute these quantities using the **Kalman smoother** described in

1. In some cases, the initial state distribution is denoted by $p(\boldsymbol{z}_0)$, and then we derive $p(\boldsymbol{z}_1|\boldsymbol{u}_1)$ by passing $p(\boldsymbol{z}_0)$ through the dynamics model. In this case, the joint distribution represents $p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{0:T}|\boldsymbol{u}_{1:T})$.

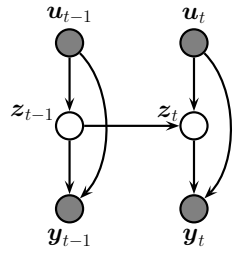
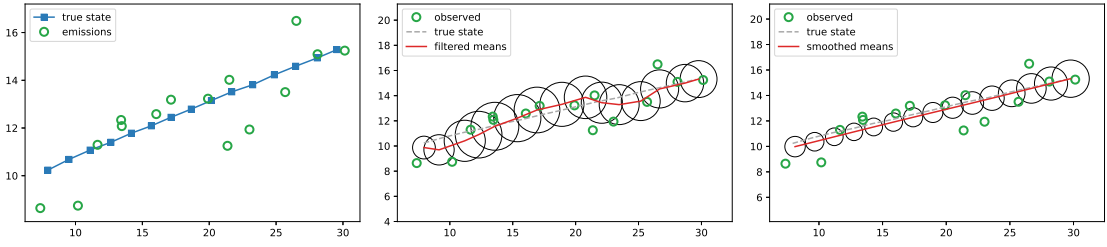


Figure 8.1: A state-space model represented as a graphical model. z_t are the hidden variables at time t, y_t are the observations (outputs), and u_t are the optional inputs.



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

Figure 8.2: Illustration of Kalman filtering and smoothing for a linear dynamical system. (a) Observations (green cirles) are generated by an object moving to the right (true location denoted by blue squares). (b) Results of online Kalman filtering. Circles are 95% confidence ellipses, whose center is the posterior mean, and whose shape is derived from the posterior covariance. (c) Same as (b), but using offline Kalman smoothing. The MSE in the trajectory for filtering is 3.13, and for smoothing is 1.71. Generated by kf_tracking_script.ipynb.

Section 8.2.3. In Figure 8.2(c) we show the result of this algorithm. We see that the resulting estimate is smoother compared to filtering, and that the posterior uncertainty is reduced (as visualized by the smaller confidence ellipses).

To understand this behavior intuitively, consider a detective trying to figure out who committed a crime. As they move through the crime scene, their uncertainty is high until he finds the key clue; then they have an "aha" moment, the uncertainty is reduced, and all the previously confusing observations are, in **hindsight**, easy to explain. Thus we see that, given all the data (including finding the clue), it is much easier to infer the state of the world.

A disadvantage of the smoothing method is that we have to wait until all the data has been observed before we start performing inference, so it cannot be used for online or realtime problems. **Fixed lag smoothing** is a useful compromise between online and offline estimation; it involves computing $p(\boldsymbol{z}_{t-\ell}|\boldsymbol{y}_{1:t})$, where $\ell > 0$ is called the lag. This gives better performance than filtering, but incurs a slight delay. By changing the size of the lag, we can trade off accuracy vs delay. See Figure 8.3 for an illustration.

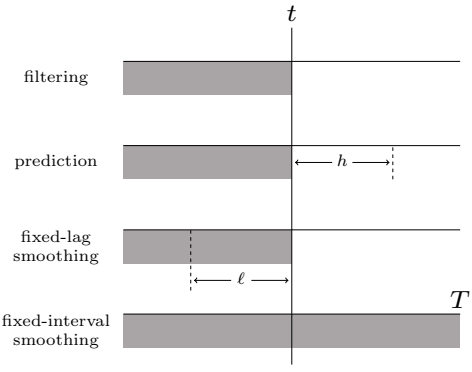


Figure 8.3: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, T is the sequence length, ℓ is the lag, and h is the prediction horizon. Used with kind permission of Peter Chang.

In addition to infering the latent state, we may want to predict future observations. We can compute the **observed predictive distribution** h steps into the future as follows:

$$p(\boldsymbol{y}_{t+h}|\boldsymbol{y}_{1:t}) = \sum_{\boldsymbol{z}_{t+h}} p(\boldsymbol{y}_{t+h}|\boldsymbol{z}_{t+h})p(\boldsymbol{z}_{t+h}|\boldsymbol{y}_{1:t}) \tag{8.2}$$

where the **hidden state predictive distribution** is obtained by pushing the current belief state through the dynamics model

$$p(\boldsymbol{z}_{t+h}|\boldsymbol{y}_{1:t}) = \sum_{\boldsymbol{z}_{t:t+h-1}} p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)p(\boldsymbol{z}_{t+2}|\boldsymbol{z}_{t+1}) \cdots p(\boldsymbol{z}_{t+h}|\boldsymbol{z}_{t+h-1}) \tag{8.3}$$

(When the states are continuous, we need to replace the sums with integrals.)

8.1.2 Bayesian filtering equations

The **Bayes filter** is an algorithm for recursively computing the **belief state** $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ given the prior belief from the previous step, $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})$, the new observation $\boldsymbol{y}_t$, and the model. This can be done using **sequential Bayesian updating**, and requires a constant amount of computation per time step (independent of t). For a dynamical model, this reduces to the **predict-update** cycle described below.

The **prediction step** is just the **Chapman-Kolmogorov equation**:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}_{t-1} \tag{8.4}$$

The prediction step computes the one-step-ahead predictive distribution for the latent state, which

updates the posterior from the previous time step into the prior for the current step.[2]

The **update step** is just Bayes' rule:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \frac{1}{Z_t} p(\boldsymbol{y}_t|\boldsymbol{z}_t) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) \tag{8.5}$$

where the normalization constant is

$$Z_t = \int p(\boldsymbol{y}_t|\boldsymbol{z}_t) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) d\boldsymbol{z}_t = p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) \tag{8.6}$$

We can use the normalization constants to compute the log likelihood of the sequence as follows:

$$\log p(\boldsymbol{y}_{1:T}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \sum_{t=1}^{T} \log Z_t \tag{8.7}$$

where we define $p(\boldsymbol{y}_1|\boldsymbol{y}_0) = p(\boldsymbol{y}_1)$. This quantity is useful for computing the MLE of the parameters.

8.1.3 Bayesian smoothing equations

In the offline setting, we want to compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})$, which is the belief about the hidden state at time t given all the data, both past and future. This is called (fixed interval) **smoothing**. We first perform the forwards or filtering pass, and then compute the smoothed belief states by working backwards, from right (time $t = T$) to left ($t = 1$), as we explain below. Hence this method is also called **forwards filtering backwards smoothing** or **FFBS**.

Suppose, by induction, that we have already computed $p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})$. We can convert this into a joint smoothed distribution over two consecutive time steps using

$$p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) \tag{8.8}$$

To derive the first term, note that from the Markov properties of the model, and Bayes' rule, we have

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:t}, \boldsymbol{y}_{t+1:T}) \tag{8.9}$$

$$= \frac{p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} \tag{8.10}$$

$$= \frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} \tag{8.11}$$

Thus the joint distribution over two consecutive time steps is given by

$$p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) = \frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} \tag{8.12}$$

2. The prediction step is not needed at $t = 1$ if $p(\boldsymbol{z}_1)$ is provided as input to the model. However, if we just provide $p(\boldsymbol{z}_0)$, we need to compute $p(\boldsymbol{z}_1|\boldsymbol{y}_{1:0}) = p(\boldsymbol{z}_1)$ by applying the prediction step.

from which we get the new smoothed marginal distribution:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) \int \left[\frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})}\right] d\boldsymbol{z}_{t+1} \tag{8.13}$$

$$= \int p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}) \frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} d\boldsymbol{z}_{t+1} \tag{8.14}$$

Intuitively we can interpret this as follows: we start with the two-slice filtered distribution, $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})$, and then we divide out the old $p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})$ and multiply in the new $p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})$, and then marginalize out $\boldsymbol{z}_{t+1}$.

8.1.4 The Gaussian ansatz

In general, computing the integrals required to implement Bayesian filtering and smoothing is intractable. However, there are two notable exceptions: if the state space is discrete, as in an HMM, we can represent the belief states as discrete distributions (histograms), which we can update using the forwards-backwards algorithm, as discussed in Section 9.2; and if the SSM is a linear-Gaussian model, then we can represent the belief states by Gaussians, which we can update using the Kalman filter and smoother, which we discuss in Section 8.2.2 and Section 8.2.3. In the nonlinear and/or non-Gaussian setting, we can still use a Gaussian to represent an approximate belief state, as we discuss in Section 8.3, Section 8.4, Section 8.5 and Section 8.6. We discuss some non-Gaussian approximations in Section 8.7.

For most of this chapter, we assume the SSM can be written as a nonlinear model subject to additive Gaussian noise:

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{f}(\boldsymbol{z}_{t-1}, \boldsymbol{u}_t) + \mathcal{N}(\boldsymbol{0}, \mathbf{Q}_t) \\ \boldsymbol{y}_t &= \boldsymbol{h}(\boldsymbol{z}_t, \boldsymbol{u}_t) + \mathcal{N}(\boldsymbol{0}, \mathbf{R}_t) \end{aligned} \tag{8.15}$$

where $\boldsymbol{f}$ is the transition or dynamics function, and $\boldsymbol{h}$ is the observation function. In some cases, we will further assume that these functions are linear.

8.2 Inference for linear-Gaussian SSMs

In this section, we discuss inference in SSMs where all the distributions are linear Gaussian. This is called a **linear Gaussian state space model** (**LG-SSM**) or a **linear dynamical system** (**LDS**). We discuss such models in detail in Section 29.6, but in brief they have the following form:

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{z}_{t-1} + \mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \mathbf{Q}_t) \tag{8.16}$$

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_t\boldsymbol{z}_t + \mathbf{D}_t\boldsymbol{u}_t + \boldsymbol{d}_t, \mathbf{R}_t) \tag{8.17}$$

where $\boldsymbol{z}_t \in \mathbb{R}^{N_z}$ is the hidden state, $\boldsymbol{y}_t \in \mathbb{R}^{N_y}$ is the observation, and $\boldsymbol{u}_t \in \mathbb{R}^{N_u}$ is the input. (We have allowed the parameters to be time-varying, for later extensions that we will consider.) We often assume the means of the process noise and observation noise (i.e., the bias or offset terms) are zero, so $\boldsymbol{b}_t = \boldsymbol{0}$ and $\boldsymbol{d}_t = \boldsymbol{0}$. In addition, we often have no inputs, so $\mathbf{B}_t = \mathbf{D}_t = \boldsymbol{0}$. In this case, the model

simplifies to the following:[3]

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{z}_{t-1}, \mathbf{Q}_t) \tag{8.18}$$

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_t\boldsymbol{z}_t, \mathbf{R}_t) \tag{8.19}$$

See Figure 8.1 for the graphical model.[4]

Note that an LG-SSM is just a special case of a Gaussian Bayes net (Section 4.2.3), so the entire joint distribution $p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{u}_{1:T})$ is a large multivariate Gaussian with $N_y N_z T$ dimensions. However, it has a special structure that makes it computationally tractable to use, as we show below. In particular, we will discuss the **Kalman filter** and **Kalman smoother**, that can perform exact filtering and smoothing in $O(TN_z^3)$ time.

8.2.1 Examples

Before diving into the theory, we give some motivating examples.

8.2.1.1 Tracking and state estimation

A common application of LG-SSMs is for **tracking** objects, such as airplanes or animals, from noisy measurements, such as radar or cameras. For example, suppose we want to track an object moving in 2d. (We discuss this example in more detail in Section 29.7.1.) The hidden state $\boldsymbol{z}_t$ encodes the location, (x_{t1}, x_{t2}), and the velocity, $(\dot{x}_{t1}, \dot{x}_{t1})$, of the moving object. The observation $\boldsymbol{y}_t$ is a noisy version of the location. (The velocity is not observed but can be inferred from the change in location.) We assume that we obtain measurements with a sampling period of Δ. The new location is the old location plus Δ times the velocity, plus noise added to all terms:

$$\boldsymbol{z}_t = \underbrace{\begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{F}} \boldsymbol{z}_{t-1} + \boldsymbol{q}_t \tag{8.20}$$

where $\boldsymbol{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$. The observation extracts the location and adds noise:

$$\boldsymbol{y}_t = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{\mathbf{H}} \boldsymbol{z}_t + \boldsymbol{r}_t \tag{8.21}$$

where $\boldsymbol{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$.

Our goal is to use this model to estimate the unknown location (and velocity) of the object given the noisy observations. In particular, in the filtering problem, we want to compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ in

3. Our notation is similar to [SS23], except he writes $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) = \mathcal{N}(\boldsymbol{x}_k|\mathbf{A}_{k-1}\boldsymbol{x}_{k-1}, \mathbf{Q}_{k-1})$ instead of $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{z}_{t-1}, \mathbf{Q}_t)$, and $p(\boldsymbol{y}_k|\boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{y}_k|\mathbf{H}_k\boldsymbol{x}_k, \mathbf{R}_k)$ instead of $p(\boldsymbol{y}_t|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_t\boldsymbol{z}_t, \mathbf{R}_t)$.

4. Note that, for some problems, the evolution of certain components of the state vector is deterministic, in which case the corresponding noise terms must be zero. To avoid singular covariance matrices, we can replace the dynamics noise $\boldsymbol{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ with $\mathbf{G}_t\tilde{\boldsymbol{w}}_t$, where $\tilde{\boldsymbol{w}}_t \sim \mathcal{N}(\mathbf{0}, \tilde{\mathbf{Q}}_t)$, where $\tilde{\mathbf{Q}}_t$ is a smaller $N_q \times N_q$ psd matrix, and $\mathbf{G}_t$ is a $N_y \times N_q$. In this case, the covariance of the noise becomes $\mathbf{Q}_t = \mathbf{G}_t\tilde{\mathbf{Q}}_t\mathbf{G}_t^{\mathsf{T}}$.

a recursive fashion. Figure 8.2(b) illustrates filtering for the linear Gaussian SSM applied to the noisy tracking data in Figure 8.2(a) (shown by the green dots). The filtered estimates are computed using the Kalman filter algorithm described in Section 8.2.2. The red line shows the posterior mean estimate of the location, and the black circles show the posterior covariance. We see that the estimated trajectory is less noisy than the raw data, since it incorporates prior knowledge about how the data was generated.

Another task of interest is the smoothing problem where we want to compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})$ using an offline dataset. Figure 8.2(c) illustrates smoothing for the LG-SSM, implemented using the Kalman smoothing algorithm described in Section 8.2.3. We see that the resulting estimate is smoother, and that the posterior uncertainty is reduced (as visualized by the smaller confidence ellipses).

8.2.1.2 Online Bayesian linear regression (recursive least squares)

In Section 29.7.2 we discuss how to use the Kalman filter to recursively compute the exact posterior $p(\boldsymbol{w}|\mathcal{D}_{1:t})$ for a linear regression model in an online fashion. This is known as the recursive least squares algorithm. The basic idea is to treat the latent state to be the parameter values, $\boldsymbol{z}_t = \boldsymbol{w}$, and to define the non-stationary observation model as $p(\boldsymbol{y}_t|\boldsymbol{z}_t) = \mathcal{N}(y_t|\boldsymbol{x}_t^\mathsf{T}\boldsymbol{z}_t, \sigma^2)$, and the dynamics model as $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, 0\mathbf{I})$.

8.2.1.3 Time series forecasting

In Section 29.12, we discuss how to use Kalman filtering to perform time series forecasting.

8.2.2 The Kalman filter

The **Kalman filter** (**KF**) is an algorithm for exact Bayesian filtering for linear Gaussian state space models. The resulting algorithm is the Gaussian analog of the HMM filter in Section 9.2.2. The belief state at time t is now given by $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$, where we use the notation $\boldsymbol{\mu}_{t|t'}$ and $\boldsymbol{\Sigma}_{t|t'}$ to represent the posterior mean and covariance given $\boldsymbol{y}_{1:t'}$.[5] Since everything is Gaussian, we can perform the prediction and update steps in closed form, as we explain below (see Section 8.2.2.4 for the derivation).

8.2.2.1 Predict step

The one-step-ahead prediction for the hidden state, also called the **time update step**, is given by the following:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.22}$$

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{F}_t\boldsymbol{\mu}_{t-1|t-1} + \mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t \tag{8.23}$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_t \tag{8.24}$$

5. We represent the mean and covariance of the filtered belief state by $\boldsymbol{\mu}_{t|t}$ and $\boldsymbol{\Sigma}_{t|t}$, but some authors use the notation $\boldsymbol{m}_t$ and $\mathbf{P}_t$ instead. We represent the mean and covariance of the smoothed belief state by $\boldsymbol{\mu}_{t|T}$ and $\boldsymbol{\Sigma}_{t|T}$, but some authors use the notation $\boldsymbol{m}_t^s$ and $\mathbf{P}_t^s$ instead. Finally, we represent the mean and covariance of the one-step-ahead posterior predictive distribution, $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$, by $\boldsymbol{\mu}_{t|t-1}$ and $\boldsymbol{\Sigma}_{t|t-1}$, whereas some authors use $\boldsymbol{m}_t^-$ and $\mathbf{P}_t^-$ instead.

8.2.2.2 Update step

The update step (also called the **measurement update step**) can be computed using Bayes' rule, as follows:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) \tag{8.25}$$

$$\hat{\boldsymbol{y}}_t = \mathbf{H}_t\boldsymbol{\mu}_{t|t-1} + \mathbf{D}_t\boldsymbol{u}_t + \boldsymbol{d}_t \tag{8.26}$$

$$\mathbf{S}_t = \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t \tag{8.27}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}\mathbf{S}_t^{-1} \tag{8.28}$$

$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \tag{8.29}$$

$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} \tag{8.30}$$

$$= \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^\mathsf{T} \tag{8.31}$$

where $\mathbf{K}_t$ is the **Kalman gain matrix**. Note that $\hat{\boldsymbol{y}}_t$ is the expected observation, so $\boldsymbol{e}_t = \boldsymbol{y}_t - \hat{\boldsymbol{y}}_t$ is the **residual error**, also called the **innovation term**. The covariance of the observation is denoted by $\mathbf{S}_t$, and the cross covariance bwteen the observation and state is denoted by $\mathbf{C}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}$. In practice, to compute the Kalman gain, we do not use $\mathbf{K}_t = \mathbf{C}_t\mathbf{S}_t^{-1}$, but instead we solve the linear system $\mathbf{K}_t\mathbf{S}_t = \mathbf{C}_t$.[6]

To understand the update step intuitively, note that the update for the latent mean, $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t\boldsymbol{e}_t$, is the predicted new latent mean plus a correction factor, which is $\mathbf{K}_t$ times the error signal $\boldsymbol{e}_t$. If $\mathbf{H}_t = \mathbf{I}$, then $\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{S}_t^{-1}$; in the scalar case, this becomes $k_t = \Sigma_{t|t-1}/S_t$, which is the ratio between the variance of the prior (from the dynamics model) and the variance of the measurement, which we can interpret as an inverse signal to noise ratio. If we have a strong prior and/or very noisy sensors, $|\mathbf{K}_t|$ will be small, and we will place little weight on the correction term. Conversely, if we have a weak prior and/or high precision sensors, then $|\mathbf{K}_t|$ will be large, and we will place a lot of weight on the correction term. Similarly, the new covariance is the old covariance minus a positive definite matrix, which depends on how informative the measurement is.

Note that, by using the matrix inversion lemma, the Kalman gain matrix can also be written as

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t)^{-1} = (\boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}\mathbf{H}_t)^{-1}\mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1} \tag{8.32}$$

This is useful if $\mathbf{R}_t^{-1}$ is precomputed (e.g., if it is constant over time) and $N_y \gg N_z$. In addition, in Equation (8.97), we give the information form of the filter, which shows that the posterior precision has the form $\boldsymbol{\Sigma}_t^{-1} = \boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}\mathbf{H}_t$, so we can also write the gain matrix as $\mathbf{K}_t = \boldsymbol{\Sigma}_t\mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}$.

8.2.2.3 Posterior predictive

The one-step-ahead posterior predictive density for the observations can be computed as follows. (We ignore inputs and bias terms, for notational brevity.) First we compute the one-step-ahead predictive density for latent states:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}_{t-1} \tag{8.33}$$

$$= \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{\mu}_{t-1|t-1}, \mathbf{F}_t\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_t) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.34}$$

6. Equivalently we have $\mathbf{S}_t^\mathsf{T}\mathbf{K}_t^\mathsf{T} = \mathbf{C}_t^\mathsf{T}$, so we can compute $\mathbf{K}_t$ in JAX using `K = jnp.linalg.lstq(S.T, C.T)[0].T`.

Then we convert this to a prediction about observations by marginalizing out $\boldsymbol{z}_t$:

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{y}_t, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}_t = \int p(\boldsymbol{y}_t|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}_t = \mathcal{N}(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_t, \mathbf{S}_t) \tag{8.35}$$

This can also be used to compute the log-likelihood of the observations: The normalization constant of the new posterior can be computed as follows:

$$\log p(\boldsymbol{y}_{1:T}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \sum_{t=1}^{T} \log Z_t \tag{8.36}$$

where we define $p(\boldsymbol{y}_1|\boldsymbol{y}_0) = p(\boldsymbol{y}_1)$. This is just a sum of the log probabilities of the one-step-ahead measurement predictions, and is a measure of how "surprised" the model is at each step.

We can generalize the prediction step to predict observations K steps into the future by first forecasting K steps in latent space, and then "grounding" the final state into predicted observations. (This is in contrast to an RNN (Section 16.3.4), which requires generating observations at each step, in order to update future hidden states.)

8.2.2.4 Derivation

In this section we derive the Kalman filter equations, following [SS23, Sec 6.3]. The results are a straightforward application of the rules for manipulating linear Gaussian systems, discussed in Section 2.3.2.

First we derive the prediction step. From Equation (2.120), the joint predictive distribution for states is given by

$$\begin{aligned} p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) &= p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1}) && (8.37)\\ &= \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{z}_{t-1}, \mathbf{Q}_t)\mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}) && (8.38)\\ &= \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}_{t-1}\\ \boldsymbol{z}_t\end{pmatrix}|\boldsymbol{\mu}', \boldsymbol{\Sigma}'\right) && (8.39)\end{aligned}$$

where

$$\boldsymbol{\mu}' = \begin{pmatrix}\boldsymbol{\mu}_{t-1|t-1}\\ \mathbf{F}_t\boldsymbol{\mu}_{t-1|t-1}\end{pmatrix}, \ \boldsymbol{\Sigma}' = \begin{pmatrix}\boldsymbol{\Sigma}_{t-1|t-1} & \boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}_t^\mathsf{T}\\ \mathbf{F}_t\boldsymbol{\Sigma}_{t-1|t-1} & \mathbf{F}_t\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_t\end{pmatrix} \tag{8.40}$$

Hence the marginal predictive distribution for states is given by

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{\mu}_{t-1|t-1}, \mathbf{F}_t\boldsymbol{\Sigma}_{t-1|t-1}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_t) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.41}$$

Now we derive the measurement update step. The joint distribution for state and observation is given by

$$\begin{aligned} p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) &= p(\boldsymbol{y}_t|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) && (8.42)\\ &= \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_t\boldsymbol{z}_t, \mathbf{R}_t)\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) && (8.43)\\ &= \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}_t\\ \boldsymbol{y}_t\end{pmatrix}|\boldsymbol{\mu}'', \boldsymbol{\Sigma}''\right) && (8.44)\end{aligned}$$

where

$$\boldsymbol{\mu}'' = \begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \mathbf{H}_t \boldsymbol{\mu}_{t|t-1} \end{pmatrix}, \; \boldsymbol{\Sigma}'' = \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} \\ \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} & \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t \end{pmatrix} \tag{8.45}$$

Finally, we convert this joint into a conditional using Equation (2.78) as follows:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_t, \boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) \tag{8.46}$$

$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t)^{-1}[\boldsymbol{y}_t - \mathbf{H}_t\boldsymbol{\mu}_{t|t-1}] \tag{8.47}$$

$$= \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t[\boldsymbol{y}_t - \mathbf{H}_t\boldsymbol{\mu}_{t|t-1}] \tag{8.48}$$

$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}(\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t)^{-1}\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} \tag{8.49}$$

$$= \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1} \tag{8.50}$$

where

$$\mathbf{S}_t = \mathbf{H}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T} + \mathbf{R}_t \tag{8.51}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}_t^\mathsf{T}\mathbf{S}_t^{-1} \tag{8.52}$$

8.2.2.5 Abstract formulation

We can represent the Kalman filter equations much more compactly by defining various functions that create and manipulate jointly Gaussian systems, as in Section 2.3.2. In particular, suppose we have the following linear Gaussian system:

$$p(\boldsymbol{z}) = \mathcal{N}(\breve{\boldsymbol{\mu}}, \breve{\boldsymbol{\Sigma}}) \tag{8.53}$$

$$p(\boldsymbol{y}|\boldsymbol{z}) = \mathcal{N}(\mathbf{A}\boldsymbol{z} + \boldsymbol{b}, \boldsymbol{\Omega}) \tag{8.54}$$

Then the joint is given by

$$p(\boldsymbol{z}, \boldsymbol{y}) = \mathcal{N}\left(\begin{pmatrix} \breve{\boldsymbol{\mu}} \\ \overline{\boldsymbol{\mu}} \end{pmatrix}, \begin{pmatrix} \breve{\boldsymbol{\Sigma}} & \mathbf{C} \\ \mathbf{C}^\mathsf{T} & \mathbf{S} \end{pmatrix}\right) = \mathcal{N}\left(\begin{pmatrix} \breve{\boldsymbol{\mu}} \\ \mathbf{A}\,\breve{\boldsymbol{\mu}} + \boldsymbol{b} \end{pmatrix}, \begin{pmatrix} \breve{\boldsymbol{\Sigma}} & \breve{\boldsymbol{\Sigma}}\,\mathbf{A}^\mathsf{T} \\ \mathbf{A}\,\breve{\boldsymbol{\Sigma}} & \mathbf{A}\,\breve{\boldsymbol{\Sigma}}\,\mathbf{A}^\mathsf{T} + \boldsymbol{\Omega} \end{pmatrix}\right) \tag{8.55}$$

and the posterior is given by

$$p(\boldsymbol{z}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{z}|\,\widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}}) = \mathcal{N}\left(\boldsymbol{z}|\,\breve{\boldsymbol{\mu}} + \mathbf{K}(\boldsymbol{y} - \overline{\boldsymbol{\mu}}), \breve{\boldsymbol{\Sigma}} - \mathbf{K}\mathbf{S}\mathbf{K}^\mathsf{T}\right) \tag{8.56}$$

where $\mathbf{K} = \mathbf{C}\mathbf{S}^{-1}$. See Algorithm 8.1 for the pseudocode.

We can now apply these functions to derive Kalman filtering as follows. In the prediction step, we compute

$$p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_{t-1|t-1} \\ \boldsymbol{\mu}_{t|t-1} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{t-1|t-1} & \boldsymbol{\Sigma}_{t-1,t|t-1} \\ \boldsymbol{\Sigma}_{t,t-1|t-1} & \boldsymbol{\Sigma}_{t|t-1} \end{pmatrix}\right) \tag{8.57}$$

$$(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \boldsymbol{\Sigma}_{t-1,t|t}) = \mathbf{GaussMoments}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \mathbf{F}_t, \mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \mathbf{Q}_t) \tag{8.58}$$

Algorithm 8.1: Functions for a linear Gaussian system.

def **GaussMoments**($\breve{\boldsymbol{\mu}}, \breve{\boldsymbol{\Sigma}}, \mathbf{A}, \boldsymbol{b}, \boldsymbol{\Omega}$) :
$\overline{\boldsymbol{\mu}} = \mathbf{A}\ \breve{\boldsymbol{\mu}} + \boldsymbol{b}$
$\mathbf{S} = \boldsymbol{\Omega} + \mathbf{A}\ \breve{\boldsymbol{\Sigma}}\ \mathbf{A}^{\mathsf{T}}$
$\mathbf{C} = \breve{\boldsymbol{\Sigma}}\ \mathbf{A}^{\mathsf{T}}$
Return $(\overline{\boldsymbol{\mu}}, \mathbf{S}, \mathbf{C})$

def **GaussCondition**($\breve{\boldsymbol{\mu}}, \breve{\boldsymbol{\Sigma}}, \overline{\boldsymbol{\mu}}, \mathbf{S}, \mathbf{C}, \boldsymbol{y}$) :
$\mathbf{K} = \mathbf{C}\mathbf{S}^{-1}$
$\widehat{\boldsymbol{\mu}} = \breve{\boldsymbol{\mu}} + \mathbf{K}(\boldsymbol{y} - \overline{\boldsymbol{\mu}})$
$\widehat{\boldsymbol{\Sigma}} = \breve{\boldsymbol{\Sigma}} - \mathbf{K}\mathbf{S}\mathbf{K}^{\mathsf{T}}$
$\ell = \log \mathcal{N}(\boldsymbol{y} | \overline{\boldsymbol{\mu}}, \mathbf{S})$
Return $(\widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}}, \ell)$

from which we get the marginal distribution

$$p(\boldsymbol{z}_t | \boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}) \tag{8.59}$$

In the update step, we compute the joint distribution

$$p(\boldsymbol{z}_t, \boldsymbol{y}_t | \boldsymbol{y}_{1:t-1}) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_{t|t-1} \\ \overline{\boldsymbol{\mu}}_t \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{t|t-1} & \mathbf{C}_t \\ \mathbf{C}_t^{\mathsf{T}} & \mathbf{S}_t \end{pmatrix} \right) \tag{8.60}$$

$$(\hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t) = \mathbf{GaussMoments}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \mathbf{H}_t, \mathbf{D}_t \boldsymbol{u}_t + \boldsymbol{d}_t, \mathbf{R}_t) \tag{8.61}$$

We then condition this on the observations to get the posterior distribution

$$p(\boldsymbol{z}_t | \boldsymbol{y}_t, \boldsymbol{y}_{1:t-1}) = p(\boldsymbol{z}_t | \boldsymbol{y}_{1:t}) = \mathcal{N}(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) \tag{8.62}$$

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \mathbf{GaussCondition}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t, \boldsymbol{y}_t) \tag{8.63}$$

The overall KF algorithm is shown in Algorithm 8.2.

Algorithm 8.2: Kalman filter.

def KF($\mathbf{F}_{1:T}, \mathbf{B}_{1:T}, \boldsymbol{b}_{1:T}, \mathbf{Q}_{1:T}, \mathbf{H}_{1:T}, \mathbf{D}_{1:T}, \boldsymbol{d}_{1:T}, \mathbf{R}_{1:T}, \boldsymbol{u}_{1:T}, \boldsymbol{y}_{1:T}, \boldsymbol{\mu}_{0|0}, \boldsymbol{\Sigma}_{0|0}$) :
foreach $t = 1 : T$ **do**
  // Predict:
  $(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \mathbf{GaussMoments}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \mathbf{F}_t, \mathbf{B}_t \boldsymbol{u}_t + \boldsymbol{b}_t, \mathbf{Q}_t)$
  // Update:
  $(\overline{\boldsymbol{\mu}}, \mathbf{S}, \mathbf{C}) = \mathbf{GaussMoments}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \mathbf{H}_t, \mathbf{D}_t \boldsymbol{u}_t + \boldsymbol{d}_t, \mathbf{R}_t)$
  $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \mathbf{GaussCondition}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \overline{\boldsymbol{\mu}}, \mathbf{S}, \mathbf{C}, \boldsymbol{y})$
Return $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})_{t=1}^{T}, \sum_{t=1}^{T} \ell_t$

8.2.2.6 Numerical issues

In practice, the Kalman filter can encounter numerical issues. One solution is to use the **information filter**, which recursively updates the natural parameters of the Gaussian, $\boldsymbol{\Lambda}_{t|t} = \boldsymbol{\Sigma}_{t|t}^{-1}$ and $\boldsymbol{\eta}_{t|t} = \boldsymbol{\Lambda}_t \boldsymbol{\mu}_{t|t}$, instead of the mean and covariance (see Section 8.2.4). Another solution is the **square root filter**, which works with the Cholesky or QR decomposition of $\boldsymbol{\Sigma}_{t|t}$, which is much more numerically stable than directly updating $\boldsymbol{\Sigma}_{t|t}$. These techniques can be combined to create the **square root information filter** (SRIF) [May79]. (According to [Bie06], the SRIF was developed in 1969 for use in JPL's Mariner 10 mission to Venus.) In [Tol22] they present an approach which uses QR decompositions instead of matrix inversions, which can also be more stable.

8.2.2.7 Continuous-time version

The Kalman filter can be extended to work with continuous time dynamical systems; the resulting method is called the **Kalman Bucy filter**. See [SS19, p208] for details. q

8.2.3 The Kalman (RTS) smoother

In Section 8.2.2, we described the Kalman filter, which sequentially computes $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ for each t. This is useful for online inference problems, such as tracking. However, in an offline setting, we can wait until all the data has arrived, and then compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})$. By conditioning on past and future data, our uncertainty will be significantly reduced. This is illustrated in Figure 8.2(c), where we see that the posterior covariance ellipsoids are smaller for the smoothed trajectory than for the filtered trajectory.

We now explain how to compute the smoothed estimates, using an algorithm called the **RTS smoother** or **RTSS**, named after its inventors, Rauch, Tung, and Striebel [RTS65]. It is also known as the **Kalman smoothing** algorithm. The algorithm is the linear-Gaussian analog to the forwards-filtering backwards-smoothing algorithm for HMMs in Section 9.2.4.

8.2.3.1 Algorithm

In this section, we state the Kalman smoother algorithm. We give the derivation in Section 8.2.3.2.

The key update equations are as follows: From this, we can extract the smoothed marginal

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) \tag{8.64}$$

$$\boldsymbol{\mu}_{t+1|t} = \mathbf{F}_t \boldsymbol{\mu}_{t|t} \tag{8.65}$$

$$\boldsymbol{\Sigma}_{t+1|t} = \mathbf{F}_t \boldsymbol{\Sigma}_{t|t} \mathbf{F}_t^{\mathsf{T}} + \mathbf{Q}_{t+1} \tag{8.66}$$

$$\mathbf{J}_t = \boldsymbol{\Sigma}_{t|t} \mathbf{F}_t^{\mathsf{T}} \boldsymbol{\Sigma}_{t+1|t}^{-1} \tag{8.67}$$

$$\boldsymbol{\mu}_{t|T} = \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) \tag{8.68}$$

$$\boldsymbol{\Sigma}_{t|T} = \boldsymbol{\Sigma}_{t|t} + \mathbf{J}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{J}_t^{\mathsf{T}} \tag{8.69}$$

8.2.3.2 Derivation

In this section, we derive the RTS smoother, following [SS23, Sec 12.2]. As in the derivation of the Kalman filter in Section 8.2.2.4, we make heavy use of the rules for manipulating linear Gaussian

systems, discussed in Section 2.3.2.

The joint filtered distribution for two consecutive time slices is

$$p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \mathcal{N}(\boldsymbol{z}_{t+1}|\mathbf{F}_t\boldsymbol{z}_t, \mathbf{Q}_{t+1})\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}) \tag{8.70}$$

$$= \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}_t \\ \boldsymbol{z}_{t+1}\end{pmatrix}|\boldsymbol{m}_1, \mathbf{V}_1\right) \tag{8.71}$$

where

$$\boldsymbol{m}_1 = \begin{pmatrix}\boldsymbol{\mu}_{t|t} \\ \mathbf{F}_t\boldsymbol{\mu}_{t|t}\end{pmatrix}, \ \mathbf{V}_1 = \begin{pmatrix}\boldsymbol{\Sigma}_{t|t} & \boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T} \\ \mathbf{F}_t\boldsymbol{\Sigma}_{t|t} & \mathbf{F}_t\boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_{t+1}\end{pmatrix} \tag{8.72}$$

By the Markov property for the hidden states we have

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:t}, \boldsymbol{y}_{t+1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:t}) \tag{8.73}$$

and hence by conditioning the joint distribution $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})$ on the future state we get

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{m}_2, \mathbf{V}_2) \tag{8.74}$$

$$\boldsymbol{\mu}_{t+1|t} = \mathbf{F}_t\boldsymbol{\mu}_{t|t} \tag{8.75}$$

$$\boldsymbol{\Sigma}_{t+1|t} = \mathbf{F}_t\boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T} + \mathbf{Q}_{t+1} \tag{8.76}$$

$$\mathbf{J}_t = \boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T}\boldsymbol{\Sigma}_{t+1|t}^{-1} \tag{8.77}$$

$$\boldsymbol{m}_2 = \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{z}_{t+1} - \boldsymbol{\mu}_{t+1|t}) \tag{8.78}$$

$$\mathbf{V}_2 = \boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^\mathsf{T} \tag{8.79}$$

where $\mathbf{J}_t$ is the backwards Kalman gain matrix.

$$\mathbf{J}_t = \boldsymbol{\Sigma}_{t,t+1|t}\boldsymbol{\Sigma}_{t+1|t}^{-1} \tag{8.80}$$

where $\boldsymbol{\Sigma}_{t,t+1|t} = \boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T}$ is the cross covariance term in the upper right block of $\mathbf{V}_1$.

The joint distribution of two consecutive time slices given all the data is

$$p(\boldsymbol{z}_{t+1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T})p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) \tag{8.81}$$

$$= \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{m}_2(\boldsymbol{z}_{t+1}), \mathbf{V}_2)\mathcal{N}(\boldsymbol{z}_{t+1}|\boldsymbol{\mu}_{t+1|T}, \boldsymbol{\Sigma}_{t+1|T}) \tag{8.82}$$

$$= \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}_{t+1} \\ \boldsymbol{z}_t\end{pmatrix}|\boldsymbol{m}_3, \mathbf{V}_3\right) \tag{8.83}$$

where

$$\boldsymbol{m}_3 = \begin{pmatrix}\boldsymbol{\mu}_{t+1|T} \\ \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t})\end{pmatrix}, \ \mathbf{V}_3 = \begin{pmatrix}\boldsymbol{\Sigma}_{t+1|T} & \boldsymbol{\Sigma}_{t+1|T}\mathbf{J}_t^\mathsf{T} \\ \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|T} & \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|T}\mathbf{J}_t^\mathsf{T} + \mathbf{V}_2\end{pmatrix} \tag{8.84}$$

From this, we can extract $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})$, with the mean and covariance given by Equation (8.68) and Equation (8.69).

8.2.3.3 Two-filter smoothing

Note that the backwards pass of the Kalman smoother does not need access to the observations, $\boldsymbol{y}_{1:T}$, but does need access to the filtered belief states from the forwards pass, $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$. There is an alternative version of the algorithm, known as **two-filter smoothing** [FP69; Kit04], in which we compute the forwards pass as usual, and then separately compute backwards messages $p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t) \propto \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}^b_{t|t}, \boldsymbol{\Sigma}^b_{t|t})$, similar to the backwards filtering algorithm in HMMs (Section 9.2.3).

However, these backwards messages are conditional likelihoods, not posteriors, which can cause numerical problems. For example, consider $t = T$; in this case, we need to set the initial covariance matrix to be $\boldsymbol{\Sigma}^b_T = \infty\mathbf{I}$, so that the backwards message has no effect on the filtered posterior (since there is no evidence beyond step T). This problem can be resolved by working in information form. An alternative approach is to generalize the two-filter smoothing equations to ensure the likelihoods are normalizable by multiplying them by artificial distributions [BDM10].

In general, the RTS smoother is preferred to the two-filter smoother, since it is more numerically stable, and it is easier to generalize it to the nonlinear case.

8.2.3.4 Time and space complexity

In general, the Kalman smoothing algorithm takes $O(N_y^3 + N_z^2 + N_y N_z)$ per step, where there are T steps. This can be slow when applied to long sequences. In [SGF21], they describe how to reduce this to $O(\log T)$ steps using a **parallel prefix scan** operator that can be run efficiently on GPUs. In addition, we can reduce the space from $O(T)$, to $O(\log T)$ using the same algorithm as in Section 9.2.5.

8.2.3.5 Forwards filtering backwards sampling

To draw posterior samples from the LG-SSM, we can leverage the following result:

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) = \mathcal{N}(\boldsymbol{z}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t) \tag{8.85}$$

$$\tilde{\boldsymbol{\mu}}_t = \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{z}_{t+1} - \mathbf{F}_t\boldsymbol{\mu}_{t|t}) \tag{8.86}$$

$$\tilde{\boldsymbol{\Sigma}}_t = \boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^\mathsf{T} = \boldsymbol{\Sigma}_{t|t} - \boldsymbol{\Sigma}_{t|t}\mathbf{F}_t^\mathsf{T}\boldsymbol{\Sigma}^{-1}_{t+1|t}\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^\mathsf{T} \tag{8.87}$$

$$= \boldsymbol{\Sigma}_{t|t}(\mathbf{I} - \mathbf{F}_t^\mathsf{T}\mathbf{J}_t^\mathsf{T}) \tag{8.88}$$

where $\mathbf{J}_t$ is the backwards Kalman gain defined in Equation (8.67).

8.2.4 Information form filtering and smoothing

This section is written by Giles Harper-Donnelly.

In this section, we derive the Kalman filter and smoother algorithms in information form. We will see that this is the "dual" of Kalman filtering/smoothing in moment form. In particular, while computing marginals in moment form is easy, computing conditionals is hard (requires a matrix inverse). Conversely, for information form, computing marginals is hard, but computing conditionals is easy.

8.2.4.1 Filtering: algorithm

The predict step has a similar structure to the update step in moment form. We start with the prior $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t-1}) = \mathcal{N}_c(\boldsymbol{z}_{t-1}|\boldsymbol{\eta}_{t-1|t-1}, \boldsymbol{\Lambda}_{t-1|t-1})$ and then compute

$$
\begin{aligned}
p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) &= \mathcal{N}_c(\boldsymbol{z}_t|\boldsymbol{\eta}_{t|t-1}, \boldsymbol{\Lambda}_{t|t-1}) && (8.89)\\
\mathbf{M}_t &= \boldsymbol{\Lambda}_{t-1|t-1} + \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{F}_t && (8.90)\\
\mathbf{J}_t &= \mathbf{Q}_t^{-1}\mathbf{F}_t\mathbf{M}_t^{-1} && (8.91)\\
\boldsymbol{\Lambda}_{t|t-1} &= \mathbf{Q}_t^{-1} - \mathbf{Q}_t^{-1}\mathbf{F}_t(\boldsymbol{\Lambda}_{t-1|t-1} + \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{F}_t)^{-1}\ \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1} && (8.92)\\
&= \mathbf{Q}_t^{-1} - \mathbf{J}_t\mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1} && (8.93)\\
&= \mathbf{Q}_t^{-1} - \mathbf{J}_t\mathbf{M}_t\mathbf{J}_t^\mathsf{T} && (8.94)\\
\boldsymbol{\eta}_{t|t-1} &= \mathbf{J}_t\boldsymbol{\eta}_{t-1|t-1} + \boldsymbol{\Lambda}_{t|t-1}(\mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t), && (8.95)
\end{aligned}
$$

where $\mathbf{J}_t$ is analogous to the Kalman gain matrix in moment form Equation (8.28). From the matrix inversion lemma, Equation (2.93), we see that Equation (8.92) is the inverse of the predicted covariance $\boldsymbol{\Sigma}_{t|t-1}$ given in Equation (8.24).

The update step in information form is as follows:

$$
\begin{aligned}
p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}) &= \mathcal{N}_c(\boldsymbol{z}_t|\boldsymbol{\eta}_{t|t}, \boldsymbol{\Lambda}_{t|t}) && (8.96)\\
\boldsymbol{\Lambda}_{t|t} &= \boldsymbol{\Lambda}_{t|t-1} + \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}\mathbf{H}_t && (8.97)\\
\boldsymbol{\eta}_{t|t} &= \boldsymbol{\eta}_{t|t-1} + \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}(\boldsymbol{y}_t - \mathbf{D}_t\boldsymbol{u}_t - \boldsymbol{d}_t). && (8.98)
\end{aligned}
$$

8.2.4.2 Filtering: derivation

For the predict step, we first derive the joint distribution over hidden states at $t, t-1$:

$$
\begin{aligned}
p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) &= p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{u}_t)p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t-1}) && (8.99)\\
&= \mathcal{N}_c(\boldsymbol{z}_t, |\mathbf{Q}_t^{-1}(\mathbf{F}_t\boldsymbol{z}_{t-1} + \mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t), \mathbf{Q}_t^{-1}) && (8.100)\\
&\times \mathcal{N}_c(\boldsymbol{z}_{t-1}, |\boldsymbol{\eta}_{t-1|t-1}, \boldsymbol{\Lambda}_{t-1|t-1}) && (8.101)\\
&= \mathcal{N}_c(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{\eta}_{t-1,t|t}, \boldsymbol{\Lambda}_{t-1,t|t}) && (8.102)
\end{aligned}
$$

where

$$
\boldsymbol{\eta}_{t-1,t|t-1} = \begin{pmatrix} \boldsymbol{\eta}_{t-1|t-1} - \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}(\mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t) \\ \mathbf{Q}_t^{-1}(\mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t) \end{pmatrix} \tag{8.103}
$$

$$
\boldsymbol{\Lambda}_{t-1,t|t-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{t-1|t-1} + \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{F}_t & -\mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1} \\ -\mathbf{Q}_t^{-1}\mathbf{F}_t & \mathbf{Q}_t^{-1} \end{pmatrix} \tag{8.104}
$$

The information form predicted parameters $\boldsymbol{\eta}_{t|t-1}, \boldsymbol{\Lambda}_{t|t-1}$ can then be derived using the marginalisation formulae in Section 2.3.1.4.

For the update step, we start with the joint distribution over the hidden state and the observation

at t:

$$p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = p(\boldsymbol{y}_t|\boldsymbol{z}_t, \boldsymbol{u}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t-1}) \tag{8.105}$$

$$= \mathcal{N}_c(\boldsymbol{y}_t, |\mathbf{R}_t^{-1}(\mathbf{H}_t\boldsymbol{z}_t + \mathbf{D}\boldsymbol{u}_t + \boldsymbol{d}_t), \mathbf{R}_t^{-1})\mathcal{N}_c(\boldsymbol{z}_t|\boldsymbol{\eta}_{t|t-1}, \boldsymbol{\Lambda}_{t|t-1}) \tag{8.106}$$

$$= \mathcal{N}_c(\boldsymbol{z}_t, \boldsymbol{y}|\boldsymbol{\eta}_{\mathrm{z,y}|t}, \boldsymbol{\Lambda}_{\mathrm{z,y}|t}) \tag{8.107}$$

where

$$\boldsymbol{\eta}_{\mathrm{z,y}|t} = \begin{pmatrix} \boldsymbol{\eta}_{t|t-1} - \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}(\mathbf{D}_t\boldsymbol{u}_t + \boldsymbol{d}_t) \\ \mathbf{R}_t^{-1}(\mathbf{D}_t\boldsymbol{u}_t + \boldsymbol{d}_t) \end{pmatrix} \tag{8.108}$$

$$\boldsymbol{\Lambda}_{\mathrm{z,y}|t} = \begin{pmatrix} \boldsymbol{\Lambda}_{t|t-1} + \mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1}\mathbf{H}_t & -\mathbf{H}_t^\mathsf{T}\mathbf{R}_t^{-1} \\ -\mathbf{R}_t^{-1}\mathbf{H}_t & \mathbf{R}_t^{-1} \end{pmatrix} \tag{8.109}$$

The information form filtered parameters $\boldsymbol{\eta}_{t|t}, \boldsymbol{\Lambda}_{t|t}$ are then derived using the conditional formulae in 2.3.1.4.

8.2.4.3 Smoothing: algorithm

The smoothing equations are as follows:

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = \mathcal{N}_c(\boldsymbol{z}_t|\boldsymbol{\eta}_{t|T}, \boldsymbol{\Lambda}_{t|T}) \tag{8.110}$$

$$\mathbf{U}_t = \mathbf{Q}_t^{-1} + \boldsymbol{\Lambda}_{t+1|T} - \boldsymbol{\Lambda}_{t+1|t} \tag{8.111}$$

$$\mathbf{L}_t = \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{U}_t^{-1} \tag{8.112}$$

$$\boldsymbol{\Lambda}_{t|T} = \boldsymbol{\Lambda}_{t|t} + \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{F}_t - \mathbf{L}_t\mathbf{Q}_t^{-1}\mathbf{F} \tag{8.113}$$

$$= \boldsymbol{\Lambda}_{t|t} + \mathbf{F}_t^\mathsf{T}\mathbf{Q}_t^{-1}\mathbf{F}_t - \mathbf{L}_t\mathbf{U}_t\mathbf{L}_t^\mathsf{T} \tag{8.114}$$

$$\boldsymbol{\eta}_{t|T} = \boldsymbol{\eta}_{t|t} + \mathbf{L}_t(\boldsymbol{\eta}_{t+1|T} - \boldsymbol{\eta}_{t+1|t}). \tag{8.115}$$

The parameters $\boldsymbol{\eta}_{t|t}$ and $\boldsymbol{\Lambda}_{t|t}$ are the filtered values from Equations (8.98) and (8.97) respectively. Similarly, $\boldsymbol{\eta}_{t+1|t}$ and $\boldsymbol{\Lambda}_{t+1|t}$ are the predicted parameters from Equations (8.95) and (8.92). The matrix $\mathbf{L}_t$ is the information form analog to the backwards Kalman gain matrix in Equation (8.67).

8.2.4.4 Smoothing: derivation

From the generic forwards-filtering backwards-smoothing equation, Equation (8.14), we have

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) \int \left[\frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} \right] d\boldsymbol{z}_{t+1} \tag{8.116}$$

$$= \int p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}) \frac{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})} d\boldsymbol{z}_{t+1} \tag{8.117}$$

$$= \int \mathcal{N}_c(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{\eta}_{t,t+1|t}, \boldsymbol{\Lambda}_{t,t+1|t}) \frac{\mathcal{N}_c(\boldsymbol{z}_{t+1}|\boldsymbol{\eta}_{t+1|T}, \boldsymbol{\Lambda}_{t+1|T})}{\mathcal{N}_c(\boldsymbol{z}_{t+1}|\boldsymbol{\eta}_{t+1|t}, \boldsymbol{\Lambda}_{t+1|t})} d\boldsymbol{z}_{t+1} \tag{8.118}$$

$$= \int \mathcal{N}_c(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{\eta}_{t,t+1|T}, \boldsymbol{\Lambda}_{t,t+1|T}) d\boldsymbol{z}_{t+1}. \tag{8.119}$$

The parameters of the joint filtering predictive distribution, $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t})$, take precisely the same form as those in the filtering derivation described in Section 8.2.4.2:

$$\boldsymbol{\eta}_{t,t+1|t} = \begin{pmatrix} \boldsymbol{\eta}_{t|t} \\ \mathbf{0} \end{pmatrix}, \quad \boldsymbol{\Lambda}_{t,t+1|t} = \begin{pmatrix} \boldsymbol{\Lambda}_{t|t} + \mathbf{F}_{t+1}^{\mathsf{T}} \mathbf{Q}_{t+1}^{-1} \mathbf{F}_{t+1} & -\mathbf{F}_{t+1}^{\mathsf{T}} \mathbf{Q}_{t+1}^{-1} \\ -\mathbf{Q}_{t+1}^{-1} \mathbf{F}_{t+1} & \mathbf{Q}_{t+1}^{-1} \end{pmatrix}, \tag{8.120}$$

We can now update this potential function by subtracting out the filtered information and adding in the smoothing information, using the rules for manipulating Gaussian potentials described in Section 2.3.3:

$$\boldsymbol{\eta}_{t,t+1|T} = \boldsymbol{\eta}_{t,t+1|t} + \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\eta}_{t+1|T} \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\eta}_{t+1|t} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\eta}_{t|t} \\ \boldsymbol{\eta}_{t+1|T} - \boldsymbol{\eta}_{t+1|t} \end{pmatrix}, \tag{8.121}$$

and

$$\boldsymbol{\Lambda}_{t,t+1|T} = \boldsymbol{\Lambda}_{t,t+1|t} + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Lambda}_{t+1|T} \end{pmatrix} - \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Lambda}_{t+1|t} \end{pmatrix} \tag{8.122}$$

$$= \begin{pmatrix} \boldsymbol{\Lambda}_{t|t} + \mathbf{F}_{t+1}^{\mathsf{T}} \mathbf{Q}_{t+1}^{-1} \mathbf{F}_{t+1} & -\mathbf{F}_{t+1}^{\mathsf{T}} \mathbf{Q}_{t+1}^{-1} \\ -\mathbf{Q}_{t+1}^{-1} \mathbf{F}_{t+1} & \mathbf{Q}_{t+1}^{-1} + \boldsymbol{\Lambda}_{t+1|T} - \boldsymbol{\Lambda}_{t+1|t} \end{pmatrix} \tag{8.123}$$

Applying the information form marginalization formula Equation (2.85) leads to Equation (8.115) and Equation (8.113).

8.3 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. (We continue to assume that the noise is additive Gaussian, as in Equation (8.15).) The basic idea is to linearize the dynamics and observation models about the previous state estimate using a first order Taylor series expansion, and then to apply the standard Kalman filter equations from Section 8.2.2. Intuitively we can think of this as approximating a stationary non-linear dynamical system with a non-stationary linear dynamical system. This approach is called the **extended Kalman filter** or **EKF**.

8.3.1 Taylor series expansion

Suppose $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$, where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$ is a differentiable and invertible function. The pdf for $\boldsymbol{y}$ is given by

$$p(\boldsymbol{y}) = |\det \mathrm{Jac}(\boldsymbol{g}^{-1})(\boldsymbol{y})| \, \mathcal{N}(\boldsymbol{g}^{-1}(\boldsymbol{y})|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{8.124}$$

In general this is intractable to compute, so we seek an approximation.

Suppose $\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\delta}$, where $\boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. Then we can form a first order Taylor series expansion of the function $\boldsymbol{g}$ as follows:

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{\mu} + \boldsymbol{\delta}) \approx \boldsymbol{g}(\boldsymbol{\mu}) + \mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta} \tag{8.125}$$

where $\mathbf{G}(\boldsymbol{\mu}) = \mathrm{Jac}(\boldsymbol{g})(\boldsymbol{\mu})$ is the Jacobian of $\boldsymbol{g}$ at $\boldsymbol{\mu}$:

$$[\mathbf{G}(\boldsymbol{\mu})]_{jj'} = \frac{\partial g_j(\boldsymbol{x})}{\partial x_{j'}}|_{\boldsymbol{x}=\boldsymbol{\mu}} \tag{8.126}$$

We now derive the induced Gaussian approximation to $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$. The mean is given by

$$\mathbb{E}[\boldsymbol{y}] \approx \mathbb{E}[\boldsymbol{g}(\boldsymbol{\mu}) + \mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta}] = \boldsymbol{g}(\boldsymbol{\mu}) + \mathbf{G}(\boldsymbol{\mu})\mathbb{E}[\boldsymbol{\delta}] = \boldsymbol{g}(\boldsymbol{\mu}) \tag{8.127}$$

The covariance is given by

$$\begin{aligned}
\text{Cov}[\boldsymbol{y}] &= \mathbb{E}\left[(\boldsymbol{g}(\boldsymbol{x}) - \mathbb{E}[\boldsymbol{g}(\boldsymbol{x})])(\boldsymbol{g}(\boldsymbol{x}) - \mathbb{E}[\boldsymbol{g}(\boldsymbol{x})])^\mathsf{T}\right] && (8.128)\\
&\approx \mathbb{E}\left[(\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{\mu}))(\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{\mu}))^\mathsf{T}\right] && (8.129)\\
&\approx \mathbb{E}\left[(\boldsymbol{g}(\boldsymbol{\mu}) + \mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta} - \boldsymbol{g}(\boldsymbol{\mu}))(\boldsymbol{g}(\boldsymbol{\mu}) + \mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta} - \boldsymbol{g}(\boldsymbol{\mu}))^\mathsf{T}\right] && (8.130)\\
&= \mathbb{E}\left[(\mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta})(\mathbf{G}(\boldsymbol{\mu})\boldsymbol{\delta})^\mathsf{T}\right] && (8.131)\\
&= \mathbf{G}(\boldsymbol{\mu})\mathbb{E}\left[\boldsymbol{\delta}\boldsymbol{\delta}^\mathsf{T}\right]\mathbf{G}(\boldsymbol{\mu})^\mathsf{T} && (8.132)\\
&= \mathbf{G}(\boldsymbol{\mu})\,\boldsymbol{\Sigma}\,\mathbf{G}(\boldsymbol{\mu})^\mathsf{T} && (8.133)
\end{aligned}$$

Algorithm 8.3: Linearized approximation to a joint Gaussian distribution.

```
def LinearizedMoments(μ, Σ, g, Ω):
  ŷ = g(μ)
  G = Jac(g)(μ)
  S = GΣG^T + Ω
  C = ΣG^T
  Return (ŷ, S, C)
```

When deriving the EKF, we need to compute the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ where

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{q},\ \boldsymbol{q} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}) \tag{8.134}$$

where $\boldsymbol{q}$ is independent of $\boldsymbol{x}$. We can compute this by defining the augmented function $\tilde{\boldsymbol{g}}(\boldsymbol{x}) = [\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x})]$ and following the procedure above. The resulting linear approximation to the joint is

$$\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \hat{\boldsymbol{y}}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma} & \mathbf{C}\\ \mathbf{C}^\mathsf{T} & \mathbf{S}\end{pmatrix}\right) = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{g}(\boldsymbol{\mu})\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma} & \boldsymbol{\Sigma}\mathbf{G}^\mathsf{T}\\ \mathbf{G}\boldsymbol{\Sigma} & \mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^\mathsf{T} + \boldsymbol{\Omega}\end{pmatrix}\right) \tag{8.135}$$

where the parameters are computed using Algorithm 8.3. We can then condition this joint Gaussian on the observed value $\boldsymbol{y}$ to get the posterior.

It is also possible to derive an approximation for the case of non-additive Gaussian noise, where $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{q})$. See [SS23, Sec 7.1] for details.

8.3.2 The extended Kalman filter (EKF)

We now derive the extended Kalman filter for performing approximate inference in the model given by Equation (8.15). We first linearize the dynamics model around $\boldsymbol{\mu}_{t-1|t-1}$ to get an approximation to the one-step-ahead predictive distribution $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{u}_{1:t}) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$. We then linearize the observation model around $\boldsymbol{\mu}_{t|t-1}$, and then perform a Gaussian update. (In Section 8.3.2.2, we consider linearizing around a different point that gives better accuracy.)

We can write one step of the EKF algorithm using the notation from Section 8.2.2.5 as follows:

$$(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \textbf{LinearizedMoments}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}(\cdot, \boldsymbol{u}_t), \mathbf{Q}_t) \tag{8.136}$$

$$(\hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t) = \textbf{LinearizedMoments}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{R}_t) \tag{8.137}$$

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \textbf{GaussCondition}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t, \boldsymbol{y}_t) \tag{8.138}$$

Spelling out the details more explicitly, we can write the predict step as follows:

$$\boldsymbol{\mu}_{t|t-1} = \boldsymbol{f}(\boldsymbol{\mu}_{t-1}, \boldsymbol{u}_t) \tag{8.139}$$

$$\boldsymbol{\Sigma}_{t|t-1} = \mathbf{F}_t \boldsymbol{\Sigma}_{t-1} \mathbf{F}_t^{\mathsf{T}} + \mathbf{Q}_t \tag{8.140}$$

where $\mathbf{F}_t \triangleq \text{Jac}(f(\cdot, \boldsymbol{u}_t))(\boldsymbol{\mu}_{t|t-1})$ is the $N_z \times N_z$ Jacobian matrix of the dynamics model. The update step is as follows:

$$\hat{\boldsymbol{y}}_t = h(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{u}_t) \tag{8.141}$$

$$\mathbf{S}_t = \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^{\mathsf{T}} + \mathbf{R}_t \tag{8.142}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^{\mathsf{T}} \mathbf{S}_t^{-1} \tag{8.143}$$

$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \tag{8.144}$$

$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathsf{T}} \tag{8.145}$$

where $\mathbf{H}_t \triangleq \text{Jac}(h(\cdot, \boldsymbol{u}_t))(\boldsymbol{\mu}_{t|t-1})$ is the $N_y \times N_z$ Jacobian matrix of the observation model and $\mathbf{K}_t$ is the $N_z \times N_y$ Kalman gain matrix. See Supplementary Section 8.2.1 for the details of the derivation.

8.3.2.1 Accuracy

The EKF is widely used because it is simple and relatively efficient. However, there are two cases when the EKF works poorly [IX00; VDMW03]. 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. The other setting where the EKF works poorly is when the function is highly nonlinear near the current mean (see Figure 8.5a).

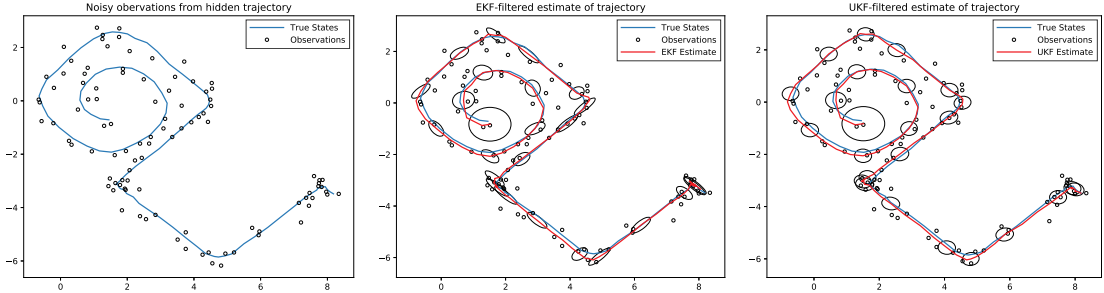
A more accurate approach is to use a second-order Taylor series approximation, known as the **second order EKF**. The resulting updates can still be computed in closed form (see [SS23, Sec 7.3] for details). We can further improve performance by repeatedly re-linearizing the equations around $\boldsymbol{\mu}_t$ instead of $\boldsymbol{\mu}_{t|t-1}$; this is called the iterated EKF (see Section 8.3.2.2). In Section 8.4.2, we will discuss an algorithm called the unscented Kalman filter (UKF) which is even more accurate, and is derivative free (does not require computing Jacobians).

8.3.2.2 Iterated EKF

Another way to improve the accuracy of the EKF is by repeatedly re-linearizing the measurement model around the current posterior, $\boldsymbol{\mu}_{t|t}$, instead of $\boldsymbol{\mu}_{t|t-1}$; this is called the **iterated EKF** [BC93]. See Algorithm 8.4 for the pseudocode. (If we set the number of iterations to $J = 1$, we recover the standard EKF.)

Algorithm 8.4: Iterated extended Kalman filter.

def **IEKF**$(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{h}, \mathbf{R}, \boldsymbol{y}_{1:T}, \boldsymbol{\mu}_{0|0}, \boldsymbol{\Sigma}_{0|0}, J)$:
  foreach $t = 1 : T$ **do**
    Predict step:
    $(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \textbf{LinearizedMoments}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}(\cdot, \boldsymbol{u}_t), \mathbf{Q}_t)$
    Update step:
    $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1}$, $\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1}$
    foreach $j = 1 : J$ **do**
      $(\hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t) = \textbf{LinearizedMoments}(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{R}_t)$
      $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \textbf{GaussCondition}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t, \boldsymbol{y}_t)$
Return $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})_{t=1}^T$



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

Figure 8.4: Illustration of filtering applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) Extended Kalman filter estimate. Generated by ekf_spiral.ipynb. (c) Unscented Kalman filter estimate. Generated by ukf_spiral.ipynb.

The IEKF can be interpreted as a Gauss–Newton method for finding MAP estimate of the state at each step [BC93]. Specifically it minimizes the following objective:

$$\mathcal{L}(\boldsymbol{z}_t) = \frac{1}{2}(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{z}_t))^\mathsf{T}\mathbf{R}_t^{-1}(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{z}_t)) + \frac{1}{2}(\boldsymbol{z}_t - \boldsymbol{\mu}_{t|t-1})^\mathsf{T}\boldsymbol{\Sigma}_{t|t-1}^{-1}(\boldsymbol{z}_t - \boldsymbol{\mu}_{t|t-1}) \tag{8.146}$$

See [SS23, Sec 7.4] for details.

Unfortunately the Gauss-Newton method can sometimes diverge. Various robust extensions — including Levenberg-Marquardt, line search, and quasi-Newton methods — have been proposed in [SHA15; SS20a]. See [SS23, Sec 7.5] for details.

8.3.2.3 Example: Tracking a point spiraling in 2d

In Section 8.2.1.1, we considered an example of state estimation and tracking of an object moving in 2d under a linear dynamics model with a linear observation model. However, motion and observation models are often nonlinear. For example, consider an object that is moving along a curved trajectory,

such as this:

$$\boldsymbol{f}(\boldsymbol{z}) = (z_1 + \Delta \sin(z_2), z_2 + \Delta \cos(z_1)) \tag{8.147}$$

where Δ is the discrete step size (see [SS19, p221] for the continuous time version). For simplicity, we assume full visibility of the state vector (modulo observation noise), so $\boldsymbol{h}(\boldsymbol{z}) = \boldsymbol{z}$.

Despite the simplicity of this model, exact inference is intractable. However, we can easily apply the EKF. The results are shown in Figure 8.4b.

8.3.2.4 Example: Neural network training

In Section 17.5.2, we show how to use the EKF to perform online parameter inference for an MLP regression model.

8.3.3 The extended Kalman smoother (EKS)

We can extend the EKF to the offline smoothing case, resulting in the **extended Kalman smoother**, also called the **extended RTS smoother**. We just need to linearize the dynamics around the filtered mean when computing $\mathbf{F}_t$, and then we can apply the standard Kalman smoother update. See [SS23, Sec 13.1] for more details.

For improved accuracy, we can use the **iterated EKS**, which relinearizes the model at the previous MAP estimate. In [Bel94], they show that IEKS is equivalent to a Gauss-Newton method for computing the MAP estimate of the smoothing posterior. Unfortunately the IEKS can diverge in some cases. A robust IEKS method, that uses line search and Levenberg-Marquardt to update the parameters, is presented in [SS20a].

8.4 Inference based on the unscented transform

In this section, we replace the local linearization of the model with a different approximation. The key idea is this: instead of computing a linear approximation to the dynamics and measurement functions, and then passing a Gaussian distribtution through the linearized functions, we instead approximate the joint distributions $p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$ and $p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$ by Gaussians, where the moments are computed using numerical integration; we can then compute the marginal and conditional of these distributions to perform the time and measurement updates.

There are many methods to compute the Gaussian moments, as we discuss in Section 8.5.1. Here we use a method based on the unscented transform (see Section 8.4.1). Using the unscented transform for the transition and observation models gives the the overall method, known as the **unscented Kalman filter** or **UKF**, [JU97; JUDW00], also called the **sigma point filter** [VDMW03].

The main advantage of the UKF over the EKF is that it can be more accurate, and more stable. (Indeed, [JU97; JUDW00] claim the term "unscented" was invented because the method "doesn't stink".) In addition, the UKF does not need to compute Jacobians of the observation and dynamics models, so it can be applied to non-differentiable models, or ones with hard constraints. However, the UKF can be slower, since it requires N_z evaluations of the dynamics and observation models. In addition, it has 3 hyper-parameters that need to be set.

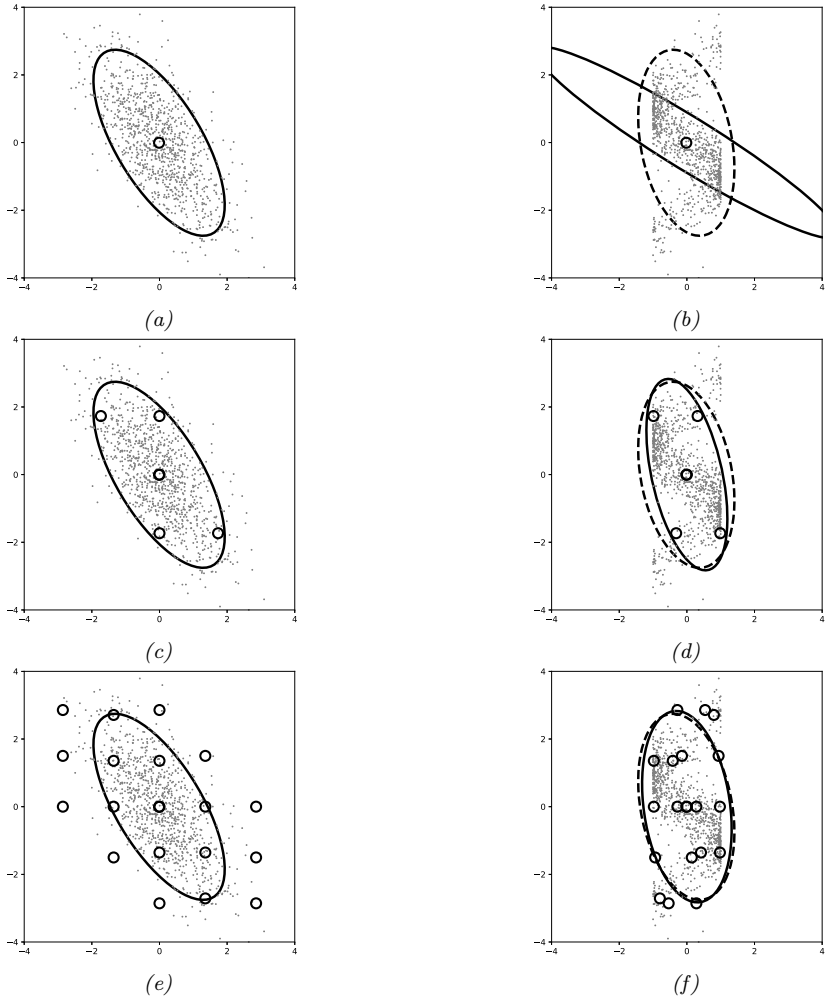


Figure 8.5: Illustration of different ways to approximate the distribution induced by a nonlinear transformation $f : \mathbb{R}^2 \to \mathbb{R}^2$. (a) Data from the source distribution, $\mathcal{D} = \{\boldsymbol{x}_i \sim p(\boldsymbol{x})\}$, with Gaussian approximation superimposed. (b) The dots show a Monte Carlo approximation to $p(f(\boldsymbol{x}))$ derived from $\mathcal{D}' = \{f(\boldsymbol{x}_i)\}$. The dotted ellipse is a Gaussian approximation to this target distribution, computed from the empirical moments. The solid ellipse is a Taylor transform. (c) Unscented sigma points. (d) Unscented transform. (e) Gauss-Hermite points (order 5). (f) GH transform. Adapted from Figures 5.3–5.4 of [Sar13]. Generated by gaussian_transforms.ipynb.

Algorithm 8.5: Computing sigma points using unscented transform.

def **SigmaPoints**$(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \alpha, \beta, \kappa)$:
$n =$ dimensionality of $\boldsymbol{\mu}$
$\lambda = \alpha^2(n + \kappa) - n$
Compute a set of $2n + 1$ sigma points:
$\mathcal{X}_0 = \boldsymbol{\mu},\ \mathcal{X}_i = \boldsymbol{\mu} + \sqrt{n + \lambda}\,[\sqrt{\boldsymbol{\Sigma}}]_{:i},\ \mathcal{X}_{i+n} = \boldsymbol{\mu} - \sqrt{n + \lambda}\,[\sqrt{\boldsymbol{\Sigma}}]_{:i}$
Compute a set of $2n + 1$ weights for the mean and covariance:
$w_0^m = \frac{\lambda}{n+\lambda},\ w_0^c = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta),\ w_i^m = w_i^c = \frac{1}{2(n+\lambda)}$
Return $(\mathcal{X}_{0:2n}, w_{0:2n}^m, w_{0:2n}^c)$

Algorithm 8.6: Unscented approximation to a joint Gaussian distribution.

def **UnscentedMoments**$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{g}, \boldsymbol{\Omega}; \alpha, \beta, \kappa)$:
$(\mathcal{X}_{0:2n}, w_{0:2n}^m, w_{0:2n}^c) =$ **SigmaPoints**$(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \alpha, \beta, \kappa)$
$\mathcal{Y}_i = \boldsymbol{g}(\mathcal{X}_i),\ i = 0 : 2n$
$\hat{\boldsymbol{y}} = \sum_{i=0}^{2n} w_i^m \mathcal{Y}_i$
$\mathbf{S} = \sum_{i=0}^{2n} w_i^c (\mathcal{Y}_i - \boldsymbol{\mu}_U)(\mathcal{Y}_i - \boldsymbol{\mu}_U)^\mathsf{T} + \boldsymbol{\Omega}$
$\mathbf{C} = \sum_{i=0}^{2n} w_i^c (\mathcal{X}_i - \boldsymbol{\mu})(\mathcal{Y}_i - \boldsymbol{\mu}_U)^\mathsf{T}$
Return $(\hat{\boldsymbol{y}}, \mathbf{S}, \mathbf{C})$

8.4.1 The unscented transform

Suppose we have two random variables $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$, where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$. The unscented transform forms a Gaussian approximation to $p(\boldsymbol{y})$ using the following process. First we compute a set of $2n+1$ sigma points, $\mathcal{X}_i$, and corresponding weights, w_i^m and w_i^c, using Algorithm 8.5, for $i = 0 : 2n$. (The notation $\mathbf{M}_{:i}$ means the i'th column of matrix $\mathbf{M}$, $\sqrt{\boldsymbol{\Sigma}}$ is the matrix square root, so $\sqrt{\boldsymbol{\Sigma}}\sqrt{\boldsymbol{\Sigma}}^\mathsf{T} = \boldsymbol{\Sigma}$.) Next we propagate the sigma points through the nonlinear function to get the following $2n + 1$ outputs:

$$\mathcal{Y}_i = \boldsymbol{g}(\mathcal{X}_i),\ i = 0 : 2n \tag{8.148}$$

Finally we estimate the mean and covariance of the resulting set of points:

$$\mathbb{E}\left[\boldsymbol{g}(\boldsymbol{x})\right] \approx \hat{\boldsymbol{y}} = \sum_{i=0}^{2n} w_i^m \mathcal{Y}_i \tag{8.149}$$

$$\mathrm{Cov}\left[\boldsymbol{g}(\boldsymbol{x})\right] \approx \mathbf{S}' = \sum_{i=0}^{2n} w_i^c (\mathcal{Y}_i - \hat{\boldsymbol{y}})(\mathcal{Y}_i - \hat{\boldsymbol{y}})^\mathsf{T} \tag{8.150}$$

Now suppose we want to approximate the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{e}$, and $\boldsymbol{e} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega})$. By defining the augmented function $\tilde{\boldsymbol{g}}(\boldsymbol{x}) = (\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x}))$, and applying the above

procedure (and adding extra noise), we get

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \hat{\boldsymbol{y}} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{C} \\ \mathbf{C}^{\mathsf{T}} & \mathbf{S} \end{pmatrix} \right) \tag{8.151}$$

where the parameters are computed using Algorithm 8.6.

The sigma points and their weights depend on three hyperparameters, α, β, and κ, which determine the spread of the sigma points around the mean. A typical recommended setting for these is $\alpha = 10^{-3}$, $\kappa = 1$, $\beta = 2$ [Bit16].

In Figure 8.5(a-b), we show the linearized Taylor transform discussed in Section 8.3.1 applied to a nonlinear function. In Figure 8.5(c-d), we show the corresponding unscented transform, which we can see is more accurate. In fact, the unscented transform (which uses $2n + 1$ sigma points) 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 (first order polynomials), because the square of a second order polynomial is already order 4. However, the UT idea can be extended to order 5 using $2n^2 + 1$ sigma points [MS67]; this can capture covariance terms exactly for quadratic functions. We discuss even more accurate approximations, based on numerical integration methods, in Section 8.5.1.4.

8.4.2 The unscented Kalman filter (UKF)

The UKF applies the unscented transform twice, once to approximate passing through the system model $\boldsymbol{f}$, and once to approximate passing through the measurement model $\boldsymbol{h}$. By analogy to Section 8.2.2.5, we can derive the UKF algorithm as follows:

$$(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \mathbf{UnscentedMoments}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}(\cdot, \boldsymbol{u}_t), \mathbf{Q}_t) \tag{8.152}$$

$$(\hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t) = \mathbf{UnscentedMoments}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{R}_t) \tag{8.153}$$

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \mathbf{GaussCondition}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t, \boldsymbol{y}_t) \tag{8.154}$$

See [SS23, Sec 8.8] for more details.

In Figure 8.4c, we illustrate the UKF algorithm (with $\alpha = 1$, $\beta = 0$, $\kappa = 2$) applied to the 2d nonlinear tracking problem from Section 8.3.2.3.

8.4.3 The unscented Kalman smoother (UKS)

The **unscented Kalman smoother**, also called the **unscented RTS smoother** [Sar08], is a simple modification of the usual Kalman smoothing method, where we approximate the nonlinearity by the unscented transform. The key insight is to notice that the reverse Kalman gain matrix $\mathbf{J}_t$ in Equation (8.80) can be defined in terms of the predicted covariance and cross covariance, both of which can be estimated using the UT. Once we have computed this, we can use the RTS equations in the usual way. See [SS23, Sec 14.4] for the details.

An interesting application of unscented Kalman smoothing was its use by the UK government as part of its COVID-19 contact tracing app [Lov+20; BCH20]. The app used the UKS to estimate the distance between (anonymized) people based on bluetooth signal strength between their mobile phones; the distance was then combined with other signals, such as contact duration and infectiousness level of the index case, to estimate the risk of transmission. (See also [MKS21] for a way to learn the risk score.)

8.5 Other variants of the Kalman filter

In this section, we briefly mention some other variants of Kalman filtering. For a more extensive review, see [Sar13; SS23; Li+17e].

8.5.1 General Gaussian filtering

This section is co-authored with Peter Chang.

Let $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $p(\boldsymbol{y}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\Omega})$ for some function $\boldsymbol{h}_{\boldsymbol{\mu}}$. 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 computed by solving

$$q(\boldsymbol{z}, \boldsymbol{y}) = \underset{q \in \mathcal{N}}{\operatorname{argmin}} \, D_{\mathbb{KL}} \left(p(\boldsymbol{z}, \boldsymbol{y}) \parallel q(\boldsymbol{z}, \boldsymbol{y}) \right) \tag{8.155}$$

As we explain in Section 5.1.4.2, this can be obtained by **moment matching**, i.e.,

$$q(\boldsymbol{z}, \boldsymbol{y}) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix} | \begin{pmatrix} \boldsymbol{\mu} \\ \hat{\boldsymbol{y}} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{C} \\ \mathbf{C}^{\mathsf{T}} & \mathbf{S} \end{pmatrix} \right) \tag{8.156}$$

where

$$\hat{\boldsymbol{y}} = \mathbb{E}\left[\boldsymbol{y}\right] = \int \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} \tag{8.157}$$

$$\mathbf{S} = \mathbb{V}\left[\boldsymbol{y}\right] = \boldsymbol{\Omega} + \int (\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})^{\mathsf{T}} \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} \tag{8.158}$$

$$\mathbf{C} = \operatorname{Cov}\left[\boldsymbol{z}, \boldsymbol{y}\right] = \int (\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})^{\mathsf{T}} \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} \tag{8.159}$$

We can use the above Gaussian approximation either for the time update (i.e., going from $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})$ to $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$ via $p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$), or for the measurement update, (i.e., going from $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$ to $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ via $p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$). For example, if the prior from the time update is $p(\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$, then the measurement update becomes

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1} \tag{8.160}$$

$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \tag{8.161}$$

$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathsf{T}} \tag{8.162}$$

The resulting method is called **general Gaussian filtering** or **GGF** [IX00; Wu+06].

8.5.1.1 Statistical linear regression

An alternative perspective on the above method is that we are approximating the likelihood by $q(\boldsymbol{y}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{y}|\mathbf{A}\boldsymbol{z} + \boldsymbol{b}, \boldsymbol{\Omega})$, where we define

$$\begin{aligned} \mathbf{A} &= \mathbf{C}^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} \\ \boldsymbol{b} &= \hat{\boldsymbol{y}} - \mathbf{A}\boldsymbol{\mu} \\ \boldsymbol{\Omega} &= \mathbf{S} - \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathsf{T}} \end{aligned} \tag{8.163}$$

This is called **statistical linear regression** or **SLR** [LBS01; AHE07], and ensures that we minimize

$$\mathcal{L}(\mathbf{A}, \boldsymbol{b}, \boldsymbol{\Omega}) = \mathbb{E}_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[D_{\mathbb{KL}}\left(p(\boldsymbol{y}|\boldsymbol{z}) \parallel q(\boldsymbol{y}|\boldsymbol{z}; \mathbf{A}, \boldsymbol{b}, \boldsymbol{\Omega})\right)\right] \tag{8.164}$$

For the proof, see [GF+15; Kam+22].

Equivalently, one can show that the above parameters minimize the following mean squared error

$$\mathcal{L}(\mathbf{A}, \boldsymbol{b}) = \mathbb{E}\left[(\boldsymbol{y} - \mathbf{A}\boldsymbol{x} - \boldsymbol{b})^{\mathsf{T}}(\boldsymbol{y} - \mathbf{A}\boldsymbol{x} - \boldsymbol{b})\right] \tag{8.165}$$

with $\boldsymbol{\Omega}$ given by the residual noise

$$\boldsymbol{\Omega} = \mathbb{E}\left[(\boldsymbol{y} - \mathbf{A}\boldsymbol{x} - \boldsymbol{b})(\boldsymbol{y} - \mathbf{A}\boldsymbol{x} - \boldsymbol{b})^{\mathsf{T}}\right] \tag{8.166}$$

See [SS23, Sec 9.4] for the proof.

Note that although SLR results in a linear model, it is different than the Taylor series approximation of Section 8.3.1, since the linearization is chosen to be optimal wrt a distribution of points (averaged over $\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$), instead of just being optimal at a single point $\boldsymbol{\mu}$.

8.5.1.2 Approximating the moments

To implement GGF, we need a way to compute $\hat{\boldsymbol{y}}$, $\mathbf{S}$ and $\mathbf{C}$. To help with this, we define two functions to compute Gaussian first and second moments:

$$g_e(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \int \boldsymbol{f}(\boldsymbol{z})\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{z} \tag{8.167}$$

$$g_c(\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \int (\boldsymbol{f}(\boldsymbol{z}) - \overline{\boldsymbol{f}})(\boldsymbol{g}(\boldsymbol{z}) - \overline{\boldsymbol{g}})^{\mathsf{T}}\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{z} \tag{8.168}$$

where $\overline{\boldsymbol{f}} = g_e(\boldsymbol{f}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\overline{\boldsymbol{g}} = g_e(\boldsymbol{g}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$. There are several ways to compute these integrals, as we discuss below.

8.5.1.3 Approximation based on linearization

The simplest approach to approximating the moments is to linearize the functions $\boldsymbol{f}$ and $\boldsymbol{g}$ around $\boldsymbol{\mu}$, which yields the following (see Section 8.3.1):

$$\hat{\boldsymbol{f}}(\boldsymbol{z}) = \boldsymbol{\mu} + \mathbf{F}(\boldsymbol{z} - \boldsymbol{\mu}) \tag{8.169}$$

$$\hat{\boldsymbol{g}}(\boldsymbol{z}) = \boldsymbol{\mu} + \mathbf{G}(\boldsymbol{z} - \boldsymbol{\mu}) \tag{8.170}$$

where $\mathbf{F}$ and $\mathbf{G}$ are the Jacobians of f and g. Thus we get the following implementation of the moment functions:

$$g_e(\hat{\boldsymbol{f}}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{E}\left[\boldsymbol{\mu} + \mathbf{F}(\boldsymbol{z} - \boldsymbol{\mu})\right] = \boldsymbol{\mu} \tag{8.171}$$

$$g_c(\hat{\boldsymbol{f}}, \hat{\boldsymbol{g}}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{E}\left[(\hat{\boldsymbol{f}}(\boldsymbol{z}) - \overline{\boldsymbol{f}})(\hat{\boldsymbol{g}}(\boldsymbol{z}) - \overline{\boldsymbol{g}})^\mathsf{T}\right] \tag{8.172}$$

$$= \mathbb{E}\left[\hat{\boldsymbol{f}}(\boldsymbol{z})\hat{\boldsymbol{g}}(\boldsymbol{z})^\mathsf{T} + \overline{\boldsymbol{f}}\overline{\boldsymbol{g}}^\mathsf{T} - \hat{\boldsymbol{f}}(\boldsymbol{z})\overline{\boldsymbol{g}}^\mathsf{T} - \overline{\boldsymbol{f}}\hat{\boldsymbol{g}}(\boldsymbol{z})^\mathsf{T}\right] \tag{8.173}$$

$$= \mathbb{E}\left[(\boldsymbol{\mu} + \mathbf{F}(\boldsymbol{z} - \boldsymbol{\mu}))(\boldsymbol{\mu} + \mathbf{G}(\boldsymbol{z} - \boldsymbol{\mu}))^\mathsf{T} + \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T} - \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T} - \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}\right] \tag{8.174}$$

$$= \mathbb{E}\left[\boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T} + \mathbf{F}(\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{z} - \boldsymbol{\mu})^\mathsf{T}\mathbf{G}^\mathsf{T} + \mathbf{F}(\boldsymbol{z} - \boldsymbol{\mu})\boldsymbol{\mu}^\mathsf{T} + \boldsymbol{\mu}(\boldsymbol{z} - \boldsymbol{\mu})^\mathsf{T}\mathbf{G}^\mathsf{T} - \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}\right] \tag{8.175}$$

$$= \mathbf{F}\mathbb{E}\left[(\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{z} - \boldsymbol{\mu})^\mathsf{T}\right]\mathbf{G}^\mathsf{T} = \mathbf{F}\boldsymbol{\Sigma}\mathbf{G}^\mathsf{T} \tag{8.176}$$

Using this inside the GGF is equivalent to the EKF in Section 8.3.2. However, this approach can lead to large errors and sometimes divergence of the filter [IX00; VDMW03].

8.5.1.4 Approximation based on Gaussian quadrature

Since we are computing integrals wrt a Gaussian measure, we can use **Gaussian quadrature** methods of the following form:

$$\int \boldsymbol{h}(\boldsymbol{z})\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\boldsymbol{z} \approx \sum_{k=1}^{K} w^k \boldsymbol{h}(\boldsymbol{z}^k) \tag{8.177}$$

for a suitable set of evaluation points $\boldsymbol{z}^k$ (sometimes called **sigma points**) and weights w^k. (Note that one-dimensional integrals are called **quadratures**, and multi-dimensional integrals are called **cubatures**.)

One way to compute the sigma points is to use the unscented transform described in Section 8.4.1. Using this inside the GGF is equivalent to the UKF in Section 8.4.2.

Alternatively, we can use **spherical cubature integration**, which gives rise to the **cubature Kalman filter** or **CKF** [AH09]. This turns out (see [SS23, Sec 8.7]) to be a special case of the UKF, with $2n_z + 1$ sigma points, and hyperparameter values 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. In particular, an order p approximation will be exact for polynomials of order up to $2p - 1$. See [SS23, Sec 8.3] for details, and Figure 8.5(e-f) for an illustration. However, this comes at a price: the number of sigma points is now p^n. Using Gauss-Hermite integration for GGF gives rise to the **Gauss-Hermite Kalman filter** or **GHKF** [IX00], also known as the **quadrature Kalman filter** or **QKF** [AHE07].

8.5.1.5 Approximation based on Monte Carlo integration

We can also approximate the integrals with Monte Carlo (see Section 11.2). Note, however, that this is not the same as particle filtering (Section 13.2), which approximates the conditional $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ rather than the joint $p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$ (see Section 8.6.1 for discussion of this difference).

8.5.2 Conditional moment Gaussian filtering

We can go beyond the Gaussian likelihood assumption by approximating the actual likelihood by a linear Gaussian model, as proposed in [TGFS18]. The only requirement is that we can compute the first and second **conditional moments** of the likelihood:

$$\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) = \mathbb{E}[\boldsymbol{y}|\boldsymbol{z}] = \int \boldsymbol{y} p(\boldsymbol{y}|\boldsymbol{z}) d\boldsymbol{y} \tag{8.178}$$

$$\boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}) = \mathrm{Cov}[\boldsymbol{y}|\boldsymbol{z}] = \int (\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}))(\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}))^{\mathsf{T}} p(\boldsymbol{y}|\boldsymbol{z}) d\boldsymbol{y} \tag{8.179}$$

Note that these integrals may be wrt a non-Gaussian measure $p(\boldsymbol{y}|\boldsymbol{z})$. Also, $\boldsymbol{y}$ may be discrete, in which case these integrals become sums.

Next we compute the unconditional moments. By the law of iterated expecations we have

$$\hat{\boldsymbol{y}} = \mathbb{E}[\boldsymbol{y}] = \mathbb{E}[\mathbb{E}[\boldsymbol{y}|\boldsymbol{z}]] = \int \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} = g_e(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{8.180}$$

Similarly

$$\mathbf{C} = \mathrm{Cov}[\boldsymbol{z}, \boldsymbol{y}] = \mathbb{E}[\mathbb{E}[(\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{y} - \hat{\boldsymbol{y}})|\boldsymbol{z}]] = \mathbb{E}[(\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})] \tag{8.181}$$

$$= \int (\boldsymbol{z} - \boldsymbol{\mu})(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})^{\mathsf{T}} \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} = g_c(\boldsymbol{z}, \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{8.182}$$

Finally

$$\mathbf{S} = \mathbb{V}[\boldsymbol{y}] = \mathbb{E}[\mathbb{V}[\boldsymbol{y}|\boldsymbol{z}]] + \mathbb{V}[\mathbb{E}[\boldsymbol{y}|\boldsymbol{z}]] \tag{8.183}$$

$$= \int \boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}) \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} + \int (\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}) - \hat{\boldsymbol{y}})^{\mathsf{T}} \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{z} \tag{8.184}$$

$$= g_e(\boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma}) + g_c(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{8.185}$$

Note that the equation for $\hat{\boldsymbol{y}}$ is the same in Equation (8.157) and Equation (8.180), and the equation for $\mathbf{C}$ is the same in Equation (8.159) and Equation (8.182). Furthermore, if $\boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}) = \boldsymbol{\Omega}$, then the equation for $\mathbf{S}$ is the same in Equation (8.158) and Equation (8.185).

We can approximate the unconditional moments using linearization or numerical integration. We can then plug them into the GGF algorithm. We call this **conditional moments Gaussian filtering** or **CMGF**.

We can use CMGF to perform approximate inference in SSMs with Poisson likelihoods. For example, if $p(y|z) = \mathrm{Poisson}(y|ce^z)$, we have

$$\boldsymbol{h}_{\boldsymbol{\mu}}(z) = \boldsymbol{h}_{\boldsymbol{\Sigma}}(z) = ce^z \tag{8.186}$$

This method can be used to perform (extended) Kalman filtering with more general exponential family likelihoods, as described in [TGFS18; Oll18]. For example, suppose we have a categorical likelihood:

$$p(y_t|\boldsymbol{z}_t) = \mathrm{Cat}(y_t|\boldsymbol{p}_t) = \mathrm{Cat}(y_t|\mathrm{softmax}(\boldsymbol{\eta}_t)) = \mathrm{Cat}(y_t|\mathrm{softmax}(h(\boldsymbol{z}_t)) \tag{8.187}$$

where $\boldsymbol{\eta}_t = h(\boldsymbol{z}_t)$ are the predicted logits. Then the conditional mean and covariance are given by

$$\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}_t) = \boldsymbol{p}_t = \text{softmax}(h(\boldsymbol{z}_t)),\ \boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}_t) = \text{diag}(\boldsymbol{p}_t) - \boldsymbol{p}_t \boldsymbol{p}_t^{\mathsf{T}} \tag{8.188}$$

(We can drop one of the classes from the vector $\boldsymbol{p}_t$ to ensure the covariance is full rank.) This approach can be used for online inference in neural network classifiers [CMJ22], as well as Gaussian process classifiers [GFTS19] and recommender systems [GU16; GUK21]. We can also use this method as a proposal distribution inside of a particle filtering algorithm (Section 13.2), as discussed in [Hos+20b].

8.5.3 Iterated filters and smoothers

The GGF method in Section 8.5.1, and the CMGF method in Section 8.5.2, both require computing moments wrt the predictive distribution $\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$ before performing the measurement update. It is possible to do one step of GGF to compute the posterior given the new observation, $\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$, and then to use this revised posterior to compute new moments in an iterated fashion. This is called **iterated posterior linearization filter** or **IPLF** [GF+15]. (This is similar to the iterated EKF which we discussed in Section 8.3.2.2.) See Algorithm 8.7 for the pseudocode, and [SS23, Sec 10.4] for more details.

Algorithm 8.7: Iterated conditional moments Gaussian filter.

def **Iterated-CMGF**$(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{h}_{\boldsymbol{\mu}}, \boldsymbol{h}_{\boldsymbol{\Sigma}}, \boldsymbol{y}_{1:T}, \boldsymbol{\mu}_{0|0}, \boldsymbol{\Sigma}_{0|0}, J, g_e, g_c)$:
  foreach $t = 1 : T$ **do**
    Predict step:
    $(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) =$ **CondMoments**$(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}, \mathbf{Q}, g_e, g_c)$
    Update step:
    $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1}$, $\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1}$
    foreach $j = 1 : J$ **do**
      $(\hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t) =$ **CondMoments**$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \boldsymbol{h}_{\boldsymbol{\mu}}, \boldsymbol{h}_{\boldsymbol{\Sigma}}, g_e, g_c)$
      $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) =$ **GaussCondition**$(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \hat{\boldsymbol{y}}_t, \mathbf{S}_t, \mathbf{C}_t, \boldsymbol{y}_t)$
  Return $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})_{t=1}^T$
def **CondMoments**$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{h}_{\boldsymbol{\mu}}, \boldsymbol{h}_{\boldsymbol{\Sigma}}, g_e, g_c)$:
  $\hat{\boldsymbol{y}} = g_e(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma})$
  $\mathbf{S} = g_e(\boldsymbol{h}_{\boldsymbol{\Sigma}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma}) + g_c(\boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma})$
  $\mathbf{C} = g_c(\boldsymbol{z}, \boldsymbol{h}_{\boldsymbol{\mu}}(\boldsymbol{z}), \boldsymbol{\mu}, \boldsymbol{\Sigma})$
  Return $(\hat{\boldsymbol{y}}, \mathbf{S}, \mathbf{C})$

In a similar way, we can derive the **iterated posterior linearization smoother** or **IPLS** [GFSS17]. This is similar to the iterated EKS which we discussed in Section 8.3.3.

Unfortunately the IPLF and IPLS can diverge. A more robust version of IPLF, that uses line search to perform damped (partial) updates, is presented in [Rai+18b]. Similarly, a more robust version of IPLS, that uses line search and Levenberg-Marquardt to update the parameters, is presented in [Lin+21c].

Various extensions of the above methods have been proposed. For example, in [HPR19] they extend IPLS to belief propagation in Forney factor graphs (Section 4.6.1.2), which enables the method to be applied to a large class of graphical models beyond SSMs. In particular, they give a general linearization formulation (including explicit message update rules) for nonlinear approximate Gaussian BP (Section 9.4.3) where the linearization can be Jacobian-based ("EKF-style"), statistical (moment matching), or anything else. They also show how any such linearization method can benefit from iterations.

In [Kam+22], they present a method based on approximate expectation propagation (Section 10.7), that is very similar to IPLS, except that the distributions that are used to compute the SLR terms, needed to compute the Gaussian messages, are different. In particular, rather than using the smoothed posterior from the last iteration, it uses the "cavity" distribution, which is the current posterior minus the incoming message that was sent at the last iteration, similar to Section 8.2.4.4. The advantage of this is that the outgoing message does not double count the evidence. The disadvantage is that this may be numerically unstable.

In [WSS21], they propose a variety of "**Bayes-Newton**" methods for approximately computing Gaussian posteriors to probabilistic models with nonlinear and/or non-Gaussian likelihoods. This generalizes all of the above methods, and can be applied to SSMs and GPs.

8.5.4 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. In particular, it is mostly used for problems where the hidden state represents an unknown physical quantity (e.g., temperature or 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**.

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].

The key idea is to represent the belief state $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ by a finite number of samples $\mathbf{Z}_{t|t} = \{\boldsymbol{z}_{t|t}^s : s = 1 : N_s\}$, where each $\boldsymbol{z}_{t|t}^s \in \mathbb{R}^{N_z}$. In contrast to particle filtering (Section 13.2), 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 $N_s \to \infty$ [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:

$$\tilde{\boldsymbol{z}}_{t|t} = \frac{1}{N_s}\sum_{s=1}^{N_s} \boldsymbol{z}_{t|t}^s = \frac{1}{N_s}\mathbf{Z}_{t|t}\mathbf{1} \tag{8.189}$$

$$\tilde{\boldsymbol{\Sigma}}_{t|t} = \frac{1}{N_s - 1}\sum_{s=1}^{N_s}(\boldsymbol{z}_t^s - \tilde{\boldsymbol{z}}_{t|t})(\boldsymbol{z}_t^s - \tilde{\boldsymbol{z}}_{t|t})^\mathsf{T} = \frac{1}{N_s - 1}\tilde{\mathbf{Z}}_{t|t}\tilde{\mathbf{Z}}_{t|t}^\mathsf{T} \tag{8.190}$$

where $\tilde{\mathbf{Z}}_{t|t} = \mathbf{Z}_{t|t} - \tilde{\boldsymbol{z}}_{t|t}\mathbf{1}^\mathsf{T} = \mathbf{Z}_{t|t}(\mathbf{I}_{N_s} - \frac{1}{N_s}\mathbf{1}\mathbf{1}^\mathsf{T})$.

We update the samples as follows. For the time update, we first draw N_s system noise variables $\boldsymbol{q}_t^s \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$, and then we pass these, and the previous state estimate, through the dynamics

model to get the one-step-ahead state predictions, $\boldsymbol{z}^s_{t|t-1} = \boldsymbol{f}(\boldsymbol{z}^s_{t-1|t-1}, \boldsymbol{q}^s_t)$, from which we get $\mathbf{Z}_{t|t-1} = \{\boldsymbol{z}^s_{t|t-1}\}$, which has size $N_z \times N_s$. Next we draw N_s observation noise variables $\boldsymbol{r}^s_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$, and use them to compute the one-step-ahead observation predictions, $\boldsymbol{y}^s_{t|t-1} = \boldsymbol{h}(\boldsymbol{z}^s_{t|t-1}, \boldsymbol{r}^s_t)$ and $\mathbf{Y}_{t|t-1} = \{\boldsymbol{y}^s_{t|t-1}\}$, which has size $N_y \times N_s$. Finally we compute the measurement update using

$$\mathbf{Z}_{t|t} = \mathbf{Z}_{t|t-1} + \tilde{\mathbf{K}}_t(\boldsymbol{y}_t \mathbf{1}^\mathsf{T} - \mathbf{Y}_{t|t-1}) \tag{8.191}$$

which is the analog of Equation (8.29).

We now discuss how to compute $\tilde{\mathbf{K}}_t$, which is the analog of the Kalman gain matrix in Equation (8.28). First note that we can write the exact Kalman gain matrix (in the linear-Gaussian case) as $\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{H}^\mathsf{T}\mathbf{S}_t^{-1} = \mathbf{C}_t\mathbf{S}_t^{-1}$, where $\mathbf{S}_t$ is the covariance of the predictive distribution for the observation vector at time t, and $\mathbf{C}_t$ is the cross-covariance of the joint predictive distribution for the next state and next observation. In the EnKF, we approximate $\mathbf{S}_t$ and $\mathbf{C}_t$ empirically as follows. First we compute the anomalies

$$\tilde{\mathbf{Z}}_{t|t-1} = \mathbf{Z}_{t|t-1} - \tilde{\boldsymbol{z}}_{t|t-1}\mathbf{1}^\mathsf{T}, \quad \tilde{\mathbf{Y}}_{t|t-1} = \mathbf{Y}_{t|t-1} - \tilde{\boldsymbol{y}}_{t|t-1}\mathbf{1}^\mathsf{T} \tag{8.192}$$

Then we compute the sample covariance matrices

$$\tilde{\mathbf{C}}_t = \frac{1}{N_s - 1}\tilde{\mathbf{Z}}_{t|t-1}\tilde{\mathbf{Y}}^\mathsf{T}_{t|t-1}, \quad \tilde{\mathbf{S}}_t = \frac{1}{N_s - 1}\tilde{\mathbf{Y}}_{t|t-1}\tilde{\mathbf{Y}}^\mathsf{T}_{t|t-1} \tag{8.193}$$

Finally we compute

$$\tilde{\mathbf{K}}_t = \tilde{\mathbf{C}}_t\tilde{\mathbf{S}}_t^{-1} \tag{8.194}$$

which has the same form as a multivariate least squares problem. For models with additive noise, we can reduce the variance of this procedure by eliminating the sampling of the predicted observations. Thus we replace $\tilde{\mathbf{Y}}_{t|t-1}$ with its deterministic version, $\tilde{\mathbf{O}}_{t|t-1} = \mathbf{H}\tilde{\mathbf{Z}}_{t|t-1}$ (assuming a linear observation model for notational simplicity). We then use $\tilde{\mathbf{C}}_t = \frac{1}{N_s-1}\tilde{\mathbf{Z}}_{t|t-1}\tilde{\mathbf{O}}^\mathsf{T}_{t|t-1}$, and $\tilde{\mathbf{S}}_t = \frac{1}{N_s-1}\tilde{\mathbf{O}}_{t|t-1}\tilde{\mathbf{O}}^\mathsf{T}_{t|t-1} + \mathbf{R}_t$. (It is also possible to eliminate the sampling for the latent states, by using the **ensemble square root filter** [Tip+03], although this may be less robust.)

We now compare the computational complexity to the KF algorithm. Recall that N_z is the number of latent dimensions, N_y is the number of observed dimensions, and N_s is the number of samples. We will assume $N_z > N_s > N_y$, as occurs in most geospatial problems. The EnKF time update takes $O(N_z^2 N_s)$ time to propagate the samples though the model (assuming that $\boldsymbol{f}$ is a linear model), whereas the KF takes $O(N_z^3)$ time to compute $\mathbf{F}_t\boldsymbol{\Sigma}_{t-1}\mathbf{F}_t + \mathbf{Q}_t$. If the transition matrix is sparse, the EnKF time reduces to $O(N_z N_s)$ and the EKF time reduces to (N_z^2). The EnKF measurement update takes $O(N_z N_y N_s)$ time to compute $\tilde{\mathbf{C}}_t$, $O(N_y^2 N_s)$ time to compute $\tilde{\mathbf{S}}_t$, $O(N_y^3)$ time to compute $\tilde{\mathbf{S}}_t^{-1}$, $O(N_z N_y^2)$ to compute $\tilde{\mathbf{K}}_t$, and $O(N_z N_y N_s)$ time to compute $\mathbf{Z}_{t|t}$, for a total of $O(N_z N_y N_s + N_s N_y^2)$, where we have dropped terms that don't depend on N_z for notational simplicity. By contrast, in the EKF, the measurement update takes $O(N_z^2 N_y)$ to compute $\mathbf{C}_t$, $O(N_z^2 N_y)$ to compute $\mathbf{S}_t$, $O(N_y^3)$ to compute $\mathbf{S}_t^{-1}$, $O(N_z N_y^2)$ to compute $\mathbf{K}_t$, and $O(N_z N_y)$ to compute $\boldsymbol{\mu}_{t|t}$, for a total of $O(N_z^2 N_y + N_z N_y^2)$. In summary, EnKF is $O(N_z N_s)$, but for EKF is $O(N_z^2)$.

Unfortunately, if N_s is too small, the EnKF can be become overconfident, and the filter can diverge. A common heuristic to reduce this is known as **covariance inflation**, in which we replace $\tilde{\mathbf{Z}}_{t|t-1}$ with $\tilde{\mathbf{Z}}_{t|t-1} = \beta(\mathbf{Z}_{t|t-1} - \tilde{\boldsymbol{z}}_{t|t-1}\mathbf{1}^\mathsf{T})$ for some fudge factor $\beta > 1$.

Unlike the particle filter, the EnKF is not guaranteed to converge to the correct posterior. However, hybrid PF/EnKF approaches have been developed (see e.g., [LGMT11; FK13b; Rei13]) with better theoretical foundations.

8.5.5 Robust Kalman filters

In practice we often have noise that is non-Gaussian. A common example is when we have clutter, or outliers, in the observation model, or sudden changes in the process model. In this case, we might use the Laplace distribution [Ara+09] or the Student t-distribution [Ara10; RÖG13; Ara+17] as noise models.

[Hua+17b] proposes a variational Bayes (Section 10.3.3) approach, that allows the dynamical prior and the observation model to both be (linear) Student distributions, but where the posterior is approximated at each step using a Gaussian, conditional on the noise scale matrix, which is modeled using an inverse Wishart distribution. An extension of this, to handle mixture distributions, can be found in [Hua+19].

8.5.6 Dual EKF

In this section, we briefly discuss one approach to estimating the parameters of an SSM. In an offline setting, we can use EM, SGD, or Bayesian inference to compute an approximation to $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T})$ (see Section 29.8). In the online setting, we want to compute $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t})$. We can do this by adding the parameters to the state space, possibly with an **artificial dynamics**, $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \epsilon\mathbf{I})$, and then performing joint inference of states and parameters. The latent variables at each step now contain the latent states, $\boldsymbol{z}_t$, and the latent parameters, $\boldsymbol{\theta}_t$. One approach to performing approximating inference in such a model is to use the **dual EKF**, in which one EKF performs state estimation and the other EKF performs parameter estimation [WN01].

8.5.7 Normalizing flow KFs

Normalizing flows, as discussed in Chapter 23, are a kind of deep generative model with a tractable exact likelihood. These can be used to "upgrade" the observation model of a linear Gaussian SSM, while still retaining tractable exact Gaussian inference, as shown in [Béz+20]. In particular, instead of observing $\boldsymbol{y}_t = \mathbf{H}_t\boldsymbol{z}_t + \boldsymbol{r}_t$, where $\boldsymbol{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$, we observe $\boldsymbol{y}_t = f_t(\boldsymbol{h}_t)$, where $\boldsymbol{h}_t = \mathbf{H}_t\boldsymbol{z}_t + \boldsymbol{r}_t$, and $f_t : \mathbb{R}^{N_y} \to \mathbb{R}^{N_y}$ is an invertible function with a tractable Jacobian. In this case, the exact posterior, $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$, is given by the usual Kalman filter equations applied to $\boldsymbol{h}_t = f_t^{-1}(\boldsymbol{y}_t)$ instead of $\boldsymbol{y}_t$, which we denote by $p_{\text{LGSSM}}(\boldsymbol{z}_t|\boldsymbol{h}_{1:t})$.

This result can be shown by induction. First note that, by the change of variable formula, $p(\boldsymbol{y}_t|\boldsymbol{z}_t) = p(\boldsymbol{h}_t|\boldsymbol{z}_t)Df_t^{-1}(\boldsymbol{y}_t)$, where $Dg(\boldsymbol{a}) \triangleq |\det \text{Jac}(g)(\boldsymbol{a})|$. By induction, we have $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1}) = p_{\text{LGSSM}}(\boldsymbol{z}_{t-}$ and hence by the linear Gaussian assumptions for the dynamics model, $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = p_{\text{LGSSM}}(\boldsymbol{z}_t|\boldsymbol{h}_{1:t-1})$ Thus the filtering posterior is given by

$$
\begin{aligned}
p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) &= \frac{p(\boldsymbol{y}_t|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{z}'_t)p(\boldsymbol{z}'_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}'_t} = \frac{Df_t^{-1}(\boldsymbol{y}_t)p(\boldsymbol{h}_t|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})}{\int Df_t^{-1}(\boldsymbol{y}_t)p(\boldsymbol{h}_t|\boldsymbol{z}'_t)p(\boldsymbol{z}'_t|\boldsymbol{y}_{1:t-1})} \\
&= \frac{p(\boldsymbol{h}_t|\boldsymbol{z}_t)p_{\text{LGSSM}}(\boldsymbol{z}_t|\boldsymbol{h}_{1:t-1})}{\int p(\boldsymbol{h}_t|\boldsymbol{z}'_t)p_{\text{LGSSM}}(\boldsymbol{z}'_t|\boldsymbol{h}_{1:t-1})} = p_{\text{LGSSM}}(\boldsymbol{z}_t|\boldsymbol{h}_{1:t})
\end{aligned}
$$

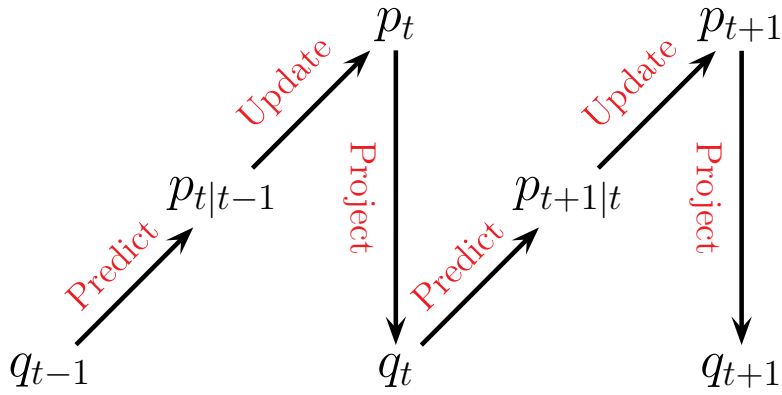


Figure 8.6: Illustration of the predict-update-project cycle of assumed density filtering. $q_t \in \mathcal{Q}$ is a tractable distribution, whereas we may have $p_{t|t-1} \notin \mathcal{Q}$ and $p_t \notin \mathcal{Q}$.

Similar reasoning applies to the smoothing distribution.

8.6 Assumed density filtering

In this section, we discuss **assumed density filtering** or **ADF** [May79]. In this approach, we *assume* the posterior has a specific form (e.g., a Gaussian). At each step, we update the previous posterior with the new likelihood; the result will often not have the desired form (e.g., will no longer be Gaussian), so we project it to the closest approximating distribution of the required type.

In more detail, we assume (by induction) that our prior $q_{t-1}(\boldsymbol{z}_{t-1}) \approx p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})$ satisfies $q_{t-1} \in \mathcal{Q}$, where $\mathcal{Q}$ is a family of tractable distributions. We can update the prior with the new measurement to get the approximate posterior as follows. First we compute the **one-step-ahead predictive distribution**

$$p_{t|t-1}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) q_{t-1}(\boldsymbol{z}_{t-1}) d\boldsymbol{z}_{t-1} \tag{8.195}$$

Then we update this prior with the likelihood for step t to get the posterior

$$p_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \frac{1}{Z_t} p(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{t|t-1}(\boldsymbol{z}_t) \tag{8.196}$$

where

$$Z_t = \int p(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{t|t-1}(\boldsymbol{z}_t) d\boldsymbol{z}_t \tag{8.197}$$

is the normalization constant. Unfortunately, we often find that the resulting posterior is no longer in our tractable family, $p(\boldsymbol{z}_t) \notin \mathcal{Q}$. So after Bayesian updating we seek the best tractable approximation by computing

$$q_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathbb{KL}}\left(p_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) \parallel q(\boldsymbol{z}_t)\right) \tag{8.198}$$

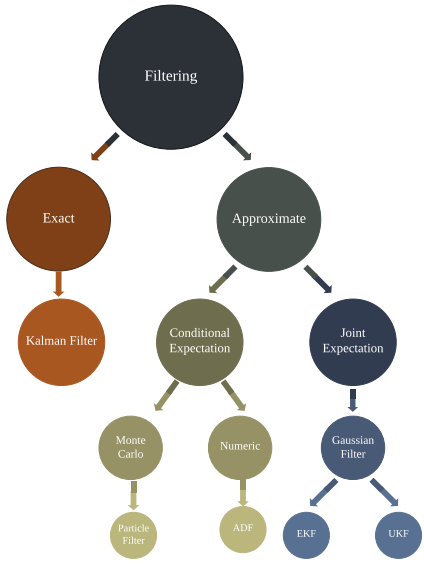


Figure 8.7: A taxonomy of filtering algorithms. Adapted from Figure 2 of [Wüt+16].

This minimizes the Kullback-Leibler divergence from the approximation $q(\boldsymbol{z}_t)$ to the "exact" posterior $p_t(\boldsymbol{z}_t)$, and can be thought of as **projecting** p onto the space of tractable distributions. Thus the overall algorithm consists of three steps — predict, update, and project — as sketched in Figure 8.6.

Computing $\min_q D_{\mathbb{KL}}(p \parallel q)$ is known as **moment projection**, since the optimal q should have the same moments as p (see Section 5.1.4.2). So in the Gaussian case, we just need to set the mean and covariance of q_t so they are the same as the mean and covariance of p_t. We will give some examples of this below. By contrast, computing $\min_q D_{\mathbb{KL}}(q \parallel p)$, as in variational inference (Section 10.1), is known as **information projection**, and will result in mode seeking behavior (see Section 5.1.4.1), rather than trying to capture overall moments.

8.6.1 Connection with Gaussian filtering

When $\mathcal{Q}$ is the set of Gaussian distributions, there is a close connection between ADF and Gaussian filtering, which we discussed in Section 8.5.1. GF corresponds to solving the following optimization problem

$$q_{t|t-1}(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\mathbb{KL}}\left(p(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t | \boldsymbol{y}_{1:t-1}) \parallel q(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t | \boldsymbol{y}_{1:t-1})\right) \tag{8.199}$$

which can be solved by moment matching (see Section 8.5.1). We then condition this joint distribution on the event $\tilde{\boldsymbol{y}}_t = \boldsymbol{y}_t$, where $\tilde{\boldsymbol{y}}_t$ is the unknown random variable and $\boldsymbol{y}_t$ is its observed value. This gives $p_t(\boldsymbol{z}_t | \boldsymbol{y}_{1:t})$, which is easy to compute, due to the Gaussian assumption. By contrast, in Gaussian ADF, we first compute the (locally) exact posterior $p_t(\boldsymbol{z}_t | \boldsymbol{y}_{1:t})$, and then approximate it with $q_t(\boldsymbol{z}_t | \boldsymbol{y}_{1:t})$ by projecting into $\mathcal{Q}$. Thus ADF approximates the conditional $p_t(\boldsymbol{z}_t | \boldsymbol{y}_{1:t})$, whereas GF approximates the joint $p_{t|t-1}(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t | \boldsymbol{y}_{1:t-1})$, from which we derive $p_t(\boldsymbol{z}_t | \boldsymbol{y}_{1:t})$ by conditioning.

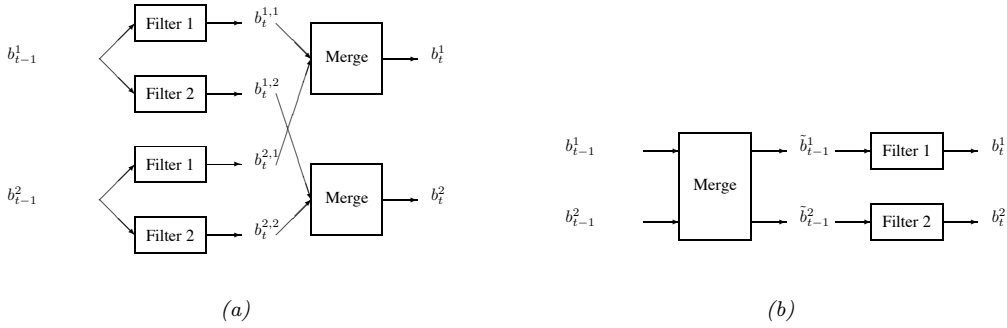


Figure 8.8: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

ADF is more accurate than GF, since it directy approximates the posterior, but it is more computationally demanding, for reasons explained in [Wüt+16]. However, in [Kam+22] they propose an approximate form of expectation propagation (which is a generalization of ADF) in which the messages are computed using the same local joint Gaussian approximation as used in Gaussian filtering. See Figure 8.7 for a summary of how these different methods relate.

8.6.2 ADF for SLDS (Gaussian sum filter)

In this section, we apply ADF to inference in switching linear dynamical systems (SLDS, Section 29.9), which are a combination of HMM and LDS models. The resulting method is known as the **Gaussian sum filter** (see e.g., [Cro+11; Wil+17]).

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:

$$b_{t-1}^i \triangleq p(\boldsymbol{z}_{t-1}, m_{t-1} = i|\boldsymbol{y}_{1:t-1}) = \pi_{t-1|t-1}^i \mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1|t-1}^i, \boldsymbol{\Sigma}_{t-1|t-1}^i) \tag{8.200}$$

where $i \in \{1, \ldots, K\}$. We then pass this through the K different linear models to get

$$b_t^{ij} \triangleq p(\boldsymbol{z}_t, m_{t-1} = i, m_t = j|\boldsymbol{y}_{1:t}) = \pi_{t|t}^{ij} \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}^{ij}, \boldsymbol{\Sigma}_{t|t}^{ij}) \tag{8.201}$$

where $\pi_{t|t}^{ij} = \pi_{t-1|t-1}^i A_{ij}$, where $A_{ij} = p(m_t = j|m_{t-1} = i)$. Finally, for each value of j, we collapse the K Gaussian mixtures down to a single mixture to give

$$b_t^j \triangleq p(\boldsymbol{z}_t, m_t = j|\boldsymbol{y}_{1:t}) = \pi_{t|t}^j \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_{t|t}^j, \boldsymbol{\Sigma}_{t|t}^j) \tag{8.202}$$

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,

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{z}] = \sum_k \pi^k \boldsymbol{\mu}^k \tag{8.203}$$

$$\boldsymbol{\Sigma} = \mathrm{Cov}[\boldsymbol{z}] = \sum_k \pi^k \left(\boldsymbol{\Sigma}^k + (\boldsymbol{\mu}^k - \boldsymbol{\mu})(\boldsymbol{\mu}^k - \boldsymbol{\mu})^\mathsf{T} \right) \tag{8.204}$$

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):

$$\pi^j = \sum_i \pi^{ij} \tag{8.205}$$

$$\pi^{j|i} = \frac{\pi^{ij}}{\sum_{j'} \pi^{ij'}} \tag{8.206}$$

$$\boldsymbol{\mu}^j = \sum_i \pi^{j|i} \boldsymbol{\mu}^{ij} \tag{8.207}$$

$$\boldsymbol{\Sigma}^j = \sum_i \pi^{j|i} \left(\boldsymbol{\Sigma}^{ij} + (\boldsymbol{\mu}^{ij} - \boldsymbol{\mu}^j)(\boldsymbol{\mu}^{ij} - \boldsymbol{\mu}_j)^\mathsf{T} \right) \tag{8.208}$$

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 m_t. See Figure 8.8b for a sketch.

8.6.3 ADF for online logistic regression

In this section we discuss the application of ADF to online Bayesian parameter inference for a binary logistic regression model, based on [Zoe07]. The overall approach is similar to the online linear regression case (discussed in Section 29.7.2), but approximates the posterior after each update step, which is necessary since the likelihood is not conjugate to the prior.

We assume our model has the following form:

$$p(y_t|\boldsymbol{x}_t, \boldsymbol{w}_t) = \mathrm{Ber}(y_t|\sigma(\boldsymbol{x}_t^\mathsf{T}\boldsymbol{w}_t)) \tag{8.209}$$

$$p(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}) = \mathcal{N}(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}, \mathbf{Q}) \tag{8.210}$$

where $\mathbf{Q}$ is the covariance of the process noise, which allows the parameters to change slowly over time. We will assume $\mathbf{Q} = \epsilon\mathbf{I}$; we can also set $\epsilon = 0$, as in the recursive least squares method (Section 29.7.2), if we believe the parameters will not change. See Figure 8.9 for an illustration of the model.

As our approximating family, we will use diagonal Gaussians, for computational efficiency. Thus the prior is the posterior from the previous time step, and has the form

$$p(\boldsymbol{w}_{t-1}|\mathcal{D}_{1:t-1}) \approx p_{t-1}(\boldsymbol{w}_{t-1}) = \prod_j \mathcal{N}(w_{t-1}^j|\mu_{t-1|t-1}^j, \tau_{t-1|t-1}^j) \tag{8.211}$$

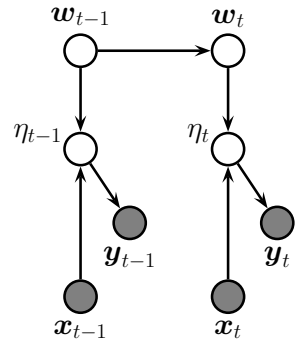


Figure 8.9: A dynamic logistic regression model. $\boldsymbol{w}_t$ are the regression weights at time t, and $\eta_t = \boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{x}_t$. Compare to Figure 29.24a.

where $\mu^j_{t-1|t-1}$ and $\tau^j_{t-1|t-1}$ are the posterior mean and variance for parameter j given past data. Now we discuss how to update this prior.

First we compute the one-step-ahead predictive density $p_{t|t-1}(\boldsymbol{w}_t)$ using the standard linear-Gaussian update, i.e., $\boldsymbol{\mu}_{t|t-1} = \boldsymbol{\mu}_{t-1|t-1}$ and $\boldsymbol{\tau}_{t|t-1} = \boldsymbol{\tau}_{t-1|t-1} + \mathbf{Q}$, where we can set $\mathbf{Q} = 0\mathbf{I}$ if there is no drift.

Now we concentrate on the measurement update step. Define the scalar sum (corresponding to the logits, if we are using binary classification) as $\eta_t = \boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{x}_t$. If $p_{t|t-1}(\boldsymbol{w}_t) = \prod_j \mathcal{N}(w_t^j | \mu^j_{t|t-1}, \tau^j_{t|t-1})$, then we can compute the 1d prior predictive distribution for η_t as follows:

$$p(\eta_t | \mathcal{D}_{1:t-1}, \boldsymbol{x}_t) \approx p_{t|t-1}(\eta_t) = \mathcal{N}(\eta_t | m_{t|t-1}, v_{t|t-1}) \tag{8.212}$$

$$m_{t|t-1} = \sum_j x_{t,j} \mu^j_{t|t-1} \tag{8.213}$$

$$v_{t|t-1} = \sum_j x^2_{t,j} \tau^j_{t|t-1} \tag{8.214}$$

The posterior for the 1d η_t is given by

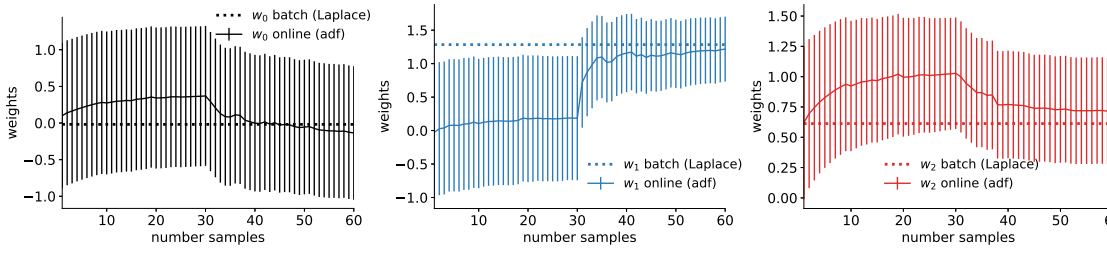
$$p(\eta_t | \mathcal{D}_{1:t}) \approx p_t(\eta_t) = \mathcal{N}(\eta_t | m_t, v_t) \tag{8.215}$$

$$m_t = \int \eta_t \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \tag{8.216}$$

$$v_t = \int \eta_t^2 \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t - m_t^2 \tag{8.217}$$

$$Z_t = \int p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \tag{8.218}$$

where $p(y_t|\eta_t) = \text{Ber}(y_t|\eta_t)$. These integrals are one dimensional, and so can be efficiently computed using Gaussian quadrature, as explained in [Zoe07; KB00].



(a) (b) (c)

Figure 8.10: 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 marginal posterior mean and variance for each parameter vs time as computed by ADF. The dotted horizontal line is the offline Laplace approximation. Generated by adf_logistic_regression_demo.ipynb.

Having inferred $p_t(\eta_t)$, we need to compute $p_t(\boldsymbol{w}|\eta_t)$. This can be done as follows. Define δ_m as the change in the mean and δ_v as the change in the variance:

$$m_t = m_{t|t-1} + \delta_m, \;\; v_t = v_{t|t-1} + \delta_v \tag{8.219}$$

Using the fact that $p(\eta_t|\boldsymbol{w}) = \mathcal{N}(\eta_t|\boldsymbol{w}^\mathsf{T}\eta_t, 0)$ is a linear Gaussian system, with prior $p(\boldsymbol{w}) = p(\boldsymbol{w}|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\tau}_{t|t-1})$ and "soft evidence" $p(\eta_t) = \mathcal{N}(m_t, v_t)$, we can derive the posterior for $p(\boldsymbol{w}|\mathcal{D}_t)$ as follows:

$$p_t(w_t^i) = \mathcal{N}(w_t^i|\mu_{t|t}^i, \tau_{t|t}^i) \tag{8.220}$$

$$\mu_{t|t}^i = \mu_{t|t-1}^i + a_i \delta_m \tag{8.221}$$

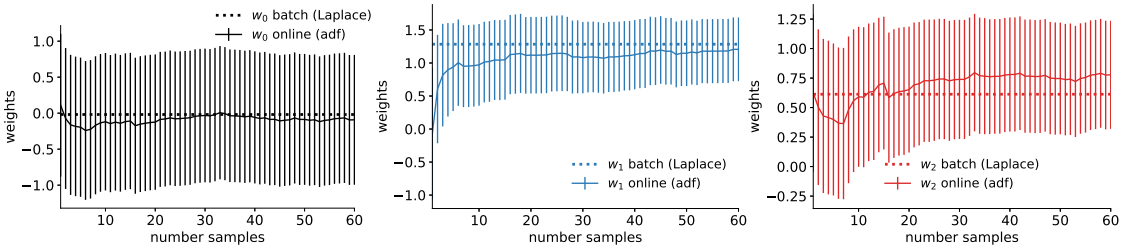
$$\tau_{t|t}^i = \tau_{t|t-1}^i + a_i^2 \delta_v \tag{8.222}$$

$$a_i \triangleq \frac{x_t^i \tau_{t|t-1}^i}{\sum_j (x_t^j)^2 + \tau_{t|t-1}^j} \tag{8.223}$$

Thus we see that the parameters which correspond to inputs i with larger magnitude (big $|x_t^i|$) or larger uncertainty (big $\tau_{t|t-1}^i$) get updated most, due to a large a_i factor, which makes intuitive sense.

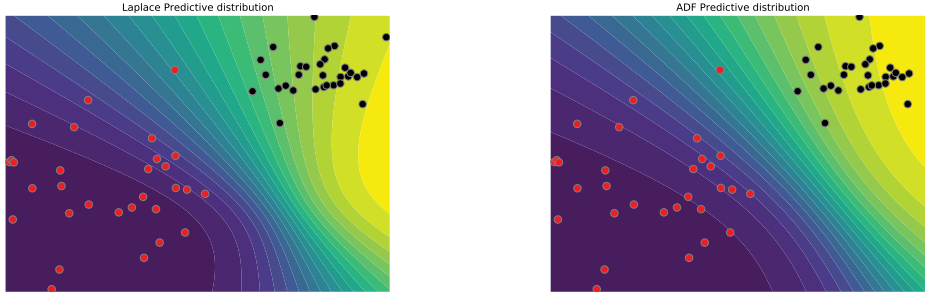
As an example, we consider a 2d binary classification problem. We sequentially compute the posterior using the ADF, and compare to the offline estimate computed using a Laplace approximation. In Figure 8.10 we plot the posterior marginals over the 3 parameters as a function of "time" (i.e., after conditioning on each training example one). We see that we converge to the offline MAP estimate. In Figure 8.11, we show the results of performing sequential Bayesian updating in a different ordering of the data. We still converge to approximate the same answer. In Figure 8.12, we see that the resulting posterior predictive distributions from the Laplace estimate and ADF estimate (at the end of training) are similar.

Note that the whole algorithm only takes $O(D)$ time and space per step, the same as SGD. However, unlike SGD, there are no step-size parameters, since the diagonal covariance implicitly specifies the size of the update for each dimension. Furthermore, we get a posterior approximation, not just a point estimate.



(a) (b) (c)

Figure 8.11: Same as Figure 8.10, except the order in which the data is visited is different. Generated by adf_logistic_regression_demo.ipynb.



(a) (b)

Figure 8.12: Predictive distribution for the binary logistic regression problem. (a) Result from Laplace approximation. (b) Result from ADF at the final step. Generated by adf_logistic_regression_demo.ipynb.

The overall approach is very similar to the generalized posterior linearization filter of Section 8.5.3, which uses quadrature (or the unscented transform) to compute a Gaussian approximation to the joint $p(y_t, \boldsymbol{w}_t|\mathcal{D}_{1:t-1})$, from which we can easily compute $p(\boldsymbol{w}_t|\mathcal{D}_{1:t})$. However, ADF approximates the posterior rather than the joint, as explained in Section 8.6.1.

8.6.4 ADF for online DNNs

In Section 17.5.3, we show how to use ADF to recursively approximate the posterior over the parameters of a deep neural network in an online fashion. This generalizes Section 8.6.3 to the case of nonlinear models.

8.7 Other inference methods for SSMs

There are a variety of other inference algorithms that can be applied to SSMs. We give a very brief summary below. For more details, see e.g., [Dau05; Sim06; Fra08; Sar13; SS23; Tri21].

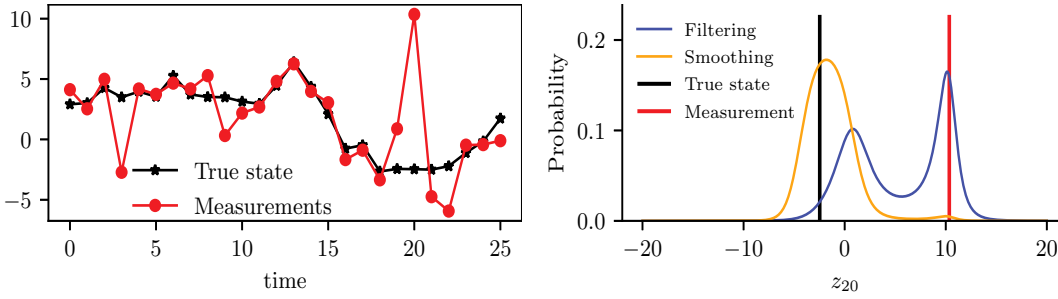


Figure 8.13: (a) Observations and true and estimated state. (b) Marginal distributions for time step $t = 20$. Generated by discretized_ssm_student.ipynb.

8.7.1 Grid-based approximations

A very simple approach to approximate inference in SSMs is to discretize the state space, and then to apply the HMM filter and smoother (see Section 9.2.3), as proposed in [RG17]. This is called a **grid-based approximation**. Unfortunately, this approach will not scale to higher dimensional problems, due to the curse of dimensionality. In particular, we know that the HMM filter takes $O(K^2)$ operations per time step, if there are K states. If we have N_z dimensions, each discretized into B bins, then we have $K = B^{N_z}$, so the approach quickly becomes intractable.

However, this approach can be useful in 1d or 2d. As an illustration, consider a simple 1d SSM with linear dynamics corrupted by additive Student noise:

$$z_t = z_{t-1} + \mathcal{T}_2(0, 1) \tag{8.224}$$

The observations are also linear, and are also corrupted by additive Student noise:

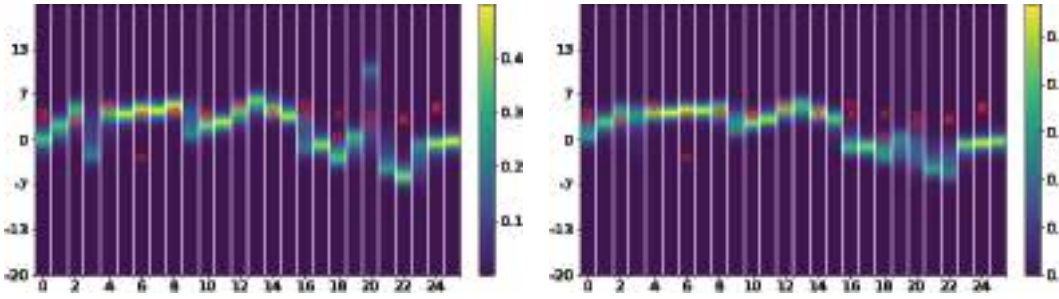
$$y_t = z_t + \mathcal{T}_2(0, 1) \tag{8.225}$$

This robust observation model is useful when there are potential outliers in the observed data, such as at time $t = 20$ in Figure 8.13a. (See also Section 8.5.5 for discussion of robust Kalman filters.)

Unfortunately the use of a non-Gaussian likelihood means that the resulting posterior can become multimodal. Fortunately, this is not a problem for the grid-based approach. We show the results for filtering and smoothing in Figure 8.14a and in Figure 8.14b. We see that at $t = 20$, the filtering distribution, $p(z_t|\boldsymbol{y}_{1:20})$, is bimodal, with a mean that is quite far from the true state (see Figure 8.13b for a detailed plot). Such a multimodal distribution can be approximated by a suitably fine discretization.

8.7.2 Expectation propagation

In Section 10.7 we discuss the expectation propagation (EP) algorithm, which can be viewed as an iterative version of ADF (Section 8.6). In particular, at each step we combine each exact local likelihood factor with approximate factors from both the past filtering distribution and the future smoothed posterior; these factors are combined to compute the locally exact posterior, which is then



(a) (b)

Figure 8.14: Discretized posterior of the latent state at each time step. Red cross is the true latent state. Red circle is observation. (a) Filtering. (b) Smoothing. Generated by discretized_ssm_student.ipynb.

projected back to the tractable family (e.g., Gaussian), before moving to the next time step. This process can be iterated for increased accuracy. In many cases the local EP update is intractable, but we can make a local Gaussian approximation, similar to the one in general Gaussian filtering (Section 8.5.1), as explained in [Kam+22].

8.7.3 Variational inference

EP can be viewed as locally minimizing the inclusive KL, $D_{\mathbb{KL}}\left(p(\boldsymbol{z}_t|\boldsymbol{y}_{1:T}) \parallel q(\boldsymbol{z}_t|\boldsymbol{y}_{1:T})\right)$, for each time step t. An alternative approach is to globally minimize the exclusive KL, $D_{\mathbb{KL}}\left(q(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}) \parallel p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}\right.$ this is called variational inference, and is explained in Chapter 10. The difference between these two objectives is discussed in more detail in Section 5.1.4.1, but from a practical point of view, the main advantage of VI is that we can derive a tractable lower bound to the objective, and can then optimize it using stochastic optimization. This method is guaranteed to converge, unlike EP. For more details on VI applied to SSMs (both state estimation and parameter estimation), see e.g., [CWS21; Cou+20; Cou+21; BFY20; FLMM21; Cam+21].

8.7.4 MCMC

In Chapter 12 we discuss Markov chain Monte Carlo (MCMC) methods, which can be used to draw samples from intractable posteriors. In the case of SSMs, this includes both the distribution over states, $p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T})$, and the distribution over parameters, $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T})$. In some cases, such as when using HMMs or linear-Gaussian SSMs, we can perform blocked Gibbs sampling, in which we use forwards filtering backwards sampling to sample an entire sequence from $p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}, \boldsymbol{\theta})$, followed by sampling the parameters, $p(\boldsymbol{\theta}|\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T})$ (see e.g., [CK96; Sco02; CMR05] for details.) Alternatively we can marginalize out the hidden states and just compute the parameter posterior $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T})$. When state inference is intractable, we can use gradient-based HMC methods (assuming the states are continuous), although this does not scale well to long sequences.

8.7.5 Particle filtering

In Section 13.2 we discuss particle filtering, which is a form of sequential Bayesian inference for SSMs which replaces the assumption that the posterior is (approximately) Gaussian with a more flexible representation, namely a set of weighted samples called "particles" (see e.g., [Aru+02; DJ11; NLS19]). Essentially the technique amounts to a form of importance sampling, combined with steps to prevent "particle impoverishment", which refers to some samples receiving negligible weight because they are too improbable in the posterior (which grows with time). Particle filtering is widely used because it is very flexible, and has good theoretical properties. In practice it may require many samples to get a good approximation, but we can use heuristic methods, such as the extended or unscented Kalman filters, as proposal distributions, which can improve the efficiency significantly. In the offline setting, we can use particle smoothing (Section 13.5) or SMC (sequential Monte Carlo) samplers (Section 13.6).

9 Message passing algorithms

9.1 Introduction

In this chapter we consider posterior inference (i.e., computing marginals, modes, samples, etc) for probability distributions that can be represented by a probabilistic graphical model (PGM, Chapter 4) with some kind of sparse graph structure (i.e., it is not a fully connected graph). The algorithms we discuss will leverage the conditional independence properties encoded in the graph structure (discussed in Chapter 4) in order to perform efficient inference. In particular, we will use the principle of **dynamic programming** (DP), which finds an optimal solution by solving subproblems and then combining them.

DP can be implemented by computing local quantities for each node (or clique) in the graph, and then sending **messages** to neighboring nodes (or cliques) so that all nodes (cliques) can come to an overall consensus about the global solutions. Hence these are known as **message passing algorithms**. Each message can be intepreted as probability distribution about the value of a node given evidence from part of the graph. These distributions are often called **belief states**, so these algorithms are also called **belief propagation** (**BP**) algorithms.

In Section 9.2, we consider the special case where the graph structure is a 1d chain, which is an important special case. (For a chain, a natural approach is to send messages forwards in time, and then backwards in time, so this method can also be used for inference in state space models, as we discuss in Chapter 8.) In Section 9.3, we can generalize this approach to work with trees, and in Section 9.4, we generalize it work with any graph, including ones with cycles or loops. However, sending messages on loopy graphs may give incorrect answers. In such cases, we may wish to convert the graph to a tree, and then send messages on it, using the methods discussed in Section 9.5 and Section 9.6. We can also pose the inference problem as an optimization problem, as we discuss in Section 9.7.

9.2 Belief propagation on chains

In this section, we consider inference for PGMs where the graph structure is a 1d chain. For notational simplicity, we focus on the case where the graphical model is directed rather than undirected, although the resulting methods are easy to generalize. In addition, we only consider the case where all the hidden variables are discrete; we discuss generalizations to handle continuous latent variables in Chapter 8 and Chapter 13.

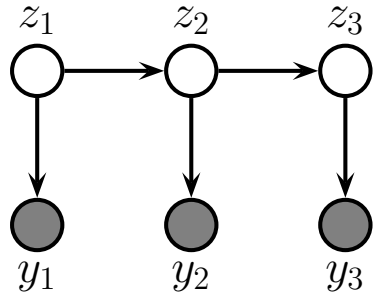


Figure 9.1: An HMM represented as a graphical model. $\boldsymbol{z}_t$ are the hidden variables at time t, $\boldsymbol{y}_t$ are the observations (outputs).

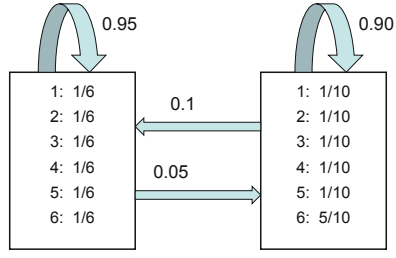


Figure 9.2: The state transition matrix **A** *and observation matrix* **B** *for the casino HMM. Adapted from [Dur+98, p54].*

9.2.1 Hidden Markov Models

In this section we assume the graphical model can be represented as a state space model, as shown in Figure 9.1. We discuss SSMs in more detail in Chapter 29, but we can think of them as latent variable sequence models with the conditional independencies shown by the chain-structured graphical model Figure 8.1. The corresponding joint distribution has the form

$$p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}) = \left[p(\boldsymbol{z}_1)\prod_{t=2}^{T} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})\right]\left[\prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{z}_t)\right] \tag{9.1}$$

where $\boldsymbol{z}_t$ are the hidden variables at time t, and $\boldsymbol{y}_t$ are the observations (outputs). If all the latent variables are discrete (as we assume in this section), the resulting model is called a **hidden Markov model** or **HMM**. We consider SSMs with continuous latent variables in Chapter 8.

9.2.1.1 Example: casino HMM

As a concrete example from [Dur+98], we consider the **occasionally dishonest casino**. We assume we are in a casino and observe a series of die rolls, $y_t \in \{1, 2, \ldots, 6\}$. Being a keen-eyed statistician, we notice that the distribution of values is not what we expect from a fair die: it seems that there

are occasional "streaks", in which 6s seem to show up more often than other values. We would like to estimate the underlying state, namely whether the die is fair or loaded, so that we make predictions about the future.

To formalize this, let $z_t \in \{1, 2\}$ represent the unknown hidden state (fair or loaded) at time t, and let $y_t \in \{1, \ldots, 6\}$ represent the observed outcome (die roll). Let $A_{jk} = p(z_t = k|z_{t-1} = j)$ be the state transition matrix. Most of the time the casino uses a fair die, $z = 1$, but occasionally it switches to a loaded die, $z = 2$, for a short period, as shown in the state transition diagram in Figure 9.2.

Let $B_{kl} = p(y_t = l|z_t = k)$ be the observation matrix corresponding to a categorical distribution over values of the die face. If $z = 1$ the observation distribution is a uniform categorical distribution over the symbols $\{1, \ldots, 6\}$. If $z = 2$, the observation distribution is skewed towards face 6. That is,

$$p(y_t|z_t = 1) = \text{Cat}(y_t|[1/6, \ldots, 1/6]) \tag{9.2}$$

$$p(y_t|z_t = 2) = \text{Cat}(y_t|[1/10, 1/10, 1/10, 1/10, 1/10, 5/10]) \tag{9.3}$$

If we sample from this model, we may generate data such as the following:

```
hid:  1111111111222211111111111111111111112222222221222211111111111111111111
obs:  1355534526553366316351551526232112113462221263264265422344645323242361
```

Here obs refers to the observation and hid refers to the hidden state (1 is fair and 2 is loaded). In the full sequence of length 300, we find the empirical fraction of times that we observe a 6 in hidden state 1 to be 0.149, and in state 2 to be 0.472, which are very close to the expected fractions. (See casino_hmm.ipynb for the code.)

9.2.1.2 Posterior inference

Our goal is to infer the hidden states by computing the posterior over all the hidden nodes in the model, $p(z_t|y_{1:T})$. This is called the **smoothing distribution**. By the Markov property, we can break this into two terms:

$$p(\boldsymbol{z}_t = j|\boldsymbol{y}_{t+1:T}, \boldsymbol{y}_{1:t}) \propto p(\boldsymbol{z}_t = j, \boldsymbol{y}_{t+1:T}|\boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_t = j|\boldsymbol{y}_{1:t})p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j, \cancel{\boldsymbol{y}_{1:t}}) \tag{9.4}$$

We will first compute the **filtering distribution** $p(\boldsymbol{z}_t = j|\boldsymbol{y}_{1:t})$ by working forwards in time. We then compute the $p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j)$ terms by working backwards in time, and then we finally combine both terms. Both passes take (TK^2) time, where K is the number of discrete hidden states. We give the details below.

9.2.2 The forwards algorithm

As we discuss in Section 8.1.2, the **Bayes filter** is an algorithm for recursively computing the **belief state** $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ given the prior belief from the previous step, $p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})$, the new observation $\boldsymbol{y}_t$, and the model. In the HMM literature, this is known as the **forwards algorithm**.

In an HMM, the latent states $\boldsymbol{z}_t$ are discrete, so we can define the belief state as a vector, $\alpha_t(j) \triangleq p(z_t = j|\boldsymbol{y}_{1:t})$, the local evidence as another vector, $\lambda_t(j) \triangleq p(\boldsymbol{y}_t|z_t = j)$, and the transition matrix as $A_{i,j} = p(z_t = j|z_{t-1} = i)$. Then the predict step becomes

$$\alpha_{t|t-1}(j) \triangleq p(z_t = j|\boldsymbol{y}_{1:t-1}) = \sum_i p(z_t = j|z_{t-1} = i)p(z_{t-1} = i|\boldsymbol{y}_{1:t-1}) = \sum_i A_{i,j}\alpha_{t-1}(i) \tag{9.5}$$

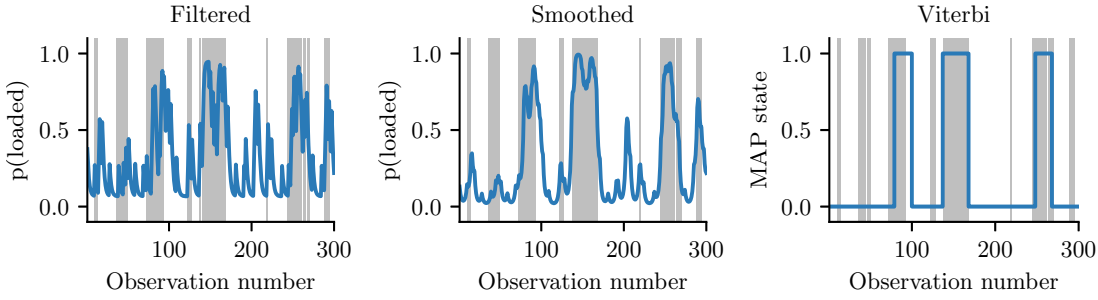


Figure 9.3: Inference in the dishonest casino. Vertical gray bars denote times when the hidden state corresponded to the loaded die. Blue lines represent the posterior probability of being in that state given different subsets of observed data. If we recover the true state exactly, the blue curve will transition at the same time as the gray bars. (a) Filtered estimates. (b) Smoothed estimates. (c) MAP trajectory. Generated by casino_hmm.ipynb.

and the update step becomes

$$\alpha_t(j) = \frac{1}{Z_t} p(\boldsymbol{y}_t|z_t = j)p(z_t = j|\boldsymbol{y}_{1:t-1}) = \frac{1}{Z_t}\lambda_t(j)\alpha_{t|t-1}(j) = \frac{1}{Z_t}\lambda_t(j)\left[\sum_i \alpha_{t-1}(i)A_{i,j}\right] \tag{9.6}$$

where the normalization constant for each time step is given by

$$Z_t \triangleq p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}) = \sum_{j=1}^{K} p(\boldsymbol{y}_t|\boldsymbol{z}_t = j)p(\boldsymbol{z}_t = j|\boldsymbol{y}_{1:t-1}) = \sum_{j=1}^{K} \lambda_t(j)\alpha_{t|t-1}(j) \tag{9.7}$$

We can write the update equation in matrix-vector notation as follows:

$$\boldsymbol{\alpha}_t = \text{normalize}\left(\boldsymbol{\lambda}_t \odot (\mathbf{A}^\mathsf{T}\boldsymbol{\alpha}_{t-1})\right) \tag{9.8}$$

where $\odot$ represents elementwise vector multiplication, and the normalize function just ensures its argument sums to one. (See Section 9.2.3.4 for more discussion on normalization.)

Figure 9.3(a) illustrates filtering for the casino HMM, applied to a random sequence $\boldsymbol{y}_{1:T}$ of length $T = 300$. In blue, we plot the probability that the die is in the loaded (vs fair) state, based on the evidence seen so far. The gray bars indicate time intervals during which the generative process actually switched to the loaded die. We see that the probability generally increases in the right places.

9.2.3 The forwards-backwards algorithm

In this section, we present the most common approach to smoothing in HMMs, known as the **forwards-backwards** or **FB** algorithm [Rab89]. In the forwards pass, we compute $\alpha_t(j) = p(\boldsymbol{z}_t = j|\boldsymbol{y}_{1:t})$ as before. In the backwards pass, we compute the conditional likelihood

$$\beta_t(j) \triangleq p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j) \tag{9.9}$$

We then combine these using

$$\gamma_t(j) = p(\boldsymbol{z}_t = j|\boldsymbol{y}_{t+1:T}, \boldsymbol{y}_{1:t}) \propto p(\boldsymbol{z}_t = j, \boldsymbol{y}_{t+1:T}|\boldsymbol{y}_{1:t}) \tag{9.10}$$

$$= p(\boldsymbol{z}_t = j|\boldsymbol{y}_{1:t})p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j, \cancel{\boldsymbol{y}_{1:t}}) = \alpha_t(j)\beta_t(j) \tag{9.11}$$

In matrix notation, this becomes

$$\boldsymbol{\gamma}_t = \text{normalize}(\boldsymbol{\alpha}_t \odot \boldsymbol{\beta}_t) \tag{9.12}$$

Note that the forwards and backwards passes can be computed independently, but both need access to the local evidence $p(\boldsymbol{y}_t|\boldsymbol{z}_t)$. The results are only combined at the end. This is therefore called **two-filter smoothing** [Kit04].

9.2.3.1 Backwards recursion

We can recursively compute the β's in a right-to-left fashion as follows:

$$\beta_{t-1}(i) = p(\boldsymbol{y}_{t:T}|\boldsymbol{z}_{t-1} = i) \tag{9.13}$$

$$= \sum_j p(\boldsymbol{z}_t = j, \boldsymbol{y}_t, \boldsymbol{y}_{t+1:T}|\boldsymbol{z}_{t-1} = i) \tag{9.14}$$

$$= \sum_j p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j, \cancel{\boldsymbol{y}_t}, \cancel{\boldsymbol{z}_t \boldsymbol{\xi}_1 \boldsymbol{\xi}} i)p(\boldsymbol{z}_t = j, \boldsymbol{y}_t|\boldsymbol{z}_{t-1} = i) \tag{9.15}$$

$$= \sum_j p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t = j)p(\boldsymbol{y}_t|\boldsymbol{z}_t = j, \cancel{\boldsymbol{z}_t \boldsymbol{\xi}_1 \boldsymbol{\xi}} i)p(\boldsymbol{z}_t = j|\boldsymbol{z}_{t-1} = i) \tag{9.16}$$

$$= \sum_j \beta_t(j)\lambda_t(j)A_{i,j} \tag{9.17}$$

We can write the resulting equation in matrix-vector form as

$$\boldsymbol{\beta}_{t-1} = \mathbf{A}(\boldsymbol{\lambda}_t \odot \boldsymbol{\beta}_t) \tag{9.18}$$

The base case is

$$\beta_T(i) = p(\boldsymbol{y}_{T+1:T}|\boldsymbol{z}_T = i) = p(\emptyset|\boldsymbol{z}_T = i) = 1 \tag{9.19}$$

which is the probability of a non-event.

Note that $\boldsymbol{\beta}_t$ is not a probability distribution over states, since it does not need to satisfy $\sum_j \beta_t(j) = 1$. However, we usually normalize it to avoid numerical underflow (see Section 9.2.3.4).

9.2.3.2 Example

In Figure 9.3(a-b), we compare filtering and smoothing for the casino HMM. We see that the posterior distributions when conditioned on all the data (past and future) are indeed smoother than when just conditioned on the past (filtering).

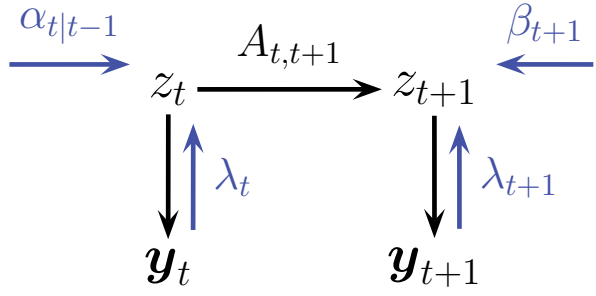


Figure 9.4: Computing the two-slice joint distribution for an HMM from the forwards messages, backwards messages, and local evidence messages.

9.2.3.3 Two-slice smoothed marginals

We can compute the two-slice marginals using the output of the forwards-backwards algorithm as follows:

$$
\begin{aligned}
p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:T}) &= p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}, \boldsymbol{y}_{t+1:T}) && (9.20)\\
&\propto p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:t}) p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}) && (9.21)\\
&= p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_{t+1}) p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{y}_{1:t}) && (9.22)\\
&= p(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_{t+1}) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) && (9.23)\\
&= p(\boldsymbol{y}_{t+1}, \boldsymbol{y}_{t+2:T}|\boldsymbol{z}_{t+1}) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) && (9.24)\\
&= p(\boldsymbol{y}_{t+1}|\boldsymbol{z}_{t+1}) p(\boldsymbol{y}_{t+2:T}|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) && (9.25)\\
&= p(\boldsymbol{y}_{t+1}|\boldsymbol{z}_{t+1}) p(\boldsymbol{y}_{t+2:T}|\boldsymbol{z}_{t+1}) p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t) && (9.26)
\end{aligned}
$$

We can rewrite this in terms of the already computed quantities as follows:

$$\xi_{t,t+1}(i,j) \propto \lambda_{t+1}(j) \beta_{t+1}(j) \alpha_t(i) A_{i,j} \tag{9.27}$$

Or in matrix-vector form:

$$\boldsymbol{\xi}_{t,t+1} \propto \mathbf{A} \odot \left[\boldsymbol{\alpha}_t (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\mathsf{T}\right] \tag{9.28}$$

Since $\boldsymbol{\alpha}_t \propto \boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}$, we can also write the above equation as follows:

$$\boldsymbol{\xi}_{t,t+1} \propto \mathbf{A} \odot \left[(\boldsymbol{\lambda}_t \odot \boldsymbol{\alpha}_{t|t-1}) \odot (\boldsymbol{\lambda}_{t+1} \odot \boldsymbol{\beta}_{t+1})^\mathsf{T}\right] \tag{9.29}$$

This can be interpreted as a product of incoming messages and local factors, as shown in Figure 9.4. In particular, we combine the factors $\boldsymbol{\alpha}_{t|t-1} = p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$, $\mathbf{A} = p(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)$, $\boldsymbol{\lambda}_t \propto p(\boldsymbol{y}_t|\boldsymbol{z}_t)$, $\boldsymbol{\lambda}_{t+1} \propto p(\boldsymbol{y}_{t+1}|\boldsymbol{z}_{t+1})$, and $\boldsymbol{\beta}_{t+1} \propto p(\boldsymbol{y}_{t+2:T}|\boldsymbol{z}_{t+1})$ to get $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, \boldsymbol{y}_t, \boldsymbol{y}_{t+1}, \boldsymbol{y}_{t+2:T}|\boldsymbol{y}_{1:t-1})$, which we can then normalize.

9.2.3.4 Numerically stable implementation

In most publications on HMMs, such as [Rab89], the forwards message is defined as the following unnormalized *joint* probability:

$$\alpha'_t(j) = p(\boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) = \lambda_t(j) \left[\sum_i \alpha'_{t-1}(i) A_{i,j} \right] \tag{9.30}$$

We instead define the forwards message as the normalized *conditional* probability

$$\alpha_t(j) = p(\boldsymbol{z}_t = j | \boldsymbol{y}_{1:t}) = \frac{1}{Z_t} \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A_{i,j} \right] \tag{9.31}$$

The unnormalized (joint) form has several problems. First, it rapidly suffers from numerical underflow, since the probability of the joint event that $(\boldsymbol{z}_t = j, \boldsymbol{y}_{1:t})$ is vanishingly small.[1] Second, it is less interpretable, since it is not a distribution over states. Third, it precludes the use of approximate inference methods that try to approximate posterior distributions (we will see such methods later). We therefore always use the normalized (conditional) form.

Of course, the two definitions only differ by a multiplicative constant, since $p(\boldsymbol{z}_t = j | \boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) / p(\boldsymbol{y}_{1:t})$ [Dev85]. So the *algorithmic* difference is just one line of code (namely the presence or absence of a call to the `normalize` function). Nevertheless, we feel it is better to present the normalized version, since it will encourage readers to implement the method properly (i.e., normalizing after each step to avoid underflow).

In practice it is more numerically stable to compute the log probabilities $\ell_t(j) = \log p(\boldsymbol{y}_t | \boldsymbol{z}_t = j)$ of the evidence, rather than the probabilities $\lambda_t(j) = p(\boldsymbol{y}_t | \boldsymbol{z}_t = j)$. We can combine the state conditional log likelihoods $\lambda_t(j)$ with the state prior $p(\boldsymbol{z}_t = j | \boldsymbol{y}_{1:t-1})$ by using the log-sum-exp trick, as in Equation (28.30).

9.2.4 Forwards filtering backwards smoothing

An alternative way to perform offline smoothing is to use forwards filtering/backwards smoothing, as discussed in Section 8.1.3. In this approach, we first perform the forwards or filtering pass, and then compute the smoothed belief states by working backwards, from right (time $t = T$) to left ($t = 1$). This approach is widely used for SSMs with continuous latent states, since the backwards likelihood $\beta_t(i)$ used in Section 9.2.3 is not always well defined when the state space is not discrete.

We assume by induction that we have already computed

$$\gamma_{t+1}(j) \triangleq p(\boldsymbol{z}_{t+1} = j | \boldsymbol{y}_{1:T}) \tag{9.32}$$

1. For example, if the observations are independent of the states, we have $p(\boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_t = j) \prod_{i=1}^{t} p(\boldsymbol{y}_i)$, which becomes exponentially small with t.

We then compute the smoothed joint distribution over two consecutive time steps:

$$\xi_{t,t+1}(i,j) \triangleq p(\boldsymbol{z}_t = i, \boldsymbol{z}_{t+1} = j|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_t = i|\boldsymbol{z}_{t+1} = j, \boldsymbol{y}_{1:t})p(\boldsymbol{z}_{t+1} = j|\boldsymbol{y}_{1:T}) \tag{9.33}$$

$$= \frac{p(\boldsymbol{z}_{t+1} = j|\boldsymbol{z}_t = i)p(\boldsymbol{z}_t = i|\boldsymbol{y}_{1:t})p(\boldsymbol{z}_{t+1} = j|\boldsymbol{y}_{1:T})}{p(\boldsymbol{z}_{t+1=j}|\boldsymbol{y}_{1:t})} \tag{9.34}$$

$$= \alpha_t(i)A_{i,j}\frac{\gamma_{t+1}(j)}{\alpha_{t+1|t}(j)} \tag{9.35}$$

where

$$\alpha_{t+1|t}(j) = p(\boldsymbol{z}_{t+1} = j|\boldsymbol{y}_{1:t}) = \sum_{i'} A(i',j)\alpha_t(i') \tag{9.36}$$

is the one-step-ahead predictive distribution. We can interpret the ratio in Equation (9.35) as dividing out the old estimate of $\boldsymbol{z}_{t+1}$ given $\boldsymbol{y}_{1:t}$, namely $\alpha_{t+1|t}$, and multiplying in the new estimate given $\boldsymbol{y}_{1:T}$, namely γ_{t+1}.

Once we have the two sliced smoothed distribution, we can easily get the marginal one slice smoothed distribution using

$$\gamma_t(i) = p(\boldsymbol{z}_t = i|\boldsymbol{y}_{1:T}) = \sum_j \xi_{t,t+1}(i,j) = \alpha_t(i)\sum_j \left[A_{i,j}\frac{\gamma_{t+1}(j)}{\alpha_{t+1|t}(j)}\right] \tag{9.37}$$

We initialize the recursion using $\gamma_T(j) = \alpha_T(j) = p(\boldsymbol{z}_T = j|\boldsymbol{y}_{1:T})$.

9.2.5 Time and space complexity

It is clear that a straightforward implementation of the forwards-backwards algorithm takes $O(K^2T)$ time, since we must perform a $K \times K$ matrix multiplication at each step. For some applications, such as speech recognition, K is very large, so the $O(K^2)$ term becomes prohibitive. Fortunately, if the transition matrix is sparse, we can reduce this substantially. For example, in a sparse left-to-right transition matrix (e.g., Figure 9.6(a)), the algorithm takes $O(TK)$ time.

In some cases, we can exploit special properties of the state space, even if the transition matrix is not sparse. In particular, suppose the states represent a discretization of an underlying continuous state-space, and the transition matrix has the form $A_{i,j} \propto \rho(\boldsymbol{z}_j - \boldsymbol{z}_i)$, where $\boldsymbol{z}_i$ is the continuous vector represented by state i and $\rho(u)$ is some scalar cost function, such as Euclidean distance. Then one can implement the forwards-backwards algorithm in $O(TK \log K)$ time. The key is to rewrite Equation (9.5) as a convolution,

$$\alpha_{t|t-1}(j) = p(z_t = j|\boldsymbol{y}_{1:t-1}) = \sum_i \alpha_{t-1}(i)A_{i,j} = \sum_i \alpha_{t-1}(i)\rho(j-i) \tag{9.38}$$

and then to apply the Fast Fourier Transform. (A similar transformation can be applied in the backwards pass.) This is very useful for models with large state spaces. See [FHK03] for details.

We can also reduce inference to $O(\log T)$ time by using a **parallel prefix scan** operator that can be run efficiently on GPUs. For details, see [HSGF21].

In some cases, the bottleneck is memory, not time. In particular, to compute the posteriors $\boldsymbol{\gamma}_t$, we must store the fitered distributions $\boldsymbol{\alpha}_t$ for $t = 1, \ldots, T$ until we do the backwards pass. It is possible

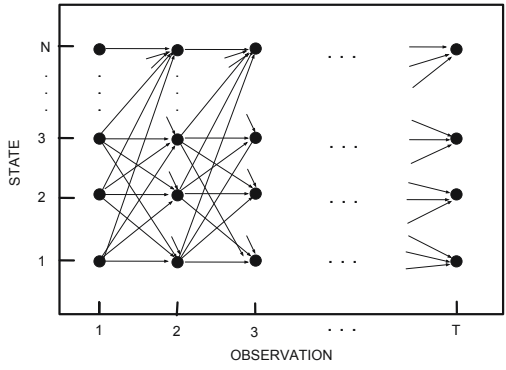


Figure 9.5: The trellis of states vs time for a Markov chain. Adapted from [Rab89].

to devise a simple divide-and-conquer algorithm that reduces the space complexity from $O(KT)$ to $O(K \log T)$ at the cost of increasing the running time from $O(K^2T)$ to $O(K^2T \log T)$. The basic idea is to store $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ vectors at a logarithmic number of intermediate checkpoints, and then recompute the missing messages on demand from these checkpoints. See [BMR97; ZP00] for details.

9.2.6 The Viterbi algorithm

The MAP estimate is (one of) the sequences with maximum posterior probability:

$$\boldsymbol{z}^*_{1:T} = \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}}\, p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}) = \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}} \log p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}) \tag{9.39}$$

$$= \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}} \log \pi_1(\boldsymbol{z}_1) + \log \lambda_1(\boldsymbol{z}_1) + \sum_{t=2}^{T} [\log A(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t) + \log \lambda_t(\boldsymbol{z}_t)] \tag{9.40}$$

This is equivalent to computing a shortest path through the **trellis diagram** in Figure 9.5, where the nodes are possible states at each time step, and the node and edge weights are log probabilities. This can be computed in $O(TK^2)$ time using the **Viterbi algorithm** [Vit67], as we explain below.

9.2.6.1 Forwards pass

Recall the (unnormalized) forwards equation

$$\alpha'_t(j) = p(\boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) = \sum_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}} p(\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) \tag{9.41}$$

Now suppose we replace sum with max to get

$$\delta_t(j) \triangleq \max_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}} p(\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}) \tag{9.42}$$

This is the maximum probability we can assign to the data so far if we end up in state j. The key insight is that the most probable path to state j at time t must consist of the most probable path to

some other state i at time $t-1$, followed by a transition from i to j. Hence

$$\delta_t(j) = \lambda_t(j) \left[\max_i \delta_{t-1}(i) A_{i,j}\right] \tag{9.43}$$

We initialize by setting $\delta_1(j) = \pi_j \lambda_1(j)$.

We often work in the log domain to avoid numerical issues. Let $\delta'_t(j) = -\log \delta_t(j)$, $\lambda'_t(j) = -\log p(\boldsymbol{y}_t|\boldsymbol{z}_t = j)$, $A'(i,j) = -\log p(\boldsymbol{z}_t = j|\boldsymbol{z}_{t-1} = i)$. Then we have

$$\delta'_t(j) = \lambda'_t(j) + \left[\min_i \delta'_{t-1}(i) + A'(i,j)\right] \tag{9.44}$$

We also need to keep track of the most likely previous (**ancestor**) state, for each possible state that we end up in:

$$a_t(j) \triangleq \operatorname*{argmax}_i \delta_{t-1}(i) A_{i,j} = \operatorname*{argmin}_i \delta'_{t-1}(i) + A'(i,j) \tag{9.45}$$

That is, $a_t(j)$ stores the identity of the previous state on the most probable path to $\boldsymbol{z}_t = j$. We will see why we need this in Section 9.2.6.2.

9.2.6.2 Backwards pass

In the backwards pass, we compute the most probable sequence of states using a **traceback** procedure, as follows: $\boldsymbol{z}^*_t = a_{t+1}(\boldsymbol{z}^*_{t+1})$, where we initialize using $\boldsymbol{z}^*_T = \arg\max_i \delta_T(i)$. This is just following the chain of ancestors along the MAP path.

If there is a unique MAP estimate, the above procedure will give the same result as picking $\hat{z}_t = \operatorname{argmax}_j \gamma_t(j)$, computed by forwards-backwards, as shown in [WF01b]. However, if there are multiple posterior modes, the latter approach may not find any of them, since it chooses each state independently, and hence may break ties in a manner that is inconsistent with its neighbors. The traceback procedure avoids this problem, since once $\boldsymbol{z}_t$ picks its most probable state, the previous nodes condition on this event, and therefore they will break ties consistently.

9.2.6.3 Example

In Figure 9.3(c), we show the Viterbi trace for the casino HMM. We see that, most of the time, the estimated state corresponds to the true state.

In Figure 9.6, we give a detailed worked example of the Viterbi algorithm, based on [Rus+95]. Suppose we observe the sequence of discrete observations $\boldsymbol{y}_{1:4} = (C_1, C_3, C_4, C_6)$, representing codebook entries in a vector-quantized version of a speech signal. The model starts in state $\boldsymbol{z}_1 = S_1$. The probability of generating $x_1 = C_1$ in $\boldsymbol{z}_1$ is 0.5, so we have $\delta_1(1) = 0.5$, and $\delta_1(i) = 0$ for all other states. Next we can self-transition to S_1 with probability 0.3, or transition to S_2 with proabability 0.7. If we end up in S_1, the probability of generating $x_2 = C_3$ is 0.3; if we end up in S_2, the probability of generating $x_2 = C_3$ is 0.2. Hence we have

$$\delta_2(1) = \delta_1(1) A(1,1) \lambda_2(1) = 0.5 \cdot 0.3 \cdot 0.3 = 0.045 \tag{9.46}$$

$$\delta_2(2) = \delta_1(1) A(1,2) \lambda_2(2) = 0.5 \cdot 0.7 \cdot 0.2 = 0.07 \tag{9.47}$$

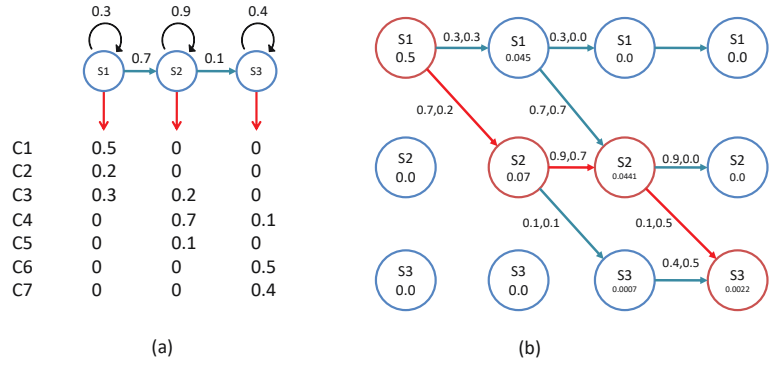


Figure 9.6: Illustration of Viterbi decoding in a simple HMM for speech recognition. (a) A 3-state HMM for a single phone. We are visualizing the state transition diagram. We assume the observations have been vector quantized into 7 possible symbols, $C_1, \ldots, C_7$. Each state S_1, S_2, S_3 has a different distribution over these symbols. Adapted from Figure 15.20 of [RN02]. (b) Illustration of the Viterbi algorithm applied to this model, with data sequence C_1, C_3, C_4, C_6. The columns represent time, and the rows represent states. The numbers inside the circles represent the $\delta_t(j)$ value for that state. An arrow from state i at $t-1$ to state j at t is annotated with two numbers: the first is the probability of the $i \to j$ transition, and the second is the probability of generating observation $\boldsymbol{y}_t$ from state j. The red lines/circles represent the most probable sequence of states. Adapted from Figure 24.27 of [RN95].

Thus state 2 is more probable at $t = 2$; see the second column of Figure 9.6(b). The algorithm continues in this way until we have reached the end of the sequence. One we have reached the end, we can follow the red arrows back to recover the MAP path (which is 1,2,2,3).

For more details on HMMs for automatic speech recognition (ASR) see e.g., [JM08].

9.2.6.4 Time and space complexity

The time complexity of Viterbi is clearly $O(K^2T)$ in general, and the space complexity is $O(KT)$, both the same as forwards-backwards. If the transition matrix has the form $A_{i,j} \propto \rho(\boldsymbol{z}_j - \boldsymbol{z}_i)$, where $\boldsymbol{z}_i$ is the continuous vector represented by state i and $\rho(u)$ is some scalar cost function, such as Euclidean distance, we can implement Viterbi in $O(TK)$ time, by using the generalized distance transform to implement Equation (9.44). See [FHK03; FH12] for details.

9.2.6.5 N-best list

There are often multiple paths which have the same likelihood. The Viterbi algorithm returns one of them, but can be extended to return the top N paths [SC90; NG01]. This is called the **N-best list**. Computing such a list can provide a better summary of the posterior uncertainty.

In addition, we can perform **discriminative reranking** [CK05] of all the sequences in $\mathcal{L}_N$, based on global features derived from $(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T})$. This technique is widely used in speech recognition. For example, consider the sentence "recognize speech". It is possible that the most probable interpretation by the system of this acoustic signal is "wreck a nice speech", or maybe "wreck a nice beach" (see

Figure 34.3). Maybe the correct interpretation is much lower down on the list. However, by using a re-ranking system, we may be able to improve the score of the correct interpretation based on a more global context.

One problem with the N-best list is that often the top N paths are very similar to each other, rather than representing qualitatively different interpretations of the data. Instead we might want to generate a more diverse set of paths to more accurately represent posterior uncertainty. One way to do this is to sample paths from the posterior, as we discuss in Section 9.2.7. Another way is to use a determinantal point process (Supplementary Section 31.8.5) which encourages points to be diverse [Bat+12; ZA12].

9.2.7 Forwards filtering backwards sampling

Rather than computing the single most probable path, it is often useful to sample multiple paths from the posterior: $\boldsymbol{z}_{1:T}^s \sim p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T})$. We can do this by modifying the forwards filtering backwards smoothing algorithm from Section 9.2.4, so that we draw samples on the backwards pass, rather than computing marginals. This is called **forwards filtering backwards sampling** (also sometimes unfortunately abbreviated to FFBS). In particular, note that we can write the joint from right to left using

$$p(\boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_T|\boldsymbol{y}_{1:T})p(\boldsymbol{z}_{T-1}|\boldsymbol{z}_T, \boldsymbol{y}_{1:T})p(\boldsymbol{z}_{T-2}|\boldsymbol{z}_{T-1}, \cancel{\boldsymbol{z}_T}, \boldsymbol{y}_{1:T}) \cdots p(\boldsymbol{z}_1|\boldsymbol{z}_2, \cancel{\boldsymbol{z}_{3:T}}, \boldsymbol{y}_{1:T}) \tag{9.48}$$

$$= p(\boldsymbol{z}_T|\boldsymbol{y}_{1:T}) \prod_{t=T-1}^{1} p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}, \boldsymbol{y}_{1:T}) \tag{9.49}$$

Thus at step t we sample $\boldsymbol{z}_t^s$ from $p(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}^s, \boldsymbol{y}_{1:T})$ given in Equation (9.49).

9.3 Belief propagation on trees

The forwards-backwards algorithm for HMMs discussed in Section 9.2.3 (and the Kalman smoother algorithm for LDS which we discuss in Section 8.2.3) can be interpreted as a message passing algorithm applied to a chain structured graphical model. In this section, we generalize these algorithms to work with trees.

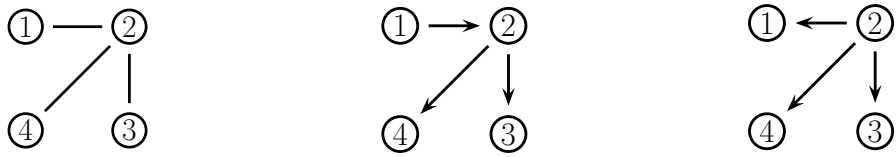
9.3.1 Directed vs undirected trees

Consider a pairwise *undirected* graphical model, which can be written as follows:

$$p^*(\boldsymbol{z}) \triangleq p(\boldsymbol{z}|\boldsymbol{y}) \propto \prod_{s \in \mathcal{V}} \psi_s(z_s|\boldsymbol{y}_s) \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(z_s, z_t) \tag{9.50}$$

where $\psi_{s,t}(z_s, z_t)$ are the pairwise clique potential, one per edge, $\psi_s(z_s|\boldsymbol{y}_s)$ are the local evidence potentials, one per node, $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. (We will henceforth drop the conditioning on the observed values $\boldsymbol{y}$ for brevity.)

Now suppose the corresponding graph structure is a tree, such as the one in Figure 9.7a. We can always convert this into a directed tree by picking an arbitrary node as the root, and then "picking



(a) (b) (c)

Figure 9.7: An undirected tree and two equivalent directed trees.

the tree up by the root" and orienting all the edges away from the root. For example, if we pick node 1 as the root we get Figure 9.7b. This corresponds to the following directed graphical model:

$$p^*(\boldsymbol{z}) \propto p^*(z_1)p^*(z_2|z_1)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.51}$$

However, if we pick node 2 as the root, we get Figure 9.7c. This corresponds to the following directed graphical model:

$$p^*(\boldsymbol{z}) \propto p^*(z_2)p^*(z_1|z_2)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.52}$$

Since these graphs express the same conditional independence properties, they represent the same family of probability distributions, and hence we are free to use any of these parameterizations.

To make the model more symmetric, it is preferable to use an undirected tree. If we define the potentials as (possibly unnormalized) marginals (i.e., $\psi_s(z_s) \propto p^*(z_s)$ and $\psi_{s,t}(z_s, z_t) = p^*(z_s, z_t)$), then we can write

$$p^*(\boldsymbol{z}) \propto \prod_{s \in \mathcal{V}} p^*(z_s) \prod_{(s,t) \in \mathcal{E}} \frac{p^*(z_s, z_t)}{p^*(z_s)p^*(z_t)} \tag{9.53}$$

For example, for Figure 9.7a we have

$$p^*(z_1, z_2, z_3, z_4) \propto p^*(z_1)p^*(z_2)p^*(z_3)p^*(z_4)\frac{p^*(z_1, z_2)p^*(z_2, z_3)p^*(z_2, z_4)}{p^*(z_1)p^*(z_2)p^*(z_2)p^*(z_3)p^*(z_2)p^*(z_4)} \tag{9.54}$$

To see the equivalence with the directed representation, we can cancel terms to get

$$p^*(z_1, z_2, z_3, z_4) \propto p^*(z_1, z_2)\frac{p^*(z_2, z_3)}{p^*(z_2)}\frac{p^*(z_2, z_4)}{p^*(z_2)} \tag{9.55}$$

$$= p^*(z_1)p^*(z_2|z_1)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.56}$$

$$= p^*(z_2)p^*(z_1|z_2)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.57}$$

where $p^*(z_t|z_s) = p^*(z_s, z_t)/p^*(z_s)$.

Thus a tree can be represented as either an undirected or directed graph. Both representations can be useful, as we will see.

// Collect to root
for each node s in post-order
    $\text{bel}_s(z_s) \propto \psi_s(z_s) \prod_{t \in \text{ch}_s} m_{t \to s}(z_s)$
    t = parent(s)
    $m_{s \to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t) \text{bel}_s(z_s)$

// Distribute from root
for each node t in pre-order
    s = parent(t)
    $m_{s \to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t) \frac{\text{bel}_s(z_s)}{m_{t \to s}(z_s)}$
    $\text{bel}_t(z_t) \propto \text{bel}_t(z_t) m_{s \to t}(z_t)$

Figure 9.8: Belief propagation on a pairwise, rooted tree.

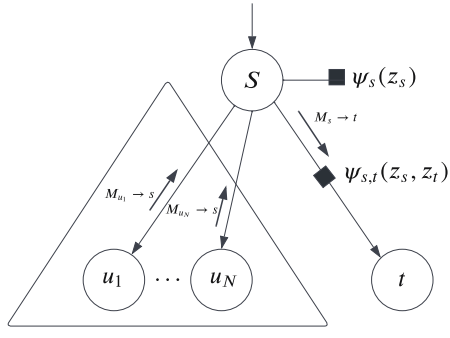


Figure 9.9: Illustration of how the top-down message from s to t is computed during BP on a tree. The u_i nodes are the other children of s, besides t. Square nodes represent clique potentials.

9.3.2 Sum-product algorithm

In this section, we assume that our model is an undirected tree, as in Equation (9.50). However, we will pick an arbitrary node as a root, and orient all the edges downwards away from this root, so that each node has a unique parent. For a directed, rooted tree, we can compute various node orderings. In particular, in a **pre-order**, we traverse from the root to the left subtree and then to right subtree, top to bottom. In a **post-order**, we traverse from the left subtree to the right subtree and then to the root, bottom to top. We will use both of these below.

We now present the **sum-product algorithm** for trees. We first send messages from the leaves to the root. This is the generalization of the forwards pass from Section 9.2.2. Let $m_{s \to t}(z_t)$ denote the message from node s to node t. This summarizes the belief state about z_t given all the evidence in the tree below the $s - t$ edge. Consider a node s in the ordering. We update its belief state by

combining the incoming messages from all its children with its own local evidence:

$$\text{bel}_s(z_s) \propto \psi_s(z_s) \prod_{t \in \text{ch}_s} m_{t \to s}(z_s) \tag{9.58}$$

To compute the outgoing message that s should send to its parent t, we pass the local belief through the pairwise potential linking s and t, and then marginalize out s to get

$$m_{s \to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t) \text{bel}_s(z_s) \tag{9.59}$$

At the root of the tree, $\text{bel}_t(z_t) = p(z_t|\boldsymbol{y})$ will have seen all the evidence. It can then send messages back down to the leaves. The message that s sends to its child t should be the product of all the messages that s received from all its *other* children u, passed through the pairwise potential, and then marginalized:

$$m_{s \to t}(z_t) = \sum_{z_t} \left(\psi_s(z_s) \psi_{st}(z_s, z_t) \prod_{u \in \text{ch}_s \setminus t} m_{u \to s}(z_s) \right) \tag{9.60}$$

See Figure 9.9. Instead of multiplying all-but-one of the messages that s has received, we can multiply all of them and then divide out by the $t \to s$ message from child t. The advantage of this is that the product of all the messages has already been computed in Equation (9.58), so we don't need to recompute that term. Thus we get

$$m_{s \to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t) \frac{\text{bel}_s(z_s)}{m_{t \to s}(z_s)} \tag{9.61}$$

We can think of $\text{bel}_s(z_s)$ as the new updated posterior $p(z_s|\boldsymbol{y})$ given all the evidence, and $m_{t \to s}(z_s)$ as the prior predictive $p(z_s|\boldsymbol{y}_t^-)$, where $\boldsymbol{y}_t^-$ is all the evidence in the subtree rooted at t. Thus the ratio contains the new evidence that t did not already know about from its own subtree. We use this to update the belief state at node t to get:

$$\text{bel}_t(z_t) \propto \text{bel}_t(z_t) m_{s \to t}(z_t) \tag{9.62}$$

(Note that Equation (9.58) is a special case of this where we don't divide out by $m_{s \to t}$, since in the upwards pass, there is no incoming message from the parent.) This is analogous to the backwards smoothing equation in Equation (9.37), with $\alpha_t(i)$ replaced by $\text{bel}_t(z_t = i)$, $A(i, j)$ replaced by $\psi_{st}(z_s = i, z_t = j)$, $\gamma_{t+1}(j)$ replaced by $\text{bel}_s(z_s = j)$, and $\alpha_{t+1|t}(j)$ replaced by $m_{t \to s}(z_s = j)$.

See Figure 9.8 for the overall pseudocode. This can be generalized to directed trees with multiple root nodes (known as **polytrees**) as described in Supplementary Section 9.1.1.

9.3.3 Max-product algorithm

In Section 9.3.2 we described the sum-product algorithm, that computes the posterior marginals:

$$\text{bel}_i(k) = \gamma_i(k) = p(z_i = k|\boldsymbol{y}) = \sum_{\boldsymbol{z}_{-i}} p(z_i = k, \boldsymbol{z}_{-i}|\boldsymbol{y}) \tag{9.63}$$

We can replace the sum operation with the max operation to get **max-product belief propagation**. The result of this computation are a set of **max marginals** for each node:

$$\zeta_i(k) = \max_{\boldsymbol{z}_{-i}} p(z_i = k, \boldsymbol{z}_{-i}|\boldsymbol{y}) \tag{9.64}$$

We can derive two different kinds of "MAP" estimates from these local quantities. The first is $\hat{z}_i = \text{argmax}_k \gamma_i(k)$; this is known as the **maximizer of the posterior marginal** or **MPM** estimate (see e.g., [MMP87; SM12]); let $\hat{\boldsymbol{z}} = [\hat{z}_1, \ldots, \hat{z}_{N_z}]$ be the sequence of such estimates. The second is $\tilde{z}_i = \text{argmax}_k \zeta_i(k)$; we call this the **maximizer of the max marginal** or **MMM** estimate; let $\tilde{\boldsymbol{z}} = [\tilde{z}_1, \ldots, \tilde{z}_{N_z}]$.

An interesting question is: what, if anything, do these estimates have to do with the "true" MAP estimate, $\boldsymbol{z}^* = \text{argmax}_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{y})$? We discuss this below.

9.3.3.1 Connection between MMM and MAP

In [YW04], they showed that, if the max marginals are unique and computed exactly (e.g., if the graph is a tree), then $\tilde{\boldsymbol{z}} = \boldsymbol{z}^*$. This means we can recover the global MAP estimate by running max product BP and then setting each node to its local max (i.e., using the MMM estimate).

However, if there are ties in the max marginals (corresponding to the case where there is more than one globally optimal solution), this "local stitching" process may result in global inconsistencies.

If we have a tree-structured model, we can use a **traceback** procedure, analogous to the Viterbi algorithm (Section 9.2.6), in which we clamp nodes to their optimal values while working backwards from the root. For details, see e.g., [KF09a, p569].

Unfortunately, traceback does not work on general graphs. An alternative, iterative approach, proposed in [YW04], is follows. First we run max product BP, and clamp all nodes which have unique max marginals to their optimal values; we then clamp a single ambiguous node to an optimal value, and condition on all these clamped values as extra evidence, and perform more rounds of message passing, until all ties are broken. This may require many rounds of inference, although the number of non-clamped (hidden) variables gets reduced at each round.

9.3.3.2 Connection between MPM and MAP

In this section, we discuss the MPM estimate, $\hat{\boldsymbol{z}}$, which computes the maximum of the posterior marginals. In general, this does not correspond to the MAP estimate, even if the posterior marginals are exact. To see why, note that MPM just looks at the belief state for each node given all the visible evidence, but ignores any dependencies or constraints that might exist in the prior.

To illustrate why this could be a problem, consider the error correcting code example from Section 5.5, where we defined $p(\boldsymbol{z}, \boldsymbol{y}) = p(z_1)p(z_2)p(z_3|z_1, z_2)\prod_{i=1}^{3} p(y_i|z_i)$, where all variables are binary. The priors $p(z_1)$ and $p(z_2)$ are uniform. The conditional term $p(z_3|z_1, z_2)$ is deterministic, and computes the parity of (z_1, z_2). In particular, we have $p(z_3 = 1|z_1, z_2) = \mathbb{I}(\text{odd}(z_1, z_2))$, so that the total number of 1s in the block $\boldsymbol{z}_{1:3}$ is even. The likelihood terms $p(y_i|z_i)$ represent a bit flipping noisy channel model, with noise level $\alpha = 0.2$.

Suppose we observe $\boldsymbol{y} = (1, 0, 0)$. In this case, the exact posterior marginals are as follows:[2] $\gamma_1 = [0.3469, 0.6531]$, $\gamma_2 = [0.6531, 0.3469]$, $\gamma_3 = [0.6531, 0.3469]$. The exact max marginals are all the same,

2. See error_correcting_code_demo.ipynb for the code.

namely $\zeta_i = [0.3265, 0.3265]$. Finally, the 3 global MAP estimates are $\boldsymbol{z}^* \in \{[0,0,0],[1,1,0],[1,0,1]\}$, each of which corresponds to a single bit flip from the observed vector. The MAP estimates are all valid code words (they have an even number of 1s), and hence are sensible hypotheses about the value of $\boldsymbol{z}$. By contrast, the MPM estimate is $\hat{\boldsymbol{z}} = [1,0,0]$, which is not a legal codeword. (And in this example, the MMM estimate is not well defined, since the max marginals are not unique.)

So, which method is better? This depends on our loss function, as we discuss in Section 34.1. If we want to minimize the prediction error of each z_i, also called **bit error**, we should compute the MPM. If we want to minimize the prediction error for the entire sequence $\boldsymbol{z}$, also called **word error**, we should use MAP, since this can take global constraints into account.

For example, suppose we are performing speech recognition and someones says "recognize speech". MPM decoding may return "wreck a nice beach", since locally it may be that "beach" is the most probable interpretation of "speech" when viewed in isolation (see Figure 34.3). However, MAP decoding would infer that "recognize speech" is the more likely overall interpretation, by taking into account the language model prior, $p(\boldsymbol{z})$.

On the other hand, if we don't have strong constraints, the MPM estimate can be more robust [MMP87; SM12], since it marginalizes out the other nodes, rather than maxing them out. For example, in the casino HMM example in Figure 9.3, we see that the MPM method makes 49 bit errors (out of a total possible of $T = 300$), and the MAP path makes 60 errors.

9.3.3.3 Connection between MPE and MAP

In the graphical models literature, computing the jointly most likely setting of all the latent variables, $\boldsymbol{z}^* = \text{argmax}_{\boldsymbol{z}} \, p(\boldsymbol{z}|\boldsymbol{y})$, is known as the **most probable explanation** or **MPE** [Pea88]. In that literature, the term "MAP" is used to refer to the case where we maximize some of the hidden variables, and marginalize (sum out) the rest. For example, if we maximize a single node, z_i, but sum out all the others, $\boldsymbol{z}_{-i}$, we get the MPM $\hat{z}_i = \text{argmax}_{z_i} \sum_{\boldsymbol{z}_{-i}} p(\boldsymbol{z}|\boldsymbol{y})$.

We can generalize the MPM estimate to compute the best guess for a set of query variables Q, given evidence on a set of visible variables V, marginalizing out the remaining variables R, to get

$$\boldsymbol{z}_Q^* = \arg\max_{\boldsymbol{z}_Q} \sum_{\boldsymbol{z}_R} p(\boldsymbol{z}_Q, \boldsymbol{z}_R | \boldsymbol{z}_V) \tag{9.65}$$

(Here $\boldsymbol{z}_R$ are called **nuisance variables**, since they are not of interest, and are not observed.) In [Pea88], this is called a MAP estimate, but we will call it an MPM estimate, to avoid confusion with the ML usage of the term "MAP" (where we maximize everything jointly).

9.4 Loopy belief propagation

In this section, we extend belief propagation to work on graphs with cycles or loops; this is called **loopy belief propagation** or **LBP**. Unfortunately, this method may not converge, and even if it does, it is not clear if the resulting estimates are valid. Indeed, Judea Pearl, who invented belief propagation for trees, wrote the following about loopy BP in 1988:

> When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly connected,

> messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the network. — [Pea88, p.195]

Despite these reservations, Pearl advocated the use of belief propagation in loopy networks as an approximation scheme (J. Pearl, personal communication). [MWJ99] found empirically that it works on various graphical models, and it is now used in many real world applications, some of which we discuss below. In addition, there is now some theory justifying its use in certain cases, as we discuss below. (For more details, see e.g., [Yed11].)

9.4.1 Loopy BP for pairwise undirected graphs

In this section, we assume (for notational simplicity) that our model is an undirected pairwise PGM, as in Equation (9.50). However, unlike Section 9.3.2, we do not assume the graph is a tree. We can apply the same message passing equations as before. However, since there is no natural node ordering, we will do this in a parallel, asynchronous way. The basic idea is that all nodes receive messages from their neighbors in parallel, they then update their belief states, and finally they send new messages back out to their neighbors. This message passing process repeats until convergence. This kind of computing architecture is called a **systolic array**, due to its resemblance to a beating heart.

More precisely, we initialize all messages to the all 1's vector. Then, in parallel, each node absorbs messages from all its neighbors using

$$\text{bel}_s(z_s) \propto \psi_s(z_s) \prod_{t \in \text{nbr}_s} m_{t \to s}(z_s) \tag{9.66}$$

Then, in parallel, each node sends messages to each of its neighbors:

$$m_{s \to t}(z_t) = \sum_{z_s} \left(\psi_s(z_s) \psi_{st}(z_s, z_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \to s}(z_s) \right) \tag{9.67}$$

The $m_{s \to t}$ message is computed by multiplying together all incoming messages, except the one sent by the recipient, and then passing through the ψ_{st} potential. We continue this process until convergence. If the graph is a tree, the method is guaranteed to converge after $D(G)$ iterations, where $D(G)$ is the **diameter** of the graph, that is, the largest distance between any two nodes.

9.4.2 Loopy BP for factor graphs

To implement loopy BP for general graphs, including those with higher-order clique potentials (beyond pairwise), it is useful to use a factor graph representation described in Section 4.6.1. In this section, we summarize the BP equations for the bipartite version of factor graphs, as derived in [KFL01].[3] For a version that works for Forney factor graphs, see [Loe+07].

3. For an efficient JAX implementation of these equations for discrete factor graphs, see https://github.com/deepmind/PGMax. For the Gaussian case, see https://github.com/probml/pgm-jax.

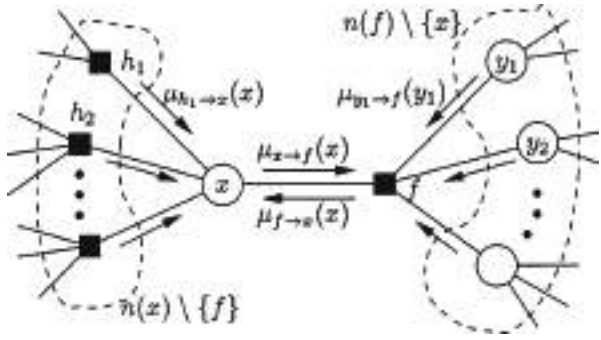


Figure 9.10: Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent variables. The y_i nodes correspond to the neighbors x'_i of f other than x. From Figure 6 of [KFL01]. Used with kind permission of Brendan Frey.

In the case of bipartite factor graphs, we have two kinds of messages: variables to factors

$$m_{x \to f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \to x}(x) \tag{9.68}$$

and factors to variables

$$m_{f \to x}(x) = \sum_{\boldsymbol{x}'} f(x, \boldsymbol{x}') \prod_{x' \in \text{nbr}(f) \setminus \{x\}} m_{x' \to f}(x') \tag{9.69}$$

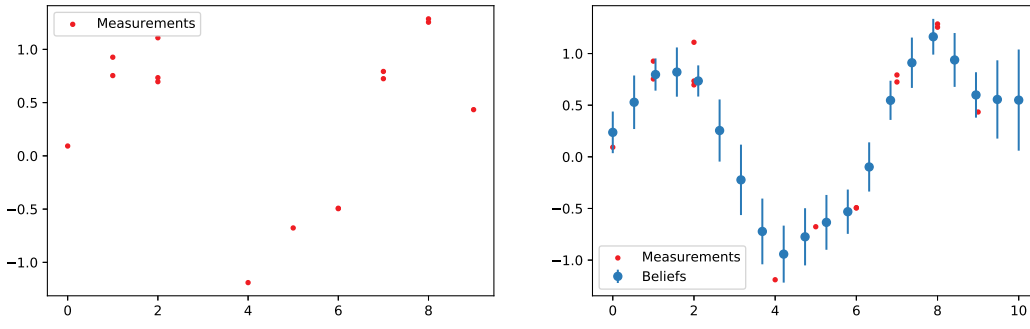
Here $\text{nbr}(x)$ are all the factors that are connected to variable x, and $\text{nbr}(f)$ are all the variables that are connected to factor f. These messages are illustrated in Figure 9.10. At convergence, we can compute the final beliefs as a product of incoming messages:

$$\text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \to x}(x) \tag{9.70}$$

The order in which the messages are sent can be determined using various heuristics, such as computing a spanning tree, and picking an arbitrary node as root. Alternatively, the update ordering can be chosen adaptively using **residual belief propagation** [EMK06]. Or fully parallel, asynchronous implementations can be used.

9.4.3 Gaussian belief propagation

It is possible to genereralize (loopy) belief propagation to the Gaussian case, by using the "calculus for linear Gaussian models" in Section 2.3.3 to compute the messages and beliefs. Note that computing the posterior mean in a linear-Gaussian system is equivalent to solving a linear system, so these methods are also useful for linear algebra. See e.g., [PL03; Bic09; Du+18] for details.



(a) (b)

Figure 9.11: Interpolating noisy data using Gaussian belief propagation applied to a 1d MRF. Generated by gauss-bp-1d-line.ipynb.

As an example of Gaussian BP, consider the problem of interpolating noisy data in 1d, as discussed in [OED21]. In particular, let $f : \mathbb{R} \to \mathbb{R}$ be an unknown function for which we get N noisy measurements y_i at locations x_i. We want to estimate $z_i = f(g_i)$ at G grid locations g_i. Let x_i be the closest location to g_i. Then we assume the measurement factor is as follows:

$$\psi_i(z_{i-1}, z_i) = \frac{1}{\sigma^2}(\hat{y}_i - y_i)^2 \tag{9.71}$$

$$\hat{y}_i = (1 - \gamma_i)z_{i-1} + \gamma_i z_i \tag{9.72}$$

$$\gamma_i = \frac{x_i - g_i}{g_i - g_{i-1}} \tag{9.73}$$

Here $\hat{y}_i$ is the predicted measurement. The potential function makes the unknown function values z_{i-1} and z_i move closer to the observation, based on how far these grid points are from where the measurement was taken. In addition, we add a pairwise smoothness potential, that encodes the prior that z_i should be close to z_{i-1} and z_{i+1}:

$$\phi_i(z_{i-1}, z_i) = \frac{1}{\tau^2}\delta_i^2 \tag{9.74}$$

$$\delta_i = z_i - z_{i-1} \tag{9.75}$$

The overall model is

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g}, \sigma^2, \tau^2) \propto \prod_{i=1}^{G} \psi_i(z_{i-1}, z_i)\phi_i(z_{i-1}, z_i) \tag{9.76}$$

Suppose the true underlying function is a sine wave. We show some sample data in Figure 9.11(a). We then apply Gaussian BP. Since this model is a chain, and the model is linear-Gaussian, the resulting posterior marginals, shown in Figure 9.11(b), are exact. We see that the method has inferred the underlying sine shape just based on a smoothness prior.

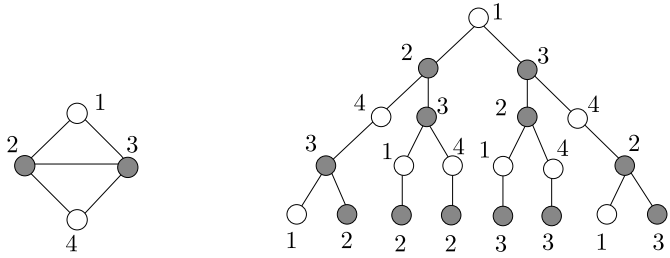


Figure 9.12: (a) A simple loopy graph. (b) The computation tree, rooted at node 1, after 4 rounds of message passing. Nodes 2 and 3 occur more often in the tree because they have higher degree than nodes 1 and 2. From Figure 8.2 of [WJ08]. Used with kind permission of Martin Wainwright.

To perform message passing in models with non-linear (but Gaussian) potentials, we can generalize the extended Kalman filter techniques from Section 8.3.2 and the moment matching techniques (based on quadrature/sigma points) from Section 8.5.1 and Section 8.5.1.1 from chains to general factor graphs (see e.g., [MHH14; PHR18; HPR19]). To extend to the non-Gaussian case, we can use **non-parametric BP** or **particle BP** (see e.g., [Sud+03; Isa03; Sud+10; Pac+14]), which uses ideas from particle filtering (Section 13.2).

9.4.4 Convergence

Loopy BP may not converge, or may only converge slowly. In this section, we discuss some techniques that increase the chances of convergence, and the speed of convergence.

9.4.4.1 When will LBP converge?

The details of the analysis of when LBP will converge are beyond the scope of this chapter, but we briefly sketch the basic idea. The key analysis tool is the **computation tree**, which visualizes the messages that are passed as the algorithm proceeds. Figure 9.12 gives a simple example. In the first iteration, node 1 receives messages from nodes 2 and 3. In the second iteration, it receives one message from node 3 (via node 2), one from node 2 (via node 3), and two messages from node 4 (via nodes 2 and 3). And so on.

The key insight is that T iterations of LBP is equivalent to exact computation in a computation tree of height $T + 1$. If the strengths of the connections on the edges is sufficiently weak, then the influence of the leaves on the root will diminish over time, and convergence will occur. See [MK05; WJ08] and references therein for more information.

9.4.4.2 Making LBP converge

Although the theoretical convergence analysis is very interesting, in practice, when faced with a model where LBP is not converging, what should we do?

One simple way to increase the chance of convergence is to use **damping**. That is, at iteration k,

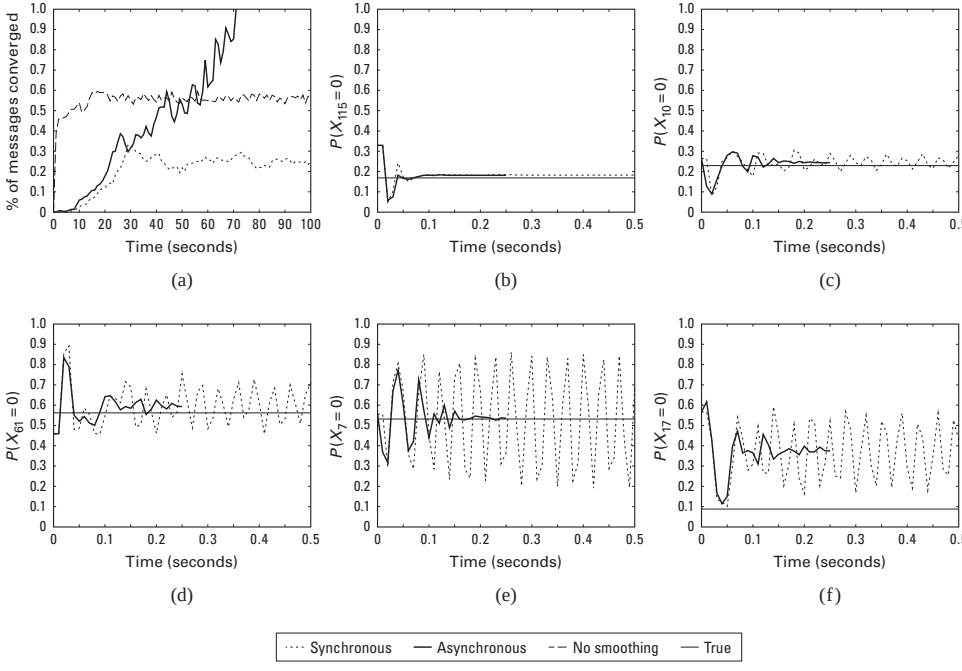


Figure 9.13: Illustration of the behavior of loopy belief propagation on an 11×11 *Ising grid with random potentials,* $w_{ij} \sim \text{Unif}(-C, C)$*, where* $C = 11$*. For larger* C*, inference becomes harder. (a) Percentage of messages that have converged vs time for 3 different update schedules: Dotted = damped synchronous (few nodes converge), dashed = undamped asychnronous (half the nodes converge), solid = damped asychnronous (all nodes converge). (b-f) Marginal beliefs of certain nodes vs time. Solid straight line = truth, dashed = sychronous, solid = damped asychronous. From Figure 11.C.1 of [KF09a]. Used with kind permission of Daphne Koller.*

we use an update of the form

$$m_{t\to s}^k(x_s) = \lambda m_{t\to s}(x_s) + (1-\lambda) m_{t\to s}^{k-1}(x_s) \tag{9.77}$$

where $m_{t\to s}(x_s)$ is the standard undamped message, where $0 \le \lambda \le 1$ is the damping factor. Clearly if $\lambda = 1$ this reduces to the standard scheme, but for $\lambda < 1$, this partial updating scheme can help improve convergence. Using a value such as $\lambda \sim 0.5$ is standard practice. The benefits of this approach are shown in Figure 9.13, where we see that damped updating results in convergence much more often than undamped updating (see [ZLG20] for some analysis of the benefits of damping).

It is possible to devise methods, known as **double loop algorithms**, which are guaranteed to converge to a local minimum of the same objective that LBP is minimizing [Yui01; WT01]. Unfortunately, these methods are rather slow and complicated, and the accuracy of the resulting marginals is usually not much greater than with standard LBP. (Indeed, oscillating marginals is sometimes a sign that the LBP approximation itself is a poor one.) Consequently, these techniques are not very widely used (although see [GF21] for a newer technique).

9.4.4.3 Increasing the convergence rate with adaptive scheduling

The standard approach when implementing LBP is to perform **synchronous updates**, where all nodes absorb messages in parallel, and then send out messages in parallel. That is, the new messages at iteration $k+1$ are computed in parallel using

$$\boldsymbol{m}_{1:E}^{k+1} = (f_1(\boldsymbol{m}^k), \ldots, f_E(\boldsymbol{m}^k)) \tag{9.78}$$

where E is the number of edges, and $f_i(\boldsymbol{m})$ is the function that computes the message for edge i given all the old messages. This is analogous to the Jacobi method for solving linear systems of equations.

It is well known [Ber97b] that the Gauss-Seidel method, which performs **asynchronous updates** in a fixed round-robin fashion, converges faster when solving linear systems of equations. We can apply the same idea to LBP, using updates of the form

$$\boldsymbol{m}_i^{k+1} = f_i\left(\{\boldsymbol{m}_j^{k+1} : j < i\}, \{\boldsymbol{m}_j^k : j > i\}\right) \tag{9.79}$$

where the message for edge i is computed using new messages (iteration $k+1$) from edges earlier in the ordering, and using old messages (iteration k) from edges later in the ordering.

This raises the question of what order to update the messages in. One simple idea is to use a fixed or random order. The benefits of this approach are shown in Figure 9.13, where we see that (damped) asynchronous updating results in convergence much more often than synchronous updating.

However, we can do even better by using an adaptive ordering. The intuition is that we should focus our computational efforts on those variables that are most uncertain. [EMK06] proposed a technique known as **residual belief propagation**, in which messages are scheduled to be sent according to the norm of the difference from their previous value. That is, we define the residual of new message $m_{s\to t}$ at iteration k to be

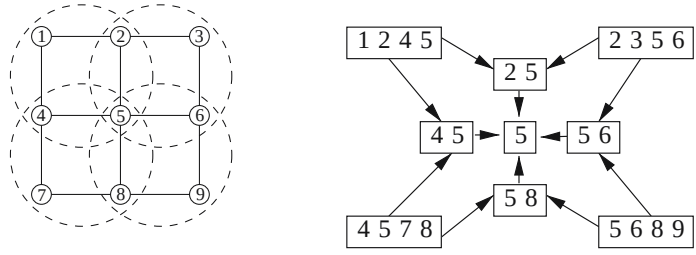
$$r(s,t,k) = ||\log m_{s\to t} - \log m_{s\to t}^k||_\infty = \max_j |\log \frac{m_{s\to t}(j)}{m_{s\to t}^k(j)}| \tag{9.80}$$

We can store messages in a priority queue, and always send the one with highest residual. When a message is sent from s to t, all of the other messages that depend on $m_{s\to t}$ (i.e., messages of the form $m_{t\to u}$ where $u \in \text{nbr}(t) \setminus s$) need to be recomputed; their residual is recomputed, and they are added back to the queue. In [EMK06], they showed (experimentally) that this method converges more often, and much faster, than using sychronous updating, or asynchronous updating with a fixed order.

A refinement of residual BP was presented in [SM07]. In this paper, they use an upper bound on the residual of a message instead of the actual residual. This means that messages are only computed if they are going to be sent; they are not just computed for the purposes of evaluating the residual. This was observed to be about five times faster than residual BP, although the quality of the final results are similar.

9.4.5 Accuracy

For a graph with a single loop, one can show that the max-product version of LBP will find the correct MAP estimate, if it converges [Wei00]. For more general graphs, one can bound the error in the approximate marginals computed by LBP, as shown in [WJW03; IFW05; Vin+10b].



(a) (b)

Figure 9.14: (a) Clusters superimposed on a 3×3 lattice graph. (b) Corresponding hyper-graph. Nodes represent clusters, and edges represent set containment. From Figure 4.5 of [WJ08]. Used with kind permission of Martin Wainwright.

Much stronger results are available in the case of Gaussian models. In particular, it can be shown that, if the method converges, the means are exact, although the variances are not (typically the beliefs are over confident). See e.g., [WF01a; JMW06; Bic09; Du+18] for details.

9.4.6 Generalized belief propagation

We can improve the accuracy of loopy BP by clustering together nodes that form a tight loop. This is known as the **cluster variational method**, or **generalized belief propagation** [YFW00].

The result of clustering is a hyper-graph, which is a graph where there are hyper-edges between sets of vertices instead of between single vertices. Note that a junction tree (Section 9.6) is a kind of hyper-graph. We can represent a hyper-graph using a poset (partially ordered set) diagram, where each node represents a hyper-edge, and there is an arrow $e_1 \to e_2$ if $e_2 \subset e_1$. See Figure 9.14 for an example.

If we allow the size of the largest hyper-edge in the hyper-graph to be as large as the treewidth of the graph, then we can represent the hyper-graph as a tree, and the method will be exact, just as LBP is exact on regular trees (with treewidth 1). In this way, we can define a continuum of approximations, from LBP all the way to exact inference. See Supplementary Section 10.4.3.3 for more information.

9.4.7 Convex BP

In Supplementary Section 10.4.3 we analyze LBP from a variational perspective, and show that the resulting optimization problem, for both standard and generalized BP, is non-convex. However it is possible to create a version of **convex BP**, as we explain in Supplementary Section 10.4.4, which has the advantage that it will always converge.

9.4.8 Application: error correcting codes

LBP was first proposed by Judea Pearl in his 1988 book [Pea88]. He recognized that applying BP to loopy graphs might not work, but recommended it as a heuristic.

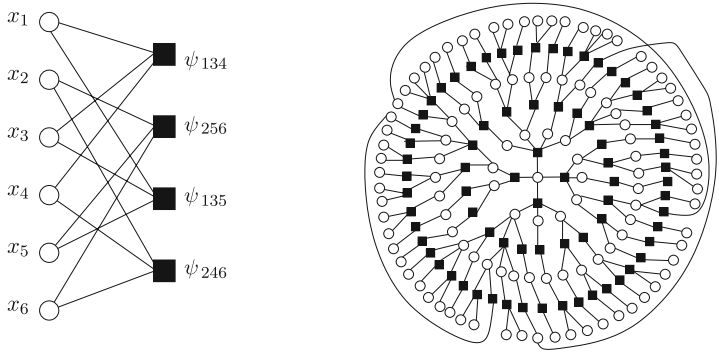


Figure 9.15: (a) A simple factor graph representation of a (2,3) low-density parity check code. Each message bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form $\psi_{stu}(x_s, x_t, x_u) = \mathbb{I}(x_s \otimes x_t \otimes x_u = 1)$, where $\otimes$ is the xor operator. The local evidence factors for each hidden node are not shown. (b) A larger example of a random LDPC code. We see that this graph is "locally tree-like", meaning there are no short cycles; rather, each cycle has length $\sim \log m$, where m is the number of nodes. This gives us a hint as to why loopy BP works so well on such graphs. (Note, however, that some error correcting code graphs have short loops, so this is not the full explanation.) From Figure 2.9 from [WJ08]. Used with kind permission of Martin Wainwright.

However, the main impetus behind the interest in LBP arose when McEliece, MacKay, and Cheng [MMC98] showed that a popular algorithm for error correcting codes, known as **turbocodes** [BGT93], could be viewed as an instance of LBP applied to a certain kind of graph.

We introduced error correcting codes in Section 5.5. Recall that the basic idea is to send the source message $\boldsymbol{x} \in \{0,1\}^m$ over a noisy channel, and for the receiver to try to infer it given noisy measurements $\boldsymbol{y} \in \{0,1\}^m$ or $\boldsymbol{y} \in \mathbb{R}^m$. That is, the receiver needs to compute $\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}}\, p(\boldsymbol{x}|\boldsymbol{y}) = \text{argmax}_{\boldsymbol{x}}\, \tilde{p}(\boldsymbol{x})$.

It is standard to represent $\tilde{p}(\boldsymbol{x})$ as a factor graph (Section 4.6.1), which can easily represent any deterministic relationships (parity constraints) between the bits. A factor graph is a bipartite graph with x_i nodes on one side, and factors on the other. A graph in which each node is connected to n factors, and in which each factor is connected to k nodes, is called an (n, k) code. Figure 9.15(a) shows a simple example of a $(2, 3)$ code, where each bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form

$$\psi_{stu}(x_s, x_t, x_u) \triangleq \begin{cases} 1 & \text{if } x_s \otimes x_t \otimes x_u = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.81}$$

If the degrees of the parity checks and variable nodes remain bounded as the blocklength m increases, this is called a **low-density parity check code**, or **LDPC code**. (Turbocodes are constructed in a similar way.)

Figure 9.15(b) shows an example of a randomly constructed LDPC code. This graph is "locally tree-like", meaning there are no short cycles; rather, each cycle has length $\sim \log m$. This fact is important to the success of LBP, which is only guaranteed to work on tree-structured graphs. Using

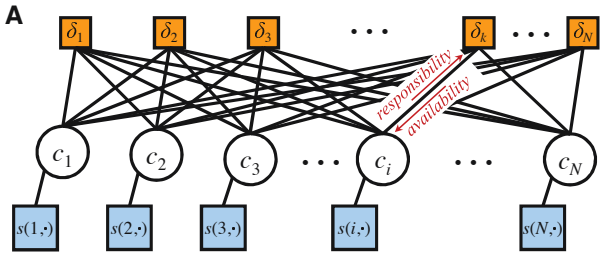


Figure 9.16: Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of [FD07a]. Used with kind permission of Brendan Frey.

methods such as these, people have been able to approach the lower bound in Shannon's channel coding theorem, meaning they have produced codes with very little redundancy for a given amount of noise in the channel. See e.g., [MMC98; Mac03] for more details. Such codes are widely used, e.g., in modern cellphones.

9.4.9 Application: affinity propagation

In this section, we discuss **affinity propagation** [FD07a], which can be seen as an improvement to K-medoids clustering, which takes as input a pairwise similarity matrix. The idea is that each datapoint must choose another datapoint as its exemplar or centroid; some datapoints will choose themselves as centroids, and this will automatically determine the number of clusters. More precisely, let $c_i \in \{1, \ldots, N\}$ represent the centroid for datapoint i. The goal is to maximize the following function

$$J(\boldsymbol{c}) = \sum_{i=1}^{N} S(i, c_i) + \sum_{k=1}^{N} \delta_k(\boldsymbol{c}) \tag{9.82}$$

where $S(i, c_i)$ is the similarity between datapoint i and its centroid c_i. The second term is a penalty term that is $-\infty$ if some datapoint i has chosen k as its exemplar (i.e., $c_i = k$), but k has not chosen itself as an exemplar (i.e., we do not have $c_k = k$). More formally,

$$\delta_k(\boldsymbol{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 & \text{otherwise} \end{cases} \tag{9.83}$$

This encourages "representative" samples to vote for themselves as centroids, thus encouraging clustering behavior.

The objective function can be represented as a factor graph. We can either use N nodes, each with N possible values, as shown in Figure 9.16, or we can use N^2 binary nodes (see [GF09] for the details). We will assume the former representation.

We can find a strong local maximum of the objective by using max-product loopy belief propagation (Section 9.4). Referring to the model in Figure 9.16, each variable node c_i sends a message to each factor node δ_k. It turns out that this vector of N numbers can be reduced to a scalar message,

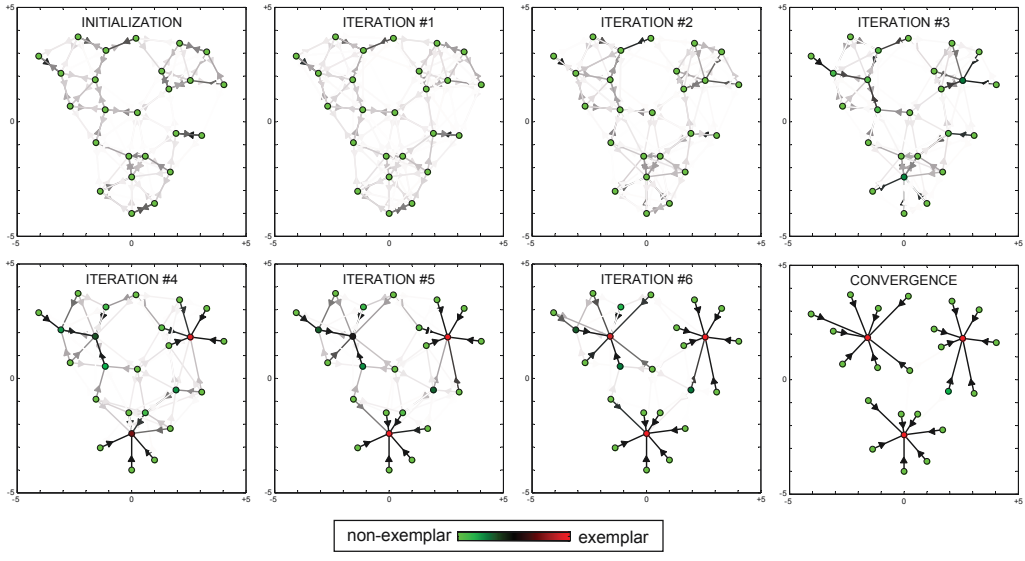


Figure 9.17: Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \to k$ arrow reflects how much point i wants to belong to exemplar k. From Figure 1 of [FD07a]. Used with kind permission of Brendan Frey.

denoted $r_{i \to k}$, known as the responsibility. This is a measure of how much i thinks k would make a good exemplar, compared to all the other exemplars i has looked at. In addition, each factor node δ_k sends a message to each variable node c_i. Again this can be reduced to a scalar message, $a_{i \leftarrow k}$, known as the availability. This is a measure of how strongly k believes it should an exemplar for i, based on all the other datapoints k has looked at.

As usual with loopy BP, the method might oscillate, and convergence is not guaranteed. However, by using damping, the method is very reliable in practice. If the graph is densely connected, message passing takes $O(N^2)$ time, but with sparse similarity matrices, it only takes $O(E)$ time, where E is the number of edges or non-zero entries in S.

The number of clusters can be controlled by scaling the diagonal terms $S(i,i)$, which reflect how much each datapoint wants to be an exemplar. Figure 9.17 gives a simple example of some 2d data, where the negative Euclidean distance was used to measured similarity. The $S(i,i)$ values were set to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are reported in [FD07a], who show that the method significantly outperforms K-medoids.

9.4.10 Emulating BP with graph neural nets

There is a close connection between message passing in PGMs and message passing in graph neural networks (GNNs), which we discuss in Section 16.3.6. However, for PGMs, the message computations are computing using (non-learned) update equations that work for any model; all that is needed

is the graph structure G, model parameters $\boldsymbol{\theta}$, and evidence $\boldsymbol{v}$. By contrast, GNNs are trained to emulate specific functions using labeled input-output pairs.

It is natural to wonder what happens if we train a GNN on the exact posterior marginals derived from a small PGM, and then apply that trained GNN to a different test PGM. In [Yoo+18; Zha+19d], they show this method can work quite well if the test PGM is similar in structure to the one used for training.

An alternative approach is to start with a known PGM, and then "unroll" the BP message passing algorithm to produce a layered feedforward model, whose connectivity is derived from the graph. The resulting network can then be trained discriminatively for some end-task (not necessarily computing posterior marginals). Thus the BP procedure applied to the PGM just provides a way to design the neural network structure. This method is called **deep unfolding** (see e.g., [HLRW14]), and can often give very good results. (See also [SW20] for a more recent version of this approach, called "**neural enhanced BP**".)

These neural methods are useful if the PGM is fixed, and we want to repeatedly perform inference or prediction with it, using different values of the evidence, but where the set of nodes which are observed is always the same. This is an example of amortized inference, where we train a model to emulate the results of running an iterative optimization scheme (see Section 10.1.5 for more discussion).

9.5 The variable elimination (VE) algorithm

In this section, we discuss an algorithm to compute a posterior marginal $p(\boldsymbol{z}_Q|\boldsymbol{y})$ for any query set Q, assuming p is defined by a graphical model. Unlike loopy BP, it is guaranteed to give the correct answers even if the graph has cycles. We assume all the hidden nodes are discrete, although a version of the algorithm can be created for the Gaussian case by using the rules for sum and product defined in Section 2.3.3.

9.5.1 Derivation of the algorithm

We will explain the algorithm by applying it to an example. Specifically, we consider the student network from Section 4.2.2.2. Suppose we want to compute $p(J = 1)$, the marginal probability that a person will get a job. Since we have 8 binary variables, we could simply enumerate over all possible assignments to all the variables (except for J), adding up the probability of each joint instantiation:

$$p(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C p(C, D, I, G, S, L, J, H) \tag{9.84}$$

However, this would take $O(2^7)$ time. We can be smarter by **pushing sums inside products**. This is the key idea behind the **variable elimination** algorithm [ZP96], also called **bucket elimination** [Dec96], or, in the context of genetic pedigree trees, the **peeling algorithm** [CTS78].

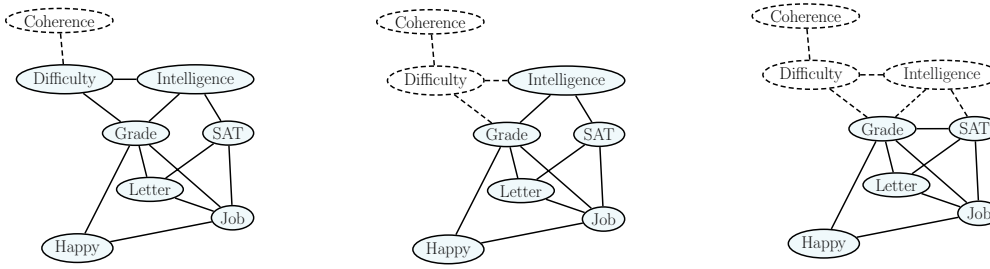


Figure 9.18: Example of the elimination process, in the order C, D, I, H, G, S, L. When we eliminate I (figure c), we add a fill-in edge between G and S, since they are not connected. Adapted from Figure 9.10 of [KF09a].

In our example, we get

$$\begin{aligned} p(J) &= \sum_{L,S,G,H,I,D,C} p(C,D,I,G,S,L,J,H) \\ &= \sum_{L,S,G,H,I,D,C} \psi_C(C)\psi_D(D,C)\psi_I(I)\psi_G(G,I,D)\psi_S(S,I)\psi_L(L,G) \\ &\times \psi_J(J,L,S)\psi_H(H,G,J) \\ &= \sum_{L,S} \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \\ &\times \sum_D \psi_G(G,I,D) \sum_C \psi_C(C)\psi_D(D,C) \end{aligned}$$

We now evaluate this expression, working right to left as shown in Table 9.1. First we multiply together all the terms in the scope of the $\sum_C$ operator to create the temporary factor

$$\tau_1'(C,D) = \psi_C(C)\psi_D(D,C) \tag{9.85}$$

Then we marginalize out C to get the new factor

$$\tau_1(D) = \sum_C \tau_1'(C,D) \tag{9.86}$$

Next we multiply together all the terms in the scope of the $\sum_D$ operator and then marginalize out to create

$$\tau_2'(G,I,D) = \psi_G(G,I,D)\tau_1(D) \tag{9.87}$$

$$\tau_2(G,I) = \sum_D \tau_2'(G,I,D) \tag{9.88}$$

And so on.

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_G(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_G(G,I,D)\tau_1(D)}_{\tau_2(G,I)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \underbrace{\sum_I \psi_S(S,I)\psi_I(I)\tau_2(G,I)}_{\tau_3(G,S)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \underbrace{\sum_H \psi_H(H,G,J)}_{\tau_4(G,J)} \tau_3(G,S)$$

$$\sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_G \psi_L(L,G)\tau_4(G,J)\tau_3(G,S)}_{\tau_5(J,L,S)}$$

$$\sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_5(J,L,S)}_{\tau_6(J,L)}$$

$$\underbrace{\sum_L \tau_6(J,L)}_{\tau_7(J)}$$

Table 9.1: Eliminating variables from Figure 4.38 in the order C, D, I, H, G, S, L to compute $P(J)$.

The above technique can be used to compute any marginal of interest, such as $p(J)$ or $p(J, H)$. To compute a conditional, we can take a ratio of two marginals, where the visible variables have been clamped to their known values (and hence don't need to be summed over). For example,

$$p(J = j|I = 1, H = 0) = \frac{p(J = j, I = 1, H = 0)}{\sum_{j'} p(J = j', I = 1, H = 0)} \tag{9.89}$$

9.5.2 Computational complexity of VE

The running time of VE is clearly exponential in the size of the largest factor, since we have to sum over all of the corresponding variables. Some of the factors come from the original model (and are thus unavoidable), but new factors may also be created in the process of summing out. For example, in Table 9.1, we created a factor involving G, I, and S; but these nodes were not originally present together in any factor.

The order in which we perform the summation is known as the **elimination order**. This can have a large impact on the size of the intermediate factors that are created. For example, consider the ordering in Table 9.1: the largest created factor (beyond the original ones in the model) has size 3, corresponding to $\tau_5(J, L, S)$. Now consider the ordering in Table 9.2: now the largest factors are $\tau_1(I, D, L, J, H)$ and $\tau_2(D, L, S, J, H)$, which are much bigger.

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \sum_I \psi_I(I)\psi_S(S,I) \underbrace{\sum_G \psi_G(G,I,D)\psi_L(L,G)\psi_H(H,G,J)}_{\tau_1(I,D,L,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_I \psi_I(I)\psi_S(S,I)\tau_1(I,D,L,J,H)}_{\tau_2(D,L,S,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_2(D,L,S,J,H)}_{\tau_3(D,L,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \underbrace{\sum_L \tau_3(D,L,J,H)}_{\tau_4(D,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \underbrace{\sum_H \tau_4(D,J,H)}_{\tau_5(D,J)}$$

$$\sum_D \underbrace{\sum_C \psi_D(D,C)\tau_5(D,J)}_{\tau_6(D,J)}$$

$$\underbrace{\sum_D \tau_6(D,J)}_{\tau_7(J)}$$

Table 9.2: Eliminating variables from Figure 4.38 in the order G, I, S, L, H, C, D.

We can determine the size of the largest factor graphically, without worrying about the actual numerical values of the factors, by running the VE algorithm "symbolically". When we eliminate a variable z_t, we connect together all variables that share a factor with z_t (to reflect the new temporary factor τ'_t). The edges created by this process are called **fill-in edges**. For example, Figure 9.18 shows the fill-in edges introduced when we eliminate in the $C, D, I, \ldots$ order. The first two steps do not introduce any fill-ins, but when we eliminate I, we connect G and S, to capture the temporary factor

$$\tau'_3(G,S,I) = \psi_S(S,I)\psi_I(I)\tau_2(G,I) \tag{9.90}$$

Let $\mathcal{G}_\prec$ be the (undirected) graph induced by applying variable elimination to $\mathcal{G}$ using elimination ordering $\prec$. The temporary factors generated by VE correspond to maximal **cliques** in the graph $\mathcal{G}_\prec$. For example, with ordering (C, D, I, H, G, S, L), the maximal cliques are as follows:

$$\{C,D\}, \{D,I,G\}, \{G,L,S,J\}, \{G,J,H\}, \{G,I,S\} \tag{9.91}$$

It is clear that the time complexity of VE is

$$\sum_{c \in \mathcal{C}(G_\prec)} K^{|c|} \tag{9.92}$$

where $\mathcal{C}(\mathcal{G})$ are the (maximal) cliques in graph $\mathcal{G}$, $|c|$ is the size of the clique c, and we assume for notational simplicity that all the variables have K states each.

Let us define the **induced width** of a graph given elimination ordering $\prec$, denoted $w_{\prec}$, as the size of the largest factor (i.e., the largest clique in the induced graph) minus 1. Then it is easy to see that the complexity of VE with ordering $\prec$ is $O(K^{w_{\prec}+1})$. The smallest possible induced width for a graph is known at its **treewidth**. Unfortunately finding the corresponding optimal elimination order is an NP-complete problem [Yan81; ACP87]. See Section 9.5.3 for a discussion of some approximate methods for finding good elimination orders.

9.5.3 Picking a good elimination order

Many algorithms take time (or space) which is exponential in the tree width of the corresponding graph. For example, this applies to Cholesky decompositions of sparse matrices, as well as to einsum contractions (see https://github.com/dgasmith/opt_einsum). Hence we would like to find an elimination ordering that minimizes the width. We say that an ordering π is a **perfect elimination ordering** if it does not introduce any fill-in edges. Every graph that is already triangulated (e.g., a tree) has a perfect elimination ordering. We call such graphs **decomposable**.

In general, we will need to add fill-in edges to ensure the resulting graph is decomposable. Different orderings can introduce different numbers of fill-in edges, which affects the width of the resulting chordal graph; for example, compare Table 9.1 to Table 9.2.

Choosing an elimination ordering with minimal width is NP-complete [Yan81; ACP87]. It is common to use greedy approximation known as the **min-fill heuristic**, which works as follows: eliminate any node which would not result in any fill-ins (i.e., all of whose uneliminated neighbors already form a clique); if there is no such node, eliminate the node which would result in the minimum number of fill-in edges. When nodes have different weights (e.g., representing different numbers of states), we can use the **min-weight heuristic**, where we try to minimize the weight of the created cliques at each step.

Of course, many other methods are possible. See [Heg06] for a general survey. [Kja90; Kja92] compared simulated annealing with the above greedy method, and found that it sometimes works better (although it is much slower). [MJ97] approximate the discrete optimization problem by a continuous optimization problem. [BG96] present a randomized approximation algorithm. [Gil88] present the nested dissection order, which is always within $O(\log N)$ of optimal. [Ami01] discuss various constant-factor appoximation algorithms. [Dav+04] present the **AMD** or approximate minimum degree ordering algorithm, which is implemented in Matlab.[4] The METIS library can be used for finding elimination orderings for large graphs; this implements the **nested dissection** algorithm [GT86]. For a planar graph with N nodes, the resulting treewidth will have the optimal size of $O(N^{3/2})$.

9.5.4 Computational complexity of exact inference

We have seen that variable elimination takes $O(NK^{w+1})$ time to compute the marginals for a graph with N nodes, and treewidth w, where each variable has K states. If the graph is densely connected, then $w = O(N)$, and so inference will take time exponential in N.

4. See the description of the symamd command at https://bit.ly/31N6E2b. ("sym" stands for symbolic, "amd" stands approximate minimum degree.)

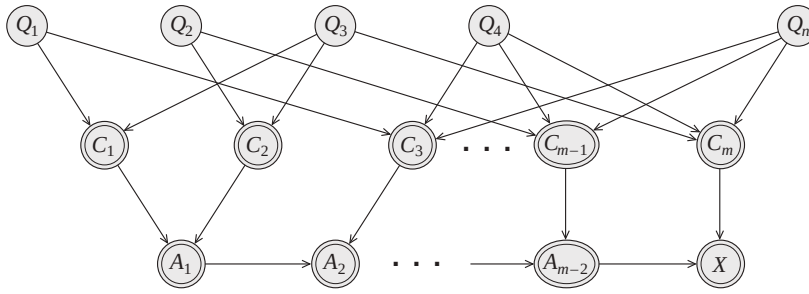


Figure 9.19: Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final x node has bounded size. The double rings denote nodes with deterministic CPDs. From Figure 9.1 of [KF09a]. Used with kind permission of Daphne Koller.

Of course, just because some particular algorithm is slow doesn't mean that there isn't some smarter algorithm out there. Unfortunately, this seems unlikely, since it is easy to show that exact inference for discrete graphical models is NP-hard [DL93]. The proof is a simple reduction from the satisfiability problem. In particular, note that we can encode any 3-SAT problem as a DPGM with deterministic links, as shown in Figure 9.19. We clamp the final node, x, to be on, and we arrange the CPTs so that $p(x = 1) > 0$ iff there is a satisfying assignment. Computing any posterior marginal requires evaluating the normalization constant, $p(x = 1)$, so inference in this model implicitly solves the SAT problem.

In fact, exact inference is #P-hard [Rot96], which is even harder than NP-hard. The intuitive reason for this is that to compute the normalizing constant, we have to *count* how many satisfying assignments there are. (By contrast, MAP estimation is provably easier for some model classes [GPS89], since, intuitively speaking, it only requires finding one satisfying assignment, not counting all of them.) Furthermore, even approximate inference is computationally hard in general [DL93; Rot96].

The above discussion was just concerned with inferring the states of discrete hidden variables. When we have continuous hidden variables, the problem can be even harder, since even a simple two-node graph, of the form $z \to y$, can be intractable to invert if the variables are high dimensional and do not have a conjugate relationship (Section 3.4). Inference in mixed discrete-continuous models can also be hard [LP01].

As a consequence of these hardness results, we often have to resort to approximate inference methods, such as variational inference (Chapter 10) and Monte Carlo inference (Chapter 11).

9.5.5 Drawbacks of VE

Consider using VE to compute all the marginals in a chain-structured graphical model, such as an HMM. We can easily compute the final marginal $p(z_T|\boldsymbol{y})$ by eliminating all the nodes z_1 to z_{T-1} in order. This is equivalent to the forwards algorithm, and takes $O(K^2T)$ time, as we discussed in Section 9.2.3. But now suppose we want to compute $p(z_{T-1}|\boldsymbol{y})$. We have to run VE again, at a cost

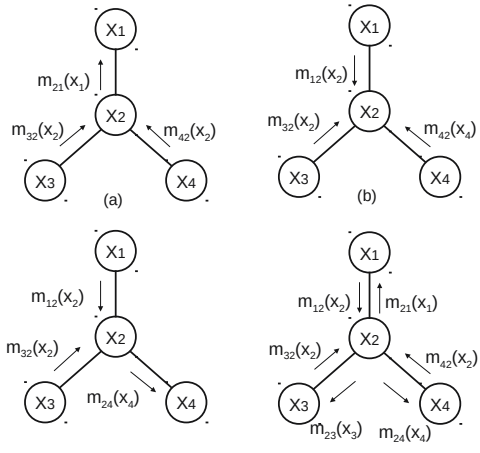


Figure 9.20: Sending multiple messages along a tree. (a) z_1 is root. (b) z_2 is root. (c) z_4 is root. (d) All of the messages needed to compute all singleton marginals. Adapted from Figure 4.3 of [Jor07].

of $O(K^2T)$ time. So the total cost to compute all the marginals is $O(K^2T^2)$. However, we know that we can solve this problem in $O(K^2T)$ using the forwards-backwards, as we discussed in Section 9.2.3. The difference is that FB caches the messages computed on the forwards pass, so it can reuse them later. (Caching previously computed results is the core idea behind dynamic programming.)

The same problem arises when applying VE to trees. For example, consider the 4-node tree in Figure 9.20. We can compute $p(z_1|\boldsymbol{y})$ by eliminating $\boldsymbol{z}_{2:4}$; this is equivalent to sending messages up to z_1 (the messages correspond to the τ factors created by VE). Similarly we can compute $p(z_2|\boldsymbol{y})$, $p(z_3|\boldsymbol{y})$ and then $p(z_4|\boldsymbol{y})$. We see that some of the messages used to compute the marginal on one node can be re-used to compute the marginals on the other nodes. By storing the messages for later re-use, we can compute all the marginals in $O(K^2T)$ time, as we show in Section 9.3.

The question is: how do we get these benefits of message passing on a tree when the graph is not a tree? We give the answer in Section 9.6.

9.6 The junction tree algorithm (JTA)

The **junction tree algorithm** or **JTA** is a generalization of variable elimination that lets us efficiently compute all the posterior marginals without repeating redundant work, by using dynamic programming, thus avoiding the problems mentioned in Section 9.5.5. The basic idea is to convert the graph into a special kind of tree, known as a **junction tree** (also called a **join tree**, or **clique tree**), and then to run belief propagation (message passing) on this tree. We can create the join tree by running variable elimination "symbolically", as discussed in Section 9.5.2, and adding the generated fill-in edges to the graph. The resulting chordal graph can then be converted to a tree, as explained in Supplementary Section 9.2.1. Once we have a tree, we can perform message passing on it, using a variant of the method Section 9.3.2. See Supplementary Section 9.2.2 for details.

9.7 Inference as optimization

In this section, we discuss how to perform posterior inference by solving an optimization problem, which is often computationally simpler. See also Supplementary Section 9.3.

9.7.1 Inference as backpropagation

In this section, we discuss how to compute posterior marginals in a graphical model using automatic differentiation. For notational simplicity, we focus on undirected graphical models, where the joint can be represented as an exponential family (Section 2.4) follows:

$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_c \psi_c(\boldsymbol{x}_c) = \exp(\sum_c \boldsymbol{\eta}_c^\mathsf{T}\mathcal{T}(\boldsymbol{x}_c) - \log A(\boldsymbol{\eta})) = \exp(\boldsymbol{\eta}^\mathsf{T}\mathcal{T}(\boldsymbol{x}) - \log A(\boldsymbol{\eta})) \tag{9.93}$$

where ψ_c is the potential function for clique c, $\boldsymbol{\eta}_c$ are the natural parameters for clique c, $\mathcal{T}(\boldsymbol{x}_c)$ are the corresponding sufficient statistics, and $A = \log Z$ is the log partition function.

We will consider pairwise models (with node and edge potentials), and discrete variables. The natural parameters are the node and edge log potentials, $\boldsymbol{\eta} = (\{\eta_{s;j}\}, \{\eta_{s,t;j,k}\})$, and the sufficient statistics are node and edge indicator functions, $\mathcal{T}(\boldsymbol{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$. (Note: we use $s, t \in \mathcal{V}$ to index nodes and $j, k \in \mathcal{X}$ to index states.)

The mean of the sufficient statistics are given by

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\boldsymbol{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s\neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{st;jk}\}_{s\neq t}) \tag{9.94}$$

The key result, from Equation (2.236), is that $\boldsymbol{\mu} = \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta})$. Thus as long as we have a function that computes $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$, we can use automatic differentiation (Section 6.2) to compute gradients, and then we can extract the corresponding node marginals from the gradient vector. If we have evidence (known values) on some of the variables, we simply "clamp" the corresponding entries to 0 or 1 in the node potentials.

The observation that probabilistic inference can be performed using automatic differentiation has been discovered independently by several groups (e.g., [Dar03; PD03; Eis16; ASM17]). It also lends itself to the development of differentiable approximations to inference (see e.g., [MB18]).

9.7.1.1 Example: inference in a small model

As a concrete example, consider a small chain structured model $x_1 - x_2 - x_3$, where each node has K states. We can represent the node potentials as $K \times 1$ tensors (table of numbers), and the edge potentials by $K \times K$ tensors. The partition function is given by

$$Z(\boldsymbol{\psi}) = \sum_{x_1,x_2,x_3} \psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_{12}(x_1,x_2)\psi_{23}(x_2,x_3) \tag{9.95}$$

Let $\boldsymbol{\eta} = \log(\boldsymbol{\psi})$ be the log potentials, and $A(\boldsymbol{\eta}) = \log Z(\boldsymbol{\eta})$ be the log partition function. We can compute the single node marginals $\boldsymbol{\mu}_s = p(x_s = 1 : K)$ using $\boldsymbol{\mu}_s = \nabla_{\boldsymbol{\eta}_s} A(\boldsymbol{\eta})$, and the pairwise marginals $\boldsymbol{\mu}_{s,t}(j,k) = p(x_s = j, x_t = k)$ using $\boldsymbol{\mu}_{s,t} = \nabla_{\boldsymbol{\eta}_{s,t}} A(\boldsymbol{\eta})$.

We can compute the partition function Z efficiently use numpy's **einsum** function, which implements tensor contraction using Einstein summation notation. We label each dimension of the tensors

by A, B, and C, so einsum knows how to match things up. We then compute gradients using an auto-diff library.[5] The result is that inference can be done in two lines of Python code, as shown in Listing 9.1:

Listing 9.1: Computing marginals from derivative of log partition function

```
import jax.numpy as jnp
from jax import grad

logZ_fun = lambda logpots: np.log(jnp.einsum("A,B,C,AB,BC",
      *[jnp.exp(lp) for lp in logpots]))
probs = grad(logZ_fun)(logpots)
```

To perform conditional inference, such as $p(x_s = k|x_t = e)$, we multiply in one-hot indicator vectors to clamp x_t to the value e so that the unnormalized joint only assigns non-zero probability to state combinations that are valid. We then sum over all values of the unclamped variables to get the constrained partition function Z_e. The gradients will now give us the marginals conditioned on the evidence [Dar03].

9.7.2 Perturb and MAP

In this section, we discuss how to draw posterior samples from a graphical model by leveraging optimization as a subroutine. The basic idea is to make S copies of the model, each of which has slightly perturbed versions of the parameters, $\boldsymbol{\theta}_s = \boldsymbol{\theta}_s + \boldsymbol{\epsilon}_s$, and then to compute the MAP estimate, $\boldsymbol{x}_s = \text{argmax}\, p(\boldsymbol{x}|\boldsymbol{y}; \boldsymbol{\theta}_s)$. For a suitably chosen noise distribution for $\boldsymbol{\epsilon}_s$, this technique — known as **perturb-and-MAP** — can be shown that this gives exact posterior samples [PY10; PY11; PY14].

9.7.2.1 Gaussian case

We first consider the case of a Gaussian MRF. Let $\boldsymbol{x} \in \mathbb{R}^N$ be the vector of hidden states with prior

$$p(\boldsymbol{x}) \propto \mathcal{N}(\mathbf{G}\boldsymbol{x}|\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p) \propto \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_x\boldsymbol{x} + \boldsymbol{h}_x^\mathsf{T}\boldsymbol{x}) \tag{9.96}$$

where $\mathbf{G} \in \mathbb{R}^{K\times N}$ is a matrix that represents prior dependencies (e.g., pairwise correlations), $\mathbf{K}_x = \mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\mathbf{G}$, and $\boldsymbol{h}_x = \mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\mu}_p$. Let $\boldsymbol{y} \in \mathbb{R}^M$ be the measurements with likelihood

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\mathbf{H}\boldsymbol{x} + \boldsymbol{c}, \boldsymbol{\Sigma}_n) \propto \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_{y|x}\boldsymbol{x} + \boldsymbol{h}_{y|x}^\mathsf{T}\boldsymbol{x} - \frac{1}{2}\boldsymbol{y}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}\boldsymbol{y}) \tag{9.97}$$

where $\mathbf{H} \in \mathbb{R}^{M\times N}$ represents dependencies between the hidden and visible variables, $\mathbf{K}_{y|x} = \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}\mathbf{H}$ and $\boldsymbol{h}_{y|x} = \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{y} - \boldsymbol{c})$. The posterior is given by the following (cf. one step of the information filter in Section 8.2.4)

$$p(\boldsymbol{x}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{9.98}$$

$$\boldsymbol{\Sigma}^{-1} = \mathbf{K} = \mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\mathbf{G} + \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}\mathbf{H} \tag{9.99}$$

$$\boldsymbol{\mu} = \mathbf{K}(\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\mu}_p + \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{y} - \boldsymbol{c})) \tag{9.100}$$

5. See ugm_inf_autodiff.py for the full (JAX) code, and see https://github.com/srush/ProbTalk for a (PyTorch) version by Sasha Rush.

where we have assumed $\mathbf{K} = \mathbf{K}_x + \mathbf{K}_{y|x}$ is invertible (although the prior or likelihood on their own may be singular).

The K rows of $\mathbf{G} = [\boldsymbol{g}_1^\mathsf{T}; \ldots; \boldsymbol{g}_K^\mathsf{T}]$ and the M rows of $\mathbf{H} = [\boldsymbol{h}_1^\mathsf{T}; \ldots; \boldsymbol{h}_M^\mathsf{T}]$ can be combined into the L rows of $\mathbf{F} = [\boldsymbol{f}_1^\mathsf{T}; \ldots; \boldsymbol{f}_L]$, which define the linear constraints of the system. If we assume that $\boldsymbol{\Sigma}_p$ and $\boldsymbol{\Sigma}_n$ are diagonal, then the structure of the graphical model is uniquely determined by the sparsity of $\mathbf{F}$. The resulting posterior factorizes as a product of L Gaussian "experts":

$$p(\boldsymbol{x}|\boldsymbol{y}) \propto \prod_{l=1}^{L} \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_l\boldsymbol{x} + \boldsymbol{h}_l^\mathsf{T}\boldsymbol{x}) \propto \prod_{l=1}^{L} \mathcal{N}(\boldsymbol{f}_l^\mathsf{T}\boldsymbol{x}; \mu_l, \Sigma_l) \tag{9.101}$$

where Σ_l equals $\boldsymbol{\Sigma}_{p,l,l}$ for $l = 1:K$ and equals $\boldsymbol{\Sigma}_{n,l',l'}$ for $l = K+1:L$ where $l' = l - K$. Similarly $\mu_l = \boldsymbol{\mu}_{p,l}$ for $l = 1:K$ and $\mu_l = (y_{l'} - c_{l'})$ for $l = K+1:L$.

To apply perturb and MAP, we proceed as follows. First perturb the prior mean by sampling $\tilde{\boldsymbol{\mu}}_p \sim \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$, and perturb the measurements by sampling $\tilde{\boldsymbol{y}} \sim \mathcal{N}(\boldsymbol{y}, \boldsymbol{\Sigma}_n)$. (Note that this is equivalent to first perturbing the linear term in each information form potential, using $\tilde{\boldsymbol{h}}_l = \boldsymbol{h}_l + \boldsymbol{f}_l\boldsymbol{\Sigma}_l^{-\frac{1}{2}}\epsilon_l$, where $\epsilon_l \sim \mathcal{N}(0,1)$.) Then compute the MAP estimate for $\boldsymbol{x}$ using the perturbed parameters:

$$\tilde{\boldsymbol{x}} = \mathbf{K}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\tilde{\boldsymbol{\mu}}_p + \mathbf{K}^{-1}\mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}(\tilde{\boldsymbol{y}} - \boldsymbol{c}) \tag{9.102}$$

$$= \underbrace{\mathbf{K}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}}_{\mathbf{A}}(\boldsymbol{\mu}_p + \boldsymbol{\epsilon}_\mu) + \underbrace{\mathbf{K}^{-1}\mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}}_{\mathbf{B}}(\boldsymbol{y} + \boldsymbol{\epsilon}_y - \boldsymbol{c}) \tag{9.103}$$

$$= \boldsymbol{\mu} + \mathbf{A}\boldsymbol{\epsilon}_\mu + \mathbf{B}\boldsymbol{\epsilon}_y \tag{9.104}$$

We see that $\mathbb{E}[\tilde{\boldsymbol{x}}] = \boldsymbol{\mu}$ and $\mathbb{E}\left[(\tilde{\boldsymbol{x}} - \boldsymbol{\mu})(\tilde{\boldsymbol{x}} - \boldsymbol{\mu})^\mathsf{T}\right] = \mathbf{K}^{-1} = \boldsymbol{\Sigma}$, so the method produces exact samples.

This approach is very scalable, since compute the MAP estimate of sparse GMRFs (i.e., posterior mean) can be done efficiently using conjugate gradient solvers. Alternatively we can use loopy belief propagation (Section 9.4), which can often compute the exact posterior mean (see e.g., [WF01a; JMW06; Bic09; Du+18]).

9.7.2.2 Discrete case

In [PY11; PY14] they extend perturb-and-MAP to the case of discrete graphical models. This setup is more complicated, and requires the use of Gumbel noise, which can be sampled using $\epsilon = -\log(-\log(u))$, where $u \sim \text{Unif}(0,1)$. This noise should be added to all the potentials in the model, but as a simple approximation, it can just be added to the unary terms, i.e., the local evidence potentials. Let the score, or unnormalized log probability, of configuration $\boldsymbol{x}$ given inputs $\boldsymbol{c}$ be

$$S(\boldsymbol{x}; \boldsymbol{c}) = \log p(\boldsymbol{x}|\boldsymbol{c}) + \text{const} = \sum_i \log \phi_i(x_i) + \sum_{ij} \log \psi_{ij}(x_{i,j}) \tag{9.105}$$

where we have assumed a pairwise CRF for notational simplicity. If we perturb the local evidence potentials $\phi_i(k)$ by adding ϵ_{ik} to each entry, where k indexes the discrete latent states, we get $\tilde{S}(\boldsymbol{x}; \boldsymbol{c})$. We then compute a sample $\tilde{\boldsymbol{x}}$ by solving $\tilde{\boldsymbol{x}} = \text{argmax}\, \tilde{S}(\boldsymbol{x}; \boldsymbol{c})$. The advantage of this approach is that it can leverage efficient MAP solvers for discrete models, such as those discussed in Supplementary Section 9.3. This can in turn be used for parameter learning, and estimating the partition function [HJ12; Erm+13].

10 Variational inference

10.1 Introduction

In this chapter, we discuss **variational inference**, which reduces posterior inference to optimization. Note that VI is a large topic; this chapter just gives a high level overview. For more details, see e.g., [Jor+98; JJ00; Jaa01; WJ08; SQ05; TLG08; Zha+19b; Bro18].

10.1.1 The variational objective

Consider a model with unknown (latent) variables $\boldsymbol{z}$, known variables $\boldsymbol{x}$, and fixed parameters $\boldsymbol{\theta}$. (If the parameters are unknown, they can be added to $\boldsymbol{z}$, as we discuss later.) We assume the prior is $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and the likelihood is $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, so the unnormalized joint is $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})$, and the posterior is $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})/p_{\boldsymbol{\theta}}(\boldsymbol{x})$. We assume that it is intractable to compute the normalization constant, $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})d\boldsymbol{z}$, and hence intractable to compute the normalized posterior. We therefore seek an approximation to the posterior, which we denote by $q(\boldsymbol{z})$, such that we minimize the following loss:

$$q = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathbb{KL}}\left(q(\boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \tag{10.1}$$

Since we are minimizing over functions (namely distributions q), this is called a **variational method**.

In practice we pick a parametric family $\mathcal{Q}$, where we use $\boldsymbol{\psi}$, known as the **variational parameters**, to specify which member of the family we are using. We can compute the best variational parameters (for given $\boldsymbol{x}$) as follows:

$$\boldsymbol{\psi}^* = \operatorname*{argmin}_{\boldsymbol{\psi}} D_{\mathbb{KL}}\left(q_{\boldsymbol{\psi}}(\boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \tag{10.2}$$

$$= \operatorname*{argmin}_{\boldsymbol{\psi}} \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log q_{\boldsymbol{\psi}}(\boldsymbol{z}) - \log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right)\right] \tag{10.3}$$

$$= \operatorname*{argmin}_{\boldsymbol{\psi}} \underbrace{\mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log q_{\boldsymbol{\psi}}(\boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right]}_{\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x})} + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.4}$$

The final term $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log(\int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})d\boldsymbol{z})$ is generally intractable to compute. Fortunately, it is independent of $\boldsymbol{\psi}$, so we can drop it. This leaves us with the first term, which we write as follows:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) + \log q_{\boldsymbol{\psi}}(\boldsymbol{z})\right] \tag{10.5}$$

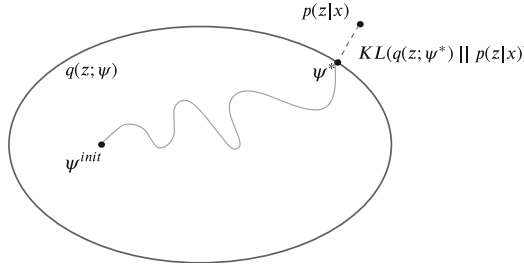


Figure 10.1: Illustration of variational inference. The large oval represents the set of variational distributions $\mathcal{Q} = \{q_{\boldsymbol{\psi}}(\boldsymbol{z}) : \boldsymbol{\psi} \in \ominus\}$, where $\ominus$ is the set of possible variational parameters. The true distribution is the point $p(\boldsymbol{z}|\boldsymbol{x})$, which we assume lies outside the set. Our goal is to find the best approximation to p within our variational family; this is the point $\boldsymbol{\psi}^*$ which is closest in KL divergence. We find this point by starting an optimization procedure from the random initial point $\boldsymbol{\psi}^{init}$. Adapted from a figure by David Blei.

Minimizing this objective will minimize the KL divergence, causing our approximation to approach the true posterior. See Figure 10.1 for an illustration. In the sections below, we give two different interpretations of this objective function.

10.1.1.1 The view from physics: minimize the variational free energy

If we define $\mathcal{E}_{\boldsymbol{\theta}}(\boldsymbol{z}) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$ as the energy, then we can rewrite the loss in Equation (10.5)

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\mathcal{E}_{\boldsymbol{\theta}}(\boldsymbol{z})\right] - \mathbb{H}(q_{\boldsymbol{\psi}}) = \text{expected energy} - \text{entropy} \tag{10.6}$$

In physics, this is known as the **variational free energy** (VFE). This is an upper bound on the **free energy** (FE), $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which follows from the fact that

$$D_{\mathbb{KL}}\left(q_{\boldsymbol{\psi}}(\boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) = \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \tag{10.7}$$

$$\underbrace{\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x})}_{\text{VFE}} \geq \underbrace{-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})}_{\text{FE}} \tag{10.8}$$

Variational inference is equivalent to minimizing the VFE. If we reach the minimum value of $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, then the KL divergence term will be 0, so our approximate posterior will be exact.

10.1.1.2 The view from statistics: maximize the evidence lower bound (ELBO)

The negative of the VFE is known as the **evidence lower bound** or **ELBO** function [BKM16]:

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) \triangleq \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z})\right] = \text{ELBO} \tag{10.9}$$

The name "ELBO" arises because

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.10}$$

where $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called the "evidence". The inequality follows from Equation (10.8). Therefore maximizing the ELBO wrt $\boldsymbol{\psi}$ will minimize the original KL, since $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a constant wrt $\boldsymbol{\psi}$.

(Note: we use the symbol Ł for the ELBO, rather than $\mathcal{L}$, since we want to maximize Ł but minimize $\mathcal{L}$.)

We can rewrite the ELBO as follows:

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}(q_{\boldsymbol{\psi}}(\boldsymbol{z})) \tag{10.11}$$

We can interpret this

$$\text{ELBO} = \text{expected log joint} + \text{entropy} \tag{10.12}$$

The second term encourages the posterior to be maximum entropy, while the first term encourages it to be a joint MAP configuration.

We can also rewrite the ELBO as

$$\begin{aligned} Ł(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z})\right] &(10.13)\\ &= \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\mathbb{KL}}\left(q_{\boldsymbol{\psi}}(\boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) &(10.14)\end{aligned}$$

We can interpret this as follows:

$$\text{ELBO} = \text{expected log likelihood} - \text{KL from posterior to prior} \tag{10.15}$$

The KL term acts like a regularizer, preventing the posterior from diverging too much from the prior.

10.1.2 Form of the variational posterior

There are two main approaches for choosing the form of the variational posterior, $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})$. In the first approach, we pick a convenient functional form, such as multivariate Gaussian, and then optimize the ELBO using gradient-based methods. This is called **fixed-form VI**, and is discussed in Section 10.2. An alternative is to make the **mean field** assumption, namely that the posterior factorizes:

$$q_{\boldsymbol{\psi}}(\boldsymbol{z}) = \prod_{j=1}^{J} q_j(\boldsymbol{z}_j) \tag{10.16}$$

where $q_j(\boldsymbol{z}_j) = q_{\boldsymbol{\psi}_j}(\boldsymbol{z}_j)$ is the posterior over the j'th group of variables. We don't need to specify the functional form for each q_j. Instead, the optimal distributional form can be derived by maximizing the ELBO wrt each group of variational parameters one at a time, in a coordinate ascent manner. This is therefore called **free-form VI**, and is discussed in Section 10.3.

We now give a simple example of variational inference applied to a 2d latent vector $\boldsymbol{z}$, representing the mean of a Gaussian. The prior is $\mathcal{N}(\boldsymbol{z}|\breve{\boldsymbol{m}}, \breve{\mathbf{V}})$, and the likelihood is

$$p(\mathcal{D}|\boldsymbol{z}) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{z}, \boldsymbol{\Sigma}) \propto \mathcal{N}(\overline{\boldsymbol{x}}|\boldsymbol{z}, \frac{1}{N}\boldsymbol{\Sigma}) \tag{10.17}$$

The exact posterior, $p(\boldsymbol{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{z}|\ \widehat{\boldsymbol{m}}, \widehat{\mathbf{V}})$, can be computed analytically, as discussed in Section 3.4.4.1. In Figure 10.2, we compare three Gaussian variational approximations to the posterior. If q uses a full covariance matrix, it matches the exact posterior; however, this is intractable in high

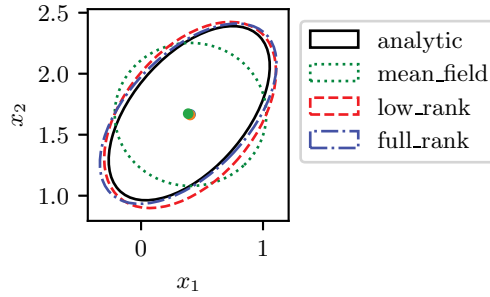


Figure 10.2: Variational approximation to the exact (Gaussian) posterior for the mean of a 2d Gaussian likelihood with a Gaussian prior. We show 3 Gaussian approximations to the posterior, using a full covariance (blue), a diagonal covariance (green), and a diagonal plus rank one covariance (red). Generated by gaussian_2d_vi.ipynb.

dimensions. If q uses a diagonal covariance matrix (corresponding to the mean field approximation), we see that the approximation is over confident, which is a well-known flaw of variational inference, due to the mode-seeking nature of minimizing $D_{\mathbb{KL}}(q \parallel p)$ (see Section 5.1.4.3 for details). Finally, if q uses a rank-1 plus diagonal approximation, we get a much better approximation; furthermore, this can be computed quite efficiently, as we discuss in Section 10.2.1.3.

10.1.3 Parameter estimation using variational EM

So far, we have assumed the model parameters $\boldsymbol{\theta}$ are known. However, we can try to estimate them by maximing the log marginal likelihood of the dataset, $\mathcal{D} = \{\boldsymbol{x}_n : n = 1 : N\}$,

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}_n|\boldsymbol{\theta}) \tag{10.18}$$

In general, this is intractable to compute, but we discuss approximations below.

10.1.3.1 MLE for latent variable models

Suppose we have a latent variable model of the form

$$p(\mathcal{D}, \boldsymbol{z}_{1:N}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{z}_n|\boldsymbol{\theta}) p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}) \tag{10.19}$$

as shown in Figure 10.3a. Furthermore, suppose we want to compute the MLE for $\boldsymbol{\theta}$ given the dataset $\mathcal{D} = \{\boldsymbol{x}_n : n = 1 : N\}$. Since the local latent variables $\boldsymbol{z}_n$ are hidden, we must marginalize them out to get the local (per example) log marginal likelihood:

$$\log p(\boldsymbol{x}_n|\boldsymbol{\theta}) = \log \left[\int p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}) p(\boldsymbol{z}_n|\boldsymbol{\theta}) d\boldsymbol{z}_n \right] \tag{10.20}$$

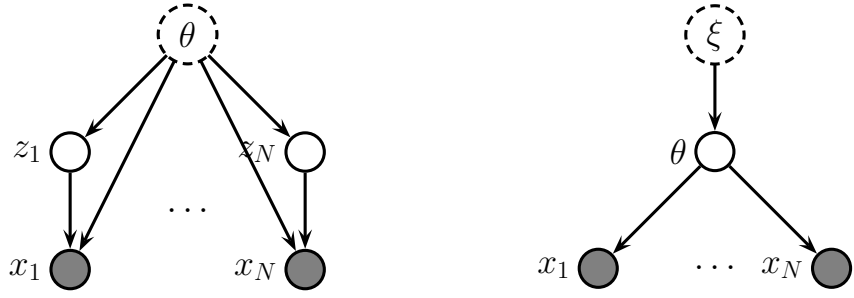


Figure 10.3: Graphical models with: (a) Local stochastic latent variables $\boldsymbol{z}_n$ and global deterministic latent parameter $\boldsymbol{\theta}$. (b) Global stochastic latent parameter $\boldsymbol{\theta}$ and global deterministic latent hyper-parameter $\boldsymbol{\xi}$. The observed variables $\boldsymbol{x}_n$ are shown by shaded circles.

Unfortunately, computing this integral is usually intractable, since it corresponds to the normalization constant of the exact posterior. Fortunately, the ELBO is a lower bound on this:

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_n | \boldsymbol{x}_n) \leq \log p(\boldsymbol{x}_n | \boldsymbol{\theta}) \tag{10.21}$$

We can thus optimize the model parameters by maximizing

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_{1:N} | \mathcal{D}) \triangleq \sum_{n=1}^{N} Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_n | \boldsymbol{x}_n) \leq \log p(\mathcal{D} | \boldsymbol{\theta}) \tag{10.22}$$

This is the basis of the **variational EM** algorithm. We discuss this in more detail in Section 6.5.6.1, but the basic idea is to alternate between maximizing the ELBO wrt the variational parameters $\{\boldsymbol{\psi}_n\}$ in the E step, to give us $q_{\boldsymbol{\psi}_n}(\boldsymbol{z}_n)$, and then maximizing the ELBO (using the new $\boldsymbol{\psi}_n$) wrt the model parameters $\boldsymbol{\theta}$ in the M step. (We can also use SGD and amortized inference to speed this up, as we explain in Sections 10.1.4 to 10.1.5.)

10.1.3.2 Empirical Bayes for fully observed models

Suppose we have a fully observed model (with no local latent variables) of the form

$$p(\mathcal{D}, \boldsymbol{\theta} | \boldsymbol{\xi}) = p(\boldsymbol{\theta} | \boldsymbol{\xi}) \prod_{n=1}^{N} p(\boldsymbol{x}_n | \boldsymbol{\theta}) \tag{10.23}$$

as shown in Figure 10.3b. In the context of Bayesian parameter inference, our goal is to compute the parameter posterior:

$$p(\boldsymbol{\theta} | \mathcal{D}, \boldsymbol{\xi}) = \frac{p(\mathcal{D} | \boldsymbol{\theta}) p(\boldsymbol{\theta} | \boldsymbol{\xi})}{p(\mathcal{D} | \boldsymbol{\xi})} \tag{10.24}$$

where $\boldsymbol{\theta}$ are the global unknown model parameters (latent variables), and $\boldsymbol{\xi}$ are the hyper-parameters for the prior. If the hyper-parameters are unknown, we can estimate them using empirical Bayes (see

Section 3.7) by computing

$$\hat{\boldsymbol{\xi}} = \underset{\boldsymbol{\xi}}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\xi}) \tag{10.25}$$

We can use variational EM to compute this, similar to Section 10.1.3.1, except now the parameters to be estimated are $\boldsymbol{\xi}$, the latent variables are the shared global parameters $\boldsymbol{\theta}$, and the observations are the entire dataset, $\mathcal{D}$. We then get the lower bound

$$\log p(\mathcal{D}|\boldsymbol{\xi}) \geq Ł(\boldsymbol{\xi}, \boldsymbol{\psi}|\mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\sum_{n=1}^{N} \log p(\boldsymbol{x}_n|\boldsymbol{\theta})\right] - D_{\mathbb{KL}}\left(q_{\boldsymbol{\psi}}(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\boldsymbol{\xi})\right) \tag{10.26}$$

We optimize this wrt the parameters of the variational posterior, $\boldsymbol{\psi}$, and wrt the prior hyper-parameters $\boldsymbol{\xi}$.

If the prior $\boldsymbol{\xi}$ is fixed, we just need to optimize the variational parameters $\boldsymbol{\psi}$ to compute the posterior, $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\mathcal{D})$. This is known as **variational Bayes**. See Section 10.3.3 for more details.

10.1.4 Stochastic VI

In Section 10.1.3, we saw that parameter estimation requires optimizing the ELBO for the entire dataset, which is defined as the sum of the ELBOs for each of the N data samples $\boldsymbol{x}_n$. Computing this objective can be slow if N is large. Fortunately, we can replace this objective with a stochastic approximation, which is faster to compute, and provides an unbiased estimate. In particular, at each step, we can draw a random minibatch of $B = |\mathcal{B}|$ examples from the dataset, and then make the approximation

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_{1:N}|\mathcal{D}) = \sum_{n=1}^{N} Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_n|\boldsymbol{x}_n) \approx \frac{N}{B} \sum_{\boldsymbol{x}_n \in \mathcal{B}} \left[\mathbb{E}_{q_{\boldsymbol{\psi}_n}(\boldsymbol{z}_n)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n|\boldsymbol{z}_n) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_n) - \log q_{\boldsymbol{\psi}_n}(\boldsymbol{z}_n)\right]\right] \tag{10.27}$$

This can be used inside of a stochastic optimization algorithm, such as SGD. This is called **stochastic variational inference** or **SVI** [Hof+13], and allows VI to scale to large datasets.

10.1.5 Amortized VI

In Section 10.1.4, we saw that in each iteration of SVI, we need to optimize the local variational parameters $\boldsymbol{\psi}_n$ for each example n in the minibatch. This nested optimization can be quite slow.

An alternative approach is to train a model, known as an **inference network** or **recognition network**, to predict $\boldsymbol{\psi}_n$ from the observed data, $\boldsymbol{x}_n$, using $\boldsymbol{\psi}_n = f_{\boldsymbol{\phi}}^{\text{inf}}(\boldsymbol{x}_n)$. This technique is known as **amortized variational inference** [GG14], or **inference compilation** [LBW17], since we are reducing the cost of per-example time inference by training a model that is shared across all examples. (See also [Amo22] for a general discussion of amortized optimization.) For brevity, we will write the amortized posterior as

$$q(\boldsymbol{z}_n|\boldsymbol{\psi}_n) = q(\boldsymbol{z}_n|f_{\boldsymbol{\phi}}^{\text{inf}}(\boldsymbol{x}_n)) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n) \tag{10.28}$$

The corresponding ELBO becomes

$$\text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathcal{D}) = \sum_{n=1}^{N} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_n)\right]\right] \tag{10.29}$$

If we combine this with SVI we get an amortized version of Equation (10.27). For example, if we use a minibatch of size 1, we get

$$\text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}_n) \approx N \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_n)\right]\right] \tag{10.30}$$

We can optimize this as shown in Algorithm 10.1. Note that the (partial) maximization wrt $\boldsymbol{\theta}$ in the M step is usually done with a gradient update, but the maximization wrt $\boldsymbol{\phi}$ in the E step is trickier, since the loss uses $\boldsymbol{\phi}$ to define an expectation operator, so we can't necessarily push the gradient operator inside; we discuss ways to optimize the variational parameters in Section 10.2 and Section 10.3.

Algorithm 10.1: Amortized stochastic variational EM

Initialize $\boldsymbol{\theta}$, $\boldsymbol{\phi}$
repeat
  Sample $\boldsymbol{x}_n \sim p_{\mathcal{D}}$
  E step: $\boldsymbol{\phi} = \text{argmax}_{\boldsymbol{\phi}}\, \text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}_n)$
  M step: $\boldsymbol{\theta} = \text{argmax}_{\boldsymbol{\theta}}\, \text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}_n)$
until *converged*

10.1.6 Semi-amortized inference

Amortized SVI is widely used for fitting LVMs, e.g., for VAEs (see Section 21.2), for topic models [SS17a], for probabilistic programming [RHG16], for CRFs [TG18], etc. However, the use of an inference network can result in a suboptimal setting of the local variational parameters $\boldsymbol{\psi}_n$. This is called the **amortization gap** [CLD18]. We can close this gap by using the inference network to warm-start an optimizer for $\boldsymbol{\psi}_n$; this is known as **semi-amortized VI** [Kim+18c]. (See also [MYM18], who propose a closely related method called **iterative amortized inference**.)

An alternative approach is to use the inference network as a proposal distribution. If we combine this with importance sampling, we get the IWAE bound of Section 10.5.1. If we use this with Metropolis-Hastings, we get a VI-MCMC hybrid (see Section 10.4.5).

10.2 Gradient-based VI

In this section, we will choose some convenient form for $q_{\boldsymbol{\psi}}(\boldsymbol{z})$, such as a Gaussian for continuous $\boldsymbol{z}$, or a product of categoricals for discrete $\boldsymbol{z}$, and then optimize the ELBO using gradient based methods.

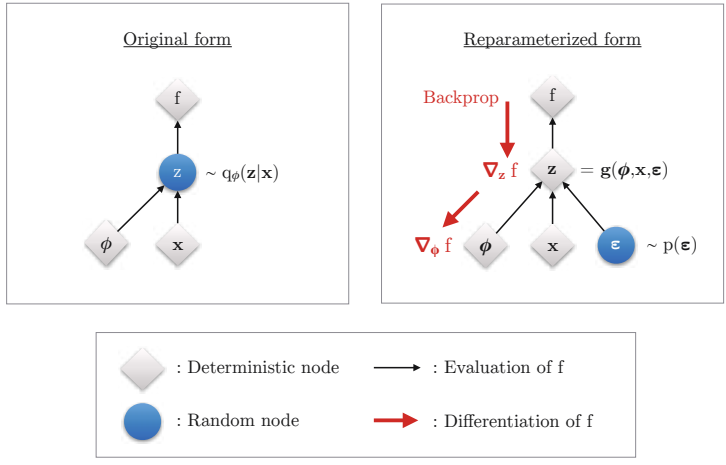


Figure 10.4: Illustration of the reparameterization trick. The objective f depends on the variational parameters $\boldsymbol{\phi}$, the observed data $\boldsymbol{x}$, and the latent random variable $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. On the left, we show the standard form of the computation graph. On the right, we show a reparameterized form, in which we move the stochasticity into the noise source $\boldsymbol{\epsilon}$, and compute $\boldsymbol{z}$ deterministically, $\boldsymbol{z} = g(\boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{\epsilon})$. The rest of the graph is deterministic, so we can backpropagate the gradient of the scalar f wrt $\boldsymbol{\phi}$ through $\boldsymbol{z}$ and into $\boldsymbol{\phi}$. From Figure 2.3 of [KW19a]. Used with kind permission of Durk Kingma.

The gradient wrt the generative parameters $\boldsymbol{\theta}$ is easy to compute, since we can push gradients inside the expectation, and use a single Monte Carlo sample:

$$\nabla_{\boldsymbol{\theta}} \text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] \tag{10.31}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\nabla_{\boldsymbol{\theta}} \left\{\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right\}\right] \tag{10.32}$$

$$\approx \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}^s) \tag{10.33}$$

where $\boldsymbol{z}^s \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. This is an unbiased estimate of the gradient, so can be used with SGD.

The gradient wrt the inference parameters $\boldsymbol{\phi}$ is harder to compute since

$$\nabla_{\boldsymbol{\phi}} \text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] \tag{10.34}$$

$$\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\nabla_{\boldsymbol{\phi}} \left\{\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right\}\right] \tag{10.35}$$

However, we can often use the reparameterization trick, which we discuss in Section 10.2.1. If not, we can use blackbox VI, which we discuss in Section 10.2.3.

10.2.1 Reparameterized VI

In this section, we discuss the **reparameterization trick** for taking gradients wrt distributions over continuous latent variables $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. We explain this in detail in Section 6.3.5, but we summarize the basic idea here.

The key trick is to rewrite the random variable $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ as some differentiable (and invertible) transformation g of another random variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, which does not depend on $\boldsymbol{\phi}$, i.e., we assume we can write

$$\boldsymbol{z} = g(\boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{\epsilon}) \tag{10.36}$$

For example,

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})) \iff \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{10.37}$$

Using this, we have

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[f(\boldsymbol{z})\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\boldsymbol{z})\right] \;\; \text{s.t.} \;\; \boldsymbol{z} = g(\boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{\epsilon}) \tag{10.38}$$

where we define

$$f_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{z}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \tag{10.39}$$

Hence

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[f(\boldsymbol{z})\right] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\boldsymbol{z})\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} f(\boldsymbol{z})\right] \tag{10.40}$$

which we can approximate with a single Monte Carlo sample. This lets us propagate gradients back through the f function. See Figure 10.4 for an illustration. This is called **reparameterized VI** or **RVI**.

Since we are now working with the random variable $\boldsymbol{\epsilon}$, we need to use the change of variables formula to compute

$$\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \log p(\boldsymbol{\epsilon}) - \log \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right) \right| \tag{10.41}$$

where $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}$ is the Jacobian:

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \begin{pmatrix} \frac{\partial z_1}{\partial \epsilon_1} & \cdots & \frac{\partial z_1}{\partial \epsilon_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_k}{\partial \epsilon_1} & \cdots & \frac{\partial z_k}{\partial \epsilon_k} \end{pmatrix} \tag{10.42}$$

We design the transformation $\boldsymbol{z} = g(\boldsymbol{\epsilon})$ such that this Jacobian is tractable to compute. We give some examples below.

10.2.1.1 Gaussian with diagonal covariance (mean field)

Suppose we use a fully factorized Gaussian posterior. Then the reparameterization process becomes

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{10.43}$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{10.44}$$

where the inference network generates the parameters of the transformation:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = f_{\boldsymbol{\phi}}^{\text{inf}}(\boldsymbol{x}) \tag{10.45}$$

Thus to sample from the posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, we sample $\boldsymbol{\epsilon}\mathcal{N}(\mathbf{0}, \mathbf{I})$, and then compute $\boldsymbol{z}$.

Given the sample, we need to evaluate the ELBO:

$$f(\boldsymbol{z}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \tag{10.46}$$

To evaluate the $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ term, we can just plug $\boldsymbol{z}$ into the likelihood. To evaluate the $\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ term, we need to use the change of variables formula from Equation (10.41). The Jacobian is given by $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \text{diag}(\boldsymbol{\sigma})$. Hence

$$\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \sum_{k=1}^{K} [\log \mathcal{N}(\epsilon_k|0,1) - \log \sigma_k] = -\sum_{k=1}^{K} \left[\frac{1}{2}\log(2\pi) + \frac{1}{2}\epsilon_k^2 + \log \sigma_k\right] \tag{10.47}$$

Finally, to evaluate the $p(\boldsymbol{z})$ term, we can use the transformation $\boldsymbol{z} = \mathbf{0} + \mathbf{1} \odot \boldsymbol{\epsilon}$, so the Jacobian is the identity and we get

$$\log p(\boldsymbol{z}) = \sum_{k=1}^{K} \left[\frac{1}{2} z_k^2 + \frac{1}{2}\log(2\pi)\right] \tag{10.48}$$

An alternative is to use the objective

$$f'(\boldsymbol{z}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + D_{\mathbb{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{Z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{Z})\right) \tag{10.49}$$

In some cases, we evaluate the second term analytically, without needing Monte Carlo. For example, if we assume a diagonal Gaussian prior, $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{I})$, and diagonal gaussian posterior, $q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$, we can use Equation (5.78) to compute the KL in closed form:

$$D_{\mathbb{KL}}\left(q \parallel p\right) = -\frac{1}{2}\sum_{k=1}^{K}\left[\log \sigma_k^2 - \sigma_k^2 - \mu_k^2 + 1\right] \tag{10.50}$$

The objective $f'(\boldsymbol{z})$ is often lower variance than $f(\boldsymbol{z})$, since it computes the KL analytically. However, it is harder to generalize this objective to settings where the prior and/or posterior are not Gaussian.

10.2.1.2 Gaussian with full covariance

Now consider using a full covariance Gaussian posterior. We will compute a Cholesky decomposition of the covariance, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$, where $\mathbf{L}$ is a lower triangular matrix with non-zero entries on the diagonal. Hence the reparameterization becomes

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{10.51}$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon} \tag{10.52}$$

The Jacobian of this affine transformation is $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \mathbf{L}$. Since $\mathbf{L}$ is a triangular matrix, its determinant is the product of its main diagonal, so

$$\log \left| \det \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right| = \sum_{k=1}^{K} \log |L_{kk}| \tag{10.53}$$

We can compute $\mathbf{L}$ using

$$\mathbf{L} = \mathbf{M} \odot \mathbf{L}' + \text{diag}(\boldsymbol{\sigma}) \tag{10.54}$$

where $\mathbf{M}$ is a masking matrix with 0s on and above the diagonal, and 1s below the diagonal, and where $(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \mathbf{L}')$ is predicted by the inference network. With this construction, the diagonal entries of $\mathbf{L}$ are given by $\boldsymbol{\sigma}$, so

$$\log \left| \det \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right| = \sum_{k=1}^{K} \log |L_{kk}| = \sum_{k=1}^{K} \log \sigma_k \tag{10.55}$$

10.2.1.3 Gaussian with low-rank plus diagonal covariance

In high dimensions, an efficient alternative to using a Cholesky decomposition is the factor decomposition

$$\boldsymbol{\Sigma} = \mathbf{B}\mathbf{B}^{\mathsf{T}} + \mathbf{C}^2 \tag{10.56}$$

where $\mathbf{B}$ is the factor loading matrix of size $d \times f$, where $f \ll d$ is the number of factors, d is the dimensionality of $\boldsymbol{z}$, and $\mathbf{C} = \text{diag}(c_1, \ldots, c_d)$. This reduces the total number of variational parameters from $d + d(d+1)/2$ to $(f+2)d$. In [ONS18], they called this approach **VAFC** (short for variational approximation with factor covariance).

In the special case where $f = 1$, the covariance matrix becomes

$$\boldsymbol{\Sigma} = \boldsymbol{b}\boldsymbol{b}^{\mathsf{T}} + \text{diag}(\boldsymbol{c}^2) \tag{10.57}$$

In this case, it is possible to compute the natural gradient (Section 6.4) of the ELBO in closed form in $O(d)$ time, as shown in [Tra+20b; TND21], who call the approach **NAGVAC-1** (natural gradient Gaussian variational approximation). This can result in much faster convergence than following the normal gradient.

In Section 10.1.2, we show that this low rank approximation is much better than a diagonal approximation. See Supplementary Section 10.1 for more examples.

10.2.1.4 Other variational posteriors

Many other kinds of distribution can be written in a reparameterizable way, as described in [Moh+20]. This includes standard exponential family distributions, such as the gamma and Dirichlet, as well as more exotic forms, such as inverse autoregressive flows (see Section 10.4.3).

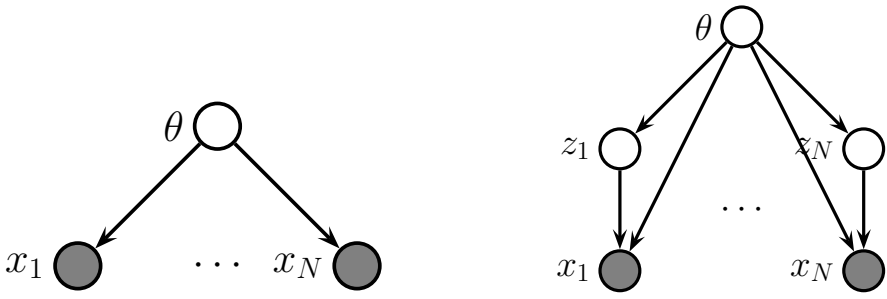


Figure 10.5: Graphical models with (a) Global latent parameter $\boldsymbol{\theta}$ and observed variables $\boldsymbol{x}_{1:N}$. (b) Local latent variables $\boldsymbol{z}_{1:N}$, global latent parameter $\boldsymbol{\theta}$, and observed variables $\boldsymbol{x}_{1:N}$.

10.2.1.5 Example: Bayesian parameter inference

In this section, we use reparameterized SVI to infer the posterior for the parameters of a Gaussian mixture model (GMM). We will marginalize out the discrete latent variables, so just need to approximate the posterior over the global latent, $p(\boldsymbol{\theta}|\mathcal{D})$. This is sometimes called a "**collapsed**" model, since we have marginalized out all the local latent variables. That is, we have converted the model in Figure 10.5b to the one in Figure 10.5a. We choose a factored (mean field) variational posterior that is conjugate to the likelihood, but is also reparameterizable. This lets us fit the posterior with SGD.

For simplicity, we assume diagonal covariance matrices for each Gaussian mixture component. Thus the likelihood for one datapoint, $\boldsymbol{x} \in \mathbb{R}^D$, is

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\lambda}_k)^{-1}) \tag{10.58}$$

where $\boldsymbol{\mu}_k = (\mu_{k1}, \ldots, \mu_{kD})$ are the means, $\boldsymbol{\lambda}_k = (\lambda_{k1}, \ldots, \lambda_{kD})$ are the precisions, and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ are the mixing weights. We use the following prior for these parameters:

$$p_{\boldsymbol{\xi}}(\boldsymbol{\theta}) = \left[\prod_{k=1}^{K}\prod_{d=1}^{D} \mathcal{N}(\mu_{kd}|0,1)\text{Ga}(\lambda_{kd}|5,5)\right] \text{Dir}(\boldsymbol{\pi}|\mathbf{1}) \tag{10.59}$$

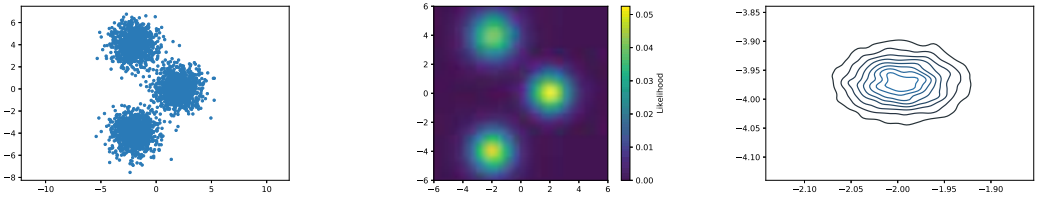
where $\boldsymbol{\xi}$ are the hyperparameters. We assume the following mean field posterior:

$$q_{\boldsymbol{\psi}}(\boldsymbol{\theta}) = \left[\prod_{k=1}^{K}\prod_{d=1}^{D} \mathcal{N}(\mu_{kd}|m_{kd}, s_{kd})\text{Ga}(\lambda_{kd}|\alpha_{kd}, \beta_{kd})\right] \text{Dir}(\boldsymbol{\pi}|\boldsymbol{c}) \tag{10.60}$$

where $\boldsymbol{\psi} = (\boldsymbol{m}_{1:K,1:D}, \boldsymbol{s}_{1:K,1:D}, \boldsymbol{\alpha}_{1:K,1:D}, \boldsymbol{\beta}_{1:K,1:D}, \boldsymbol{c})$ are the variational parameters for $\boldsymbol{\theta}$.

We can compute the ELBO using

$$Ł(\boldsymbol{\xi}, \boldsymbol{\psi}|\mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p_{\boldsymbol{\xi}}(\boldsymbol{\theta}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta})\right] \tag{10.61}$$



(a) (b) (c)

Figure 10.6: SVI for fitting a mixture of 3 Gaussians in 2d. (a) 3000 training points. (b) Fitted density, plugging in the posterior mean parameters. (c) Kernel density estimate fit to 10,000 samples from $q(\boldsymbol{\mu}_1|\boldsymbol{\psi})$. Generated by svi_gmm_demo_2d.ipynb.

Since the distributions are reparameterizable, we can and push gradients inside this expression. We can approximate the expectation by drawing a single posterior sample, and can approximate the log likelihood using minibatching. We can then update the variational parameters, (and optionally the hyperparameters of the prior, as we discussed in Section 10.1.3.2) using the pseudcode in Algorithm 10.2.

Algorithm 10.2: Reparameterized SVI for Bayesian parameter inference

Initialize $\boldsymbol{\psi}$, $\boldsymbol{\xi}$
repeat
  Sample minibatch $\mathcal{B} = \{\boldsymbol{x}_b \sim \mathcal{D} : b = 1 : B\}$
  Sample $\boldsymbol{\epsilon} \sim q_0$
  Compute $\tilde{\boldsymbol{\theta}} = g(\boldsymbol{\psi}, \boldsymbol{\epsilon})$
  Compute $\mathcal{L}(\boldsymbol{\psi}|\mathcal{D}, \tilde{\boldsymbol{\theta}}) = -\frac{N}{B}\sum_{\boldsymbol{x}_n \in \mathcal{B}} \log p(\boldsymbol{x}_n|\tilde{\boldsymbol{\theta}}) - \log p_{\boldsymbol{\xi}}(\tilde{\boldsymbol{\theta}}) + \log q_{\boldsymbol{\psi}}(\tilde{\boldsymbol{\theta}})$
  Update $\boldsymbol{\xi} := \boldsymbol{\xi} - \eta \nabla_{\boldsymbol{\xi}} \mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\psi}|\mathcal{D}, \tilde{\boldsymbol{\theta}})$
  Update $\boldsymbol{\psi} := \boldsymbol{\psi} - \eta \nabla_{\boldsymbol{\psi}} \mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\psi}|\mathcal{D}, \tilde{\boldsymbol{\theta}})$
until converged

Figure 10.6 gives an example of this in practice. We generate a dataset from a mixture of 3 Gaussians in 2d, using $\boldsymbol{\mu}_1^* = [2, 0]$, $\boldsymbol{\mu}_2^* = [-2, -4]$, $\boldsymbol{\mu}_3^* = [-2, 4]$, precisions $\lambda_{dk}^* = 1$, and uniform mixing weights, $\boldsymbol{\pi}^* = [1/3, 1/3, 1/3]$. Figure 10.6a shows the training set of 3000 points. We fit this using SVI, with a batch size of 500, for 1000 epochs, using the Adam optimizer. Figure 10.6b shows the predictions of the fitted model. More precisely, it shows $p(\boldsymbol{x}|\overline{\boldsymbol{\theta}})$, where $\overline{\boldsymbol{\theta}} = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi})}[\boldsymbol{\theta}]$. Figure 10.6c shows a kernel density estimate fit to 10,000 samples from $q(\boldsymbol{\mu}_1|\boldsymbol{\psi})$. We see that the posterior mean is $\mathbb{E}[\boldsymbol{\mu}_1] \approx [-2, -4]$. Due to label switching unidentifiability, we see this matches $\boldsymbol{\mu}_2^*$ rather than $\boldsymbol{\mu}_1^*$.

10.2.1.6 Example: MLE for LVMs

In this section, we consider reparameterized SVI for computing the MLE for latent variable models (LVMs) with continuous latents, such as variational autoencoders (Section 21.2). Unlike Section 10.2.1.5, we cannot analytically marginalize out the local latents. Instead we will use amortized inference, as in Section 10.1.5, which means we learn an inference network (with parameters $\boldsymbol{\phi}$) to predict the local variational parameters $\boldsymbol{\psi}_n$ given input $\boldsymbol{x}_n$. If we sample a single example $\boldsymbol{x}_n$ from the dataset at each iteration, and a single latent variable $\boldsymbol{z}_n$ from the variational posterior, then we get the pseudocode in Algorithm 10.3.

Algorithm 10.3: Reparameterized amortized SVI for MLE of an LVM

Initialize $\boldsymbol{\theta}$, $\boldsymbol{\phi}$
repeat
  Sample $\boldsymbol{x}_n \sim p_{\mathcal{D}}$
  Sample $\boldsymbol{\epsilon}_n \sim q_0$
  Compute $\boldsymbol{z}_n = g(\boldsymbol{\phi}, \boldsymbol{x}_n, \boldsymbol{\epsilon}_n)$
  Compute $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}_n, \boldsymbol{z}_n) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n) + \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$
  Update $\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{x}_n, \boldsymbol{z}_n)$
  Update $\boldsymbol{\phi} := \boldsymbol{\phi} - \eta \nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{x}_n, \boldsymbol{z}_n)$
until converged

10.2.2 Automatic differentiation VI

To apply Gaussian VI, we need to transform constrained parameters (such as variance terms) to unconstrained form, so they live in $\mathbb{R}^D$. This technique can be used for any distribution for which we can define a bijection to $\mathbb{R}^D$. This approach is called **automatic differentiation variational inference** or **ADVI** [Kuc+16]. We give the details below.

10.2.2.1 Basic idea

Our goal is to approximate the posterior $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \Theta$ lives in some D-dimensional constrained parameter space. Let $T : \Theta \to \mathbb{R}^D$ be a bijective mapping that maps from the constrained space Θ to the unconstrained space $\mathbb{R}^D$. with inverse $T^{-1} : \mathbb{R}^D \to \Theta$. Let $\boldsymbol{u} = T(\boldsymbol{\theta})$ be the unconstrained latent variables. We will use a Gaussian variational approximation to the posterior for $\boldsymbol{u}$, i.e.,: $q_{\boldsymbol{\psi}}(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}_d, \boldsymbol{\Sigma})$, where $\boldsymbol{\psi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

By the change of variable formula Equation (2.257), we have

$$p(\boldsymbol{u}) = p(T^{-1}(\boldsymbol{u}))|\det(\mathbf{J}_{T^{-1}}(\boldsymbol{u}))| \tag{10.62}$$

where $\mathbf{J}_{T^{-1}}$ is the Jacobian of the inverse mapping $\boldsymbol{u} \to \boldsymbol{\theta}$. Hence the ELBO becomes

$$Ł(\boldsymbol{\psi}) = E_{\boldsymbol{u}\sim q_{\boldsymbol{\psi}}(\boldsymbol{u})}\left[\log p(\mathcal{D}|T^{-1}(\boldsymbol{u})) + \log p(T^{-1}(\boldsymbol{u})) + \log|\det(\mathbf{J}_{T^{-1}}(\boldsymbol{u}))|\right] + \mathbb{H}(\boldsymbol{\psi}) \tag{10.63}$$

This is a tractable objective, assuming the Jacobian is tractable, since the final entropy term is available in closed form, and we can use a Monte Carlo approximation of the expectation over $\boldsymbol{u}$.

Since the objective is stochastic, and reparamterizable, we can use SGD to optimize it. However, [Ing20] propose **deterministic ADVI**, in which the samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are held fixed during the optimization process. This is called the common random numbers trick (Section 11.6.1), and makes the objective a deterministic function; this allows for the use of more powerful second-order optimization methods, such as BFGS. (Of course, if the dataset is large, we might need to use minibatch subsampling, which reintroduces stochasticity.)

10.2.2.2 Example: ADVI for beta-binomial model

To illustrate ADVI, we consider the 1d beta-binomial model from Section 7.4.4. We want to approximate $p(\theta|\mathcal{D})$ using the prior $p(\theta) = \text{Beta}(\theta|a, b)$ and likelihood $p(\mathcal{D}|\theta) = \prod_i \text{Ber}(y_i|\theta)$, where the sufficient statistics are $N_1 = 10$, $N_0 = 1$, and the prior is uninformative, $a = b = 1$. We use the transformation $\theta = T^{-1}(z) = \sigma(z)$, and optimize the ELBO with SGD. The results of this method are shown in Figure 7.4 and show that the Gaussian fit is a good approximation, despite the skewed nature of the posterior.

10.2.2.3 Example: ADVI for GMMs

In this section, we use ADVI to approximate the posterior of the parameters of a mixture of Gaussians. The difference from the VBEM algorithm of Section 10.3.6 is that we use ADVI combined with a Gaussian variational posterior, rather than using a mean field approximation defined by a product of conjugate distributions.

To apply ADVI, we marginalize out the discrete local discrete latents $m_n \in \{1, \ldots, K\}$ analytically, so the likelihood has the form

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{y}_n|\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\Sigma}_k)) \right] \tag{10.64}$$

We use an uniformative Gaussian prior for the $\boldsymbol{\mu}_k$, a uniform LKJ prior for the $\mathbf{L}_k$, a log-normal prior for the $\boldsymbol{\sigma}_k$, and a uniform Dirichlet prior for the mixing weights $\boldsymbol{\pi}$. (See [Kuc+16, Fig 21] for a definition of the model in STAN syntax.) The posterior approximation for the unconstrained parameters is a block-diagonal gaussian. $q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{\psi}_{\boldsymbol{\mu}}, \boldsymbol{\psi}_{\boldsymbol{\Sigma}})$, where the unconstrained parameters are computed using suitable bijections (see code for details).

We apply this method to the Old Faithful dataset from Figure 10.12, using $K = 10$ mixture components. The results are shown in Figure 10.7. In the top left, we show the special case where we constrain the posterior to be a MAP estimate, by setting $\boldsymbol{\psi}_{\boldsymbol{\Sigma}} = \mathbf{0}$. We see that there is no sparsity in the posterior, since there is no Bayesian "Occam factor" from marginalizing out the parameters. In panels c–d, we show 3 samples from the posterior. We see that the Bayesian method strongly prefers just 2 mixture components, although there is a small amount of support for some other Gaussian components (shown by the faint ellipses).

10.2.2.4 More complex posteriors

We can combine ADVI with any of the improved posterior approximations that we discuss in Section 10.4 — such as Gaussian mixtures [Mor+21b] or normalizing flows [ASD20] — to create a high-quality, automatic approximate inference scheme.

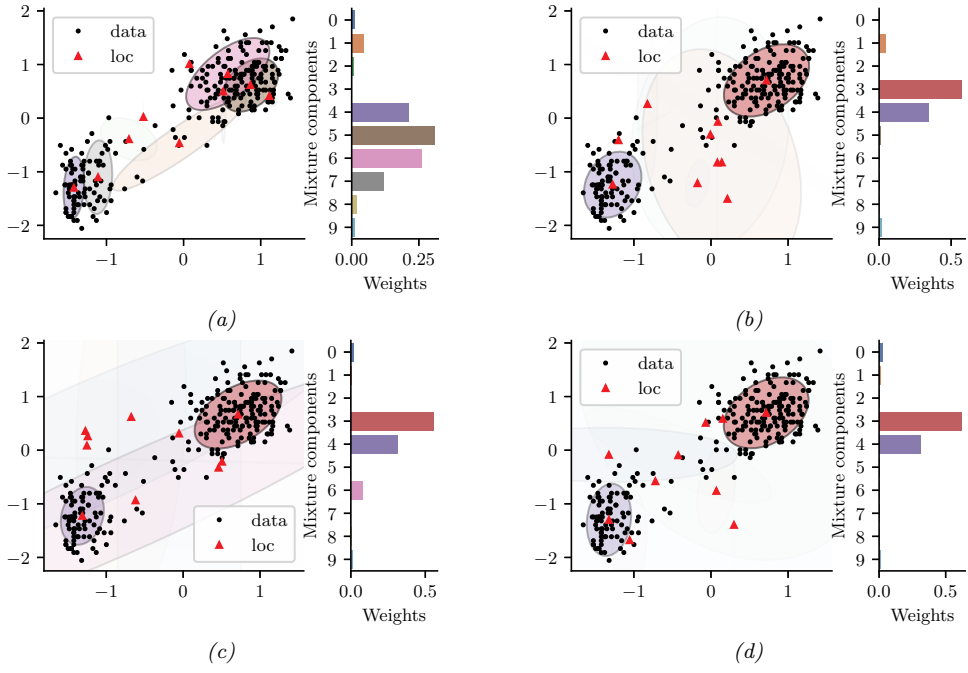


Figure 10.7: Posterior over the mixing weights (histogram) and the means and covariances of each Gaussian mixture component, using $K = 10$, when fitting the model to the Old Faithful dataset from Figure 10.12. (a) MAP approximation. (b-d) 3 samples from the Gaussian approximation. The intensity of the shading is proportional to the mixture weight. Generated by gmm_advi_bijax.ipynb.

10.2.3 Blackbox variational inference

In this section, we assume that we can evaluate $\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}) = \log p(\boldsymbol{z}, \boldsymbol{x}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z})$ pointwise, but we do not assume we can take gradients of this function. (For example, $\boldsymbol{z}$ may contain discrete variables.) We are thus treating the model as a "blackbox". Hence this approach is called **blackbox variational inference** or **BBVI** [RGB14; ASD20].

10.2.3.1 Estimating the gradient using REINFORCE

To estimate the gradient of the ELBO, we will use the **score function estimator**, also called the **REINFORCE** estimator (Section 6.3.4). In particular, suppose we write the ELBO as

$$Ł(\boldsymbol{\psi}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z})\right] = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log p(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{\psi})\right] \tag{10.65}$$

Then from Equation (6.58) we have

$$\nabla_{\boldsymbol{\psi}} Ł(\boldsymbol{\psi}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}) \nabla_{\boldsymbol{\psi}} \log q(\boldsymbol{z}|\boldsymbol{\psi})\right] \tag{10.66}$$

We can then compute a Monte Carlo approximation to this:

$$\widehat{\nabla_{\boldsymbol{\psi}} \mathrm{L}(\boldsymbol{\psi}_t)} = \frac{1}{S} \sum_{s=1}^{S} \tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}_s) \nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}_s)|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \tag{10.67}$$

We can pass this to any kind of gradient optimizer, such as SGD or Adam.

10.2.3.2 Reducing the variance using control variates

In practice, the variance of this estimator is quite large, so it is important to use methods such as **control variates** or **CV** (Section 6.3.4.1). To see how this works, consider the naive gradient estimator in Equation (10.67), which for the i'th component we can write as

$$\widehat{\nabla_{\boldsymbol{\psi}_i} \mathrm{L}(\boldsymbol{\psi}_t)}^{\text{naive}} = \frac{1}{S} \sum_{s=1}^{S} \tilde{g}_i(\boldsymbol{z}_s) \tag{10.68}$$

$$\tilde{g}_i(\boldsymbol{z}_s) = g_i(\boldsymbol{z}_s) \times \tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}_s) \tag{10.69}$$

$$g_i(\boldsymbol{z}_s) = \nabla_{\boldsymbol{\psi}_i} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}_s) \tag{10.70}$$

The control variate version of this can be obtained by replacing $\tilde{g}_i(\boldsymbol{z}_s)$ with

$$\tilde{g}_i^{cv}(\boldsymbol{z}) = \tilde{g}_i(\boldsymbol{z}) + c_i(\mathbb{E}\left[b_i(\boldsymbol{z})\right] - b_i(\boldsymbol{z})) \tag{10.71}$$

where $b_i(\boldsymbol{z})$ is a baseline function and c_i is some constant, to be specified below. A convenient baseline is the score function, $b_i(\boldsymbol{z}) = \nabla_{\boldsymbol{\psi}_i} \log q_{\boldsymbol{\psi}_i}(\boldsymbol{z}) = g_i(\boldsymbol{z})$, since this is correlated with $\tilde{g}_i(\boldsymbol{z})$, and has the property that $\mathbb{E}\left[b_i(\boldsymbol{z})\right] = \mathbf{0}$, since the expected value of the score function is zero, as we showed in Equation (3.44). Hence

$$\tilde{g}_i^{cv}(\boldsymbol{z}) = \tilde{g}_i(\boldsymbol{z}) - c_i g_i(\boldsymbol{z}) = g_i(\boldsymbol{z})(\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}) - c_i) \tag{10.72}$$

so the CV estimator is given by

$$\widehat{\nabla_{\boldsymbol{\psi}_i} \mathrm{L}(\boldsymbol{\psi}_t)}^{\text{cv}} = \frac{1}{S} \sum_{s=1}^{S} g_i(\boldsymbol{z}_s) \times (\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}_s) - c_i) \tag{10.73}$$

One can show that the optimal c_i that minimizes the variance of the CV estimator is

$$c_i = \frac{\mathrm{Cov}\left[g_i(\boldsymbol{z})\tilde{\mathcal{L}}(\boldsymbol{\psi}, \boldsymbol{z}), g_i(\boldsymbol{z})\right]}{\mathbb{V}\left[g_i(\boldsymbol{z})\right]} \tag{10.74}$$

For more details, see e.g., [TND21].

10.3 Coordinate ascent VI

A common approximation in variational inference is to assume that all the latent variables are independent, i.e.,

$$q_{\boldsymbol{\psi}}(\boldsymbol{z}) = \prod_{j=1}^{J} q_j(z_j) \tag{10.75}$$

where J is the number of hidden variables, and $q_j(z_j)$ is shorthand for $q_{\boldsymbol{\psi}_j}(z_j)$, where $\boldsymbol{\psi}_j$ are the variational parameters for the j'th distribution. This is called the **mean field** approximation.

From Equation (10.11), the ELBO becomes

$$Ł(\boldsymbol{\psi}) = \int q_{\boldsymbol{\psi}}(\boldsymbol{z}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} + \sum_{j=1}^{J} \mathbb{H}(q_j) \tag{10.76}$$

since the entropy of a product distribution is the sum of entropies of each component in the product. The first term also often decomposes according to the Markov properties of the graphical model. This allows us to use a coordinate ascent optimization scheme to estimate each $\boldsymbol{\psi}_j$, as we explain in Section 10.3.1. This is called **coordinate ascent variational inference** or **CAVI**, and is an alternative to gradient-based VI.

10.3.1 Derivation of CAVI algorithm

In this section, we derive the coordinate ascent variational inference (CAVI) procedure.

To derive the update equations, we initially assume there are just 3 discrete latent variables, to simplify notation. In this case the ELBO is given by

$$Ł(q_1, q_2, q_3) = \sum_{z_1} \sum_{z_2} \sum_{z_3} q_1(z_1) q_2(z_2) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) + \sum_{j=1}^{3} \mathbb{H}(q_j) \tag{10.77}$$

where we define $\tilde{p}(\boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$ for brevity. We will optimize this wrt each q_i, one at a time, keeping the others fixed.

Let us look at the objective for q_3:

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\sum_{z_1} \sum_{z_2} q_1(z_1) q_2(z_2) \log \tilde{p}(z_1, z_2, z_3) \right] + \mathbb{H}(q_3) + \text{const} \tag{10.78}$$

$$= \sum_{z_3} q_3(z_3) \left[g_3(z_3) - \log q_3(z_3) \right] + \text{const} \tag{10.79}$$

where

$$g_3(z_3) \triangleq \sum_{z_1} \sum_{z_2} q_1(z_1) q_2(z_2) \log \tilde{p}(z_1, z_2, z_3) = \mathbb{E}_{\boldsymbol{z}_{-3}} \left[\log \tilde{p}(z_1, z_2, z_3) \right] \tag{10.80}$$

where $\boldsymbol{z}_{-3} = (z_1, z_2)$ is all variables except z_3. Here $g_3(z_3)$ can be interpreted as an expected negative energy (log probability). We can convert this into an unnormalized probability distribution by defining

$$\tilde{f}_3(z_3) = \exp(g_3(z_3)) \tag{10.81}$$

which we can normalize to get

$$f_3(z_3) = \frac{\tilde{f}_3(z_3)}{\sum_{z_3'} \tilde{f}_3(z_3')} \propto \exp(g_3(z_3)) \tag{10.82}$$

Since $g_3(z_3) \propto \log f_3(z_3)$ we get

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\log f_3(z_3) - \log q_3(z_3)\right] + \text{const} = -D_{\mathbb{KL}}\left(q_3 \parallel f_3\right) + \text{const} \tag{10.83}$$

Since $D_{\mathbb{KL}}\left(q_3 \parallel f_3\right)$ achieves its minimal value of 0 when $q_3(z_3) = f_3(z_3)$ for all z_3, we see that $q_3^*(z_3) = f_3(z_3)$.

Now suppose that the joint distribution is defined by a Markov chain, where $z_1 \to z_2 \to z_3$, so $z_1 \perp z_3|z_2$. Hence $\log \tilde{p}(z_1, z_2, z_3) = \log \tilde{p}(z_2, z_3|z_1) + \log \tilde{p}(z_1)$, where the latter term is independent of $q_3(z_3)$. Thus the ELBO simplifies to

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\sum_{z_2} q_2(z_2) \log \tilde{p}(z_2, z_3)\right] + \mathbb{H}(q_3) + \text{const} \tag{10.84}$$

$$= \sum_{z_3} q_3(z_3) \left[\log f_3(z_3) - \log q_3(z_3)\right] + \text{const} \tag{10.85}$$

where

$$f_3(z_3) \propto \exp\left[\sum_{z_2} q_2(z_2) \log \tilde{p}(z_2, z_3)\right] = \exp\left[\mathbb{E}_{\boldsymbol{z}_{\text{mb}_3}}\left[\log \tilde{p}(z_2, z_3)\right]\right] \tag{10.86}$$

where $\boldsymbol{z}_{\text{mb}_3} = z_2$ is the Markov blanket (Section 4.2.4.3) of z_3. As before, the optimal variational distribution is given by $q_3(z_3) = f_3(z_3)$.

In general, when we have J groups of variables, the optimal variational distribution for the j'th group is given by

$$q_j(\boldsymbol{z}_j) \propto \exp\left[\mathbb{E}_{\boldsymbol{z}_{\text{mb}_j}}\left[\log \tilde{p}(\boldsymbol{z}_j, \boldsymbol{z}_{\text{mb}_j})\right]\right] \tag{10.87}$$

(Compare to the equation for Gibbs sampling in Equation (12.19).) The CAVI method simply computes q_j for each dimension j in turn, in an iterative fashion (see Algorithm 10.4). Convergence is guaranteed since the bound is concave wrt each of the factors q_i [Bis06, p. 466].

Algorithm 10.4: Coordinate ascent variational inference (CAVI).

```
Initialize q_j(z_j) for j = 1 : J
foreach t = 1 : T do
    foreach j = 1 : J do
        Compute g_j(z_j) = E_{z_mb_i}[log p̃(z_i, z_mb_i)]
        Compute q_j(z_j) ∝ exp(g_j(z_j))
```

Note that the functional form of the q_i distributions does not need to be specified in advance, but will be determined by the form of the log joint. This is therefore called **free-form VI**, as opposed to fixed-form, where we explicitly choose a convenient distributional type for q (we discuss fixed-form VI in Section 10.2). We give some examples below that will make this clearer.

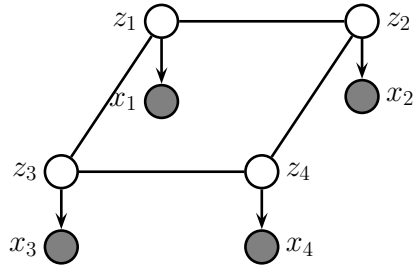


Figure 10.8: A grid-structured MRF with hidden nodes z_i and local evidence nodes x_i. The prior $p(\boldsymbol{z})$ is an undirected Ising model, and the likelihood $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(x_i|z_i)$ is a directed fully factored model.

10.3.2 Example: CAVI for the Ising model

In this section, we apply CAVI to perform mean field inference in an Ising model (Section 4.3.2.1), which is a kind of Markov random field defined on binary random variables, $z_i \in \{-1, +1\}$, arranged in a 2d grid.

Originally Ising models were developed as models of atomic spins for magnetic materials, although we will apply them to an image denoising problem. Specifically, let z_i be the hidden value of pixel i, and $x_i \in \mathbb{R}$ be the observed noisy value. See Figure 10.8 for the graphical model.

Let $L_i(z_i) \triangleq \log p(x_i|z_i)$ be the log likelihood for the i'th pixel (aka the **local evidence** for node i in the graphical model). The overall likelihood has the form

$$p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(x_i|z_i) = \exp(\sum_i L_i(z_i)) \tag{10.88}$$

Our goal is to approximate the posterior $p(\boldsymbol{z}|\boldsymbol{x})$. We will use an Ising model for the prior:

$$p(\boldsymbol{z}) = \frac{1}{Z_0} \exp(-\mathcal{E}_0(\boldsymbol{z})) \tag{10.89}$$

$$\mathcal{E}_0(\boldsymbol{z}) = -\sum_{i \sim j} W_{ij} z_i z_j \tag{10.90}$$

where we sum over each $i - j$ edge. Therefore the posterior has the form

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp(-\mathcal{E}(\boldsymbol{z})) \tag{10.91}$$

$$\mathcal{E}(\boldsymbol{z}) = \mathcal{E}_0(\boldsymbol{z}) - \sum_i L_i(z_i) \tag{10.92}$$

We will now make the following fully factored approximation:

$$q(\boldsymbol{z}) = \prod_i q_i(z_i) = \prod_i \text{Ber}(z_i|\mu_i) \tag{10.93}$$

where $\mu_i = \mathbb{E}_{q_i}[z_i]$ is the mean value of node i. To derive the update for the variational parameter μ_i, we first compute the unnormalized log joint, $\log \tilde{p}(\boldsymbol{z}) = -\mathcal{E}(\boldsymbol{z})$, dropping terms that do not involve

z_i:

$$\log \tilde{p}(\boldsymbol{z}) = z_i \sum_{j \in \text{nbr}_i} W_{ij} z_j + L_i(z_i) + \text{const} \tag{10.94}$$

This only depends on the states of the neighboring nodes. Hence

$$q_i(z_i) \propto \exp(\mathbb{E}_{q_{-i}(\boldsymbol{z})}[\log \tilde{p}(\boldsymbol{z})]) = \exp\left(z_i \sum_{j \in \text{nbr}_i} W_{ij}\mu_j + L_i(z_i)\right) \tag{10.95}$$

where $q_{-i}(\boldsymbol{z}) = \prod_{j \neq i} q(z_j)$. Thus we replace the states of the neighbors by their average values. (Note that this replaces binary variables with continuous ones.)

We now simplify this expression. Let $m_i = \sum_{j \in \text{nbr}_i} W_{ij}\mu_j$ be the mean field influence on node i. Also, let $L_i^+ \triangleq L_i(+1)$ and $L_i^- \triangleq L_i(-1)$. The approximate marginal posterior is given by

$$q_i(z_i = 1) = \frac{e^{m_i + L_i^+}}{e^{m_i + L_i^+} + e^{-m_i + L_i^-}} = \frac{1}{1 + e^{-2m_i + L_i^- - L_i^+}} = \sigma(2a_i) \tag{10.96}$$

$$a_i \triangleq m_i + 0.5(L_i^+ - L_i^-) \tag{10.97}$$

Similarly, we have $q_i(z_i = -1) = \sigma(-2a_i)$. From this we can compute the new mean for site i:

$$\mu_i = \mathbb{E}_{q_i}[z_i] = q_i(z_i = +1) \cdot (+1) + q_i(z_i = -1) \cdot (-1) \tag{10.98}$$

$$= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} = \tanh(a_i) \tag{10.99}$$

We can turn the above equations into a fixed point algorithm by writing

$$\mu_i^t = \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{10.100}$$

Following [MWJ99], we can use **damped updates** of the following form to improve convergence:

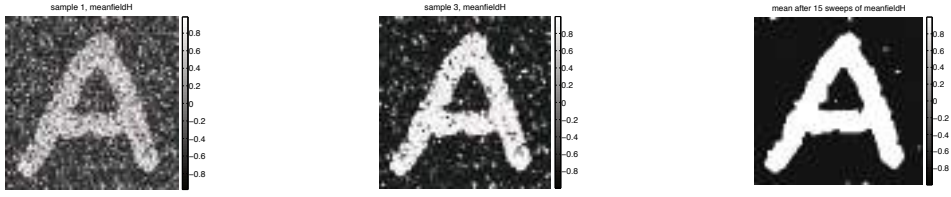
$$\mu_i^t = (1 - \lambda)\mu_i^{t-1} + \lambda \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{10.101}$$

for $0 < \lambda < 1$. We can update all the nodes in parallel, or update them asynchronously.

Figure 10.9 shows the method in action, applied to a 2d Ising model with homogeneous attractive potentials, $W_{ij} = 1$. We use parallel updates with a damping factor of $\lambda = 0.5$. (If we don't use damping, we tend to get "checkerboard" artifacts.)

10.3.3 Variational Bayes

In Bayesian modeling, we treat the parameters $\boldsymbol{\theta}$ as latent variables. Thus our goal is to approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta})$. Applying VI to this problem is called **variational Bayes** [Att00].



(a) (b) (c)

Figure 10.9: Example of image denoising using mean field (with parallel updates and a damping factor of 0.5). We use an Ising prior with $W_{ij} = 1$ and a Gaussian noise model with $\sigma = 2$. We show the results after 1, 3 and 15 iterations across the image. Compare to Figure 12.3, which shows the results of using Gibbs sampling. Generated by ising_image_denoise_demo.ipynb.

In this section, we assume there are no latent variables except for the shared global parameters, so the model has the form

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathcal{D}_n|\boldsymbol{\theta}) \tag{10.102}$$

These conditional independencies are illustrated in Figure 10.5a.

We will fit the variational posterior by maximizing the ELBO

$$Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})}[\log p(\boldsymbol{\theta}, \mathcal{D})] + \mathbb{H}(q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})) \tag{10.103}$$

We will assume the variational posterior factorizes over the parameters:

$$q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) = \prod_j q(\boldsymbol{\theta}_j|\boldsymbol{\psi}_{\theta_j}) \tag{10.104}$$

We can then update each $\boldsymbol{\psi}_{\theta_j}$ using CAVI (Section 10.3.1).

10.3.4 Example: VB for a univariate Gaussian

Consider inferring the parameters of a 1d Gaussian. The likelihood is given by $p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \lambda$ where μ is the mean and λ is the precision. Suppose we use a conjugate prior of the form

$$p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, (\kappa_0\lambda)^{-1})\text{Ga}(\lambda|a_0, b_0) \tag{10.105}$$

It is possible to derive the posterior $p(\mu, \lambda|\mathcal{D})$ for this model exactly, as shown in Section 3.4.3.3. However, here we use the VB method with the following factored approximate posterior:

$$q(\mu, \lambda) = q(\mu|\boldsymbol{\psi}_\mu)q(\lambda|\boldsymbol{\psi}_\lambda) \tag{10.106}$$

We do not need to specify the forms for the distributions $q(\mu|\boldsymbol{\psi}_\mu)$ and $q(\lambda|\boldsymbol{\psi}_\lambda)$; the optimal forms will "fall out" automatically during the derivation (and conveniently, they turn out to be Gaussian and gamma respectively). Our presentation follows [Mac03, p429].

10.3.4.1 Target distribution

The unnormalized log posterior has the form

$$\log \tilde{p}(\mu, \lambda) = \log p(\mu, \lambda, \mathcal{D}) = \log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda) \tag{10.107}$$

$$= \frac{N}{2}\log\lambda - \frac{\lambda}{2}\sum_{n=1}^{N}(x_n - \mu)^2 - \frac{\kappa_0\lambda}{2}(\mu - \mu_0)^2$$

$$+ \frac{1}{2}\log(\kappa_0\lambda) + (a_0 - 1)\log\lambda - b_0\lambda + \text{const} \tag{10.108}$$

10.3.4.2 Updating $q(\mu|\boldsymbol{\psi}_\mu)$

The optimal form for $q(\mu|\boldsymbol{\psi}_\mu)$ is obtained by averaging over λ:

$$\log q(\mu|\boldsymbol{\psi}_\mu) = \mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)}\left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda)\right] + \text{const} \tag{10.109}$$

$$= -\frac{\mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)}[\lambda]}{2}\left\{\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^{N}(x_n - \mu)^2\right\} + \text{const} \tag{10.110}$$

By completing the square one can show that $q(\mu|\boldsymbol{\psi}_\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, where

$$\mu_N = \frac{\kappa_0\mu_0 + N\overline{x}}{\kappa_0 + N},\ \kappa_N = (\kappa_0 + N)\mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)}[\lambda] \tag{10.111}$$

At this stage we don't know what $q(\lambda|\boldsymbol{\psi}_\lambda)$ is, and hence we cannot compute $\mathbb{E}[\lambda]$, but we will derive this below.

10.3.4.3 Updating $q(\lambda|\boldsymbol{\psi}_\lambda)$

The optimal form for $q(\lambda|\boldsymbol{\psi}_\lambda)$ is given by

$$\log q(\lambda|\boldsymbol{\psi}_\lambda) = \mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)}\left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda)\right] + \text{const} \tag{10.112}$$

$$= (a_0 - 1)\log\lambda - b_0\lambda + \frac{1}{2}\log\lambda + \frac{N}{2}\log\lambda$$

$$- \frac{\lambda}{2}\mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)}\left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^{N}(x_n - \mu)^2\right] + \text{const} \tag{10.113}$$

We recognize this as the log of a gamma distribution, hence $q(\lambda|\boldsymbol{\psi}_\lambda) = \text{Ga}(\lambda|a_N, b_N)$, where

$$a_N = a_0 + \frac{N+1}{2} \tag{10.114}$$

$$b_N = b_0 + \frac{1}{2}\mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)}\left[\kappa_0(\mu - \mu_0)^2 + \sum_{n=1}^{N}(x_n - \mu)^2\right] \tag{10.115}$$

10.3.4.4 Computing the expectations

To implement the updates, we have to specify how to compute the various expectations. Since $q(\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, we have

$$\mathbb{E}_{q(\mu)}[\mu] = \mu_N \tag{10.116}$$

$$\mathbb{E}_{q(\mu)}\left[\mu^2\right] = \frac{1}{\kappa_N} + \mu_N^2 \tag{10.117}$$

Since $q(\lambda) = \text{Ga}(\lambda|a_N, b_N)$, we have

$$\mathbb{E}_{q(\lambda)}[\lambda] = \frac{a_N}{b_N} \tag{10.118}$$

We can now give explicit forms for the update equations. For $q(\mu)$ we have

$$\mu_N = \frac{\kappa_0\mu_0 + N\overline{x}}{\kappa_0 + N} \tag{10.119}$$

$$\kappa_N = (\kappa_0 + N)\frac{a_N}{b_N} \tag{10.120}$$

and for $q(\lambda)$ we have

$$a_N = a_0 + \frac{N+1}{2} \tag{10.121}$$

$$b_N = b_0 + \frac{1}{2}\kappa_0(\mathbb{E}\left[\mu^2\right] + \mu_0^2 - 2\mathbb{E}[\mu]\,\mu_0) + \frac{1}{2}\sum_{n=1}^{N}\left(x_n^2 + \mathbb{E}\left[\mu^2\right] - 2\mathbb{E}[\mu]\,x_n\right) \tag{10.122}$$

We see that μ_N and a_N are in fact fixed constants, and only κ_N and b_N need to be updated iteratively. (In fact, one can solve for the fixed points of κ_N and b_N analytically, but we don't do this here in order to illustrate the iterative updating scheme.)

10.3.4.5 Illustration

Figure 10.10 gives an example of this method in action. The green contours represent the exact posterior, which is Gaussian-gamma. The dotted red contours represent the variational approximation over several iterations. We see that the final approximation is reasonably close to the exact solution. However, it is more "compact" than the true distribution. It is often the case that mean field inference underestimates the posterior uncertainty, for reasons explained in Section 5.1.4.1.

10.3.4.6 Lower bound

In VB, we maximize a lower bound on the log marginal likelihood:

$$Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) \leq \log p(\mathcal{D}) = \log \iint p(\mathcal{D}|\mu,\lambda)p(\mu,\lambda)d\mu d\lambda \tag{10.123}$$

It is very useful to compute the lower bound itself, for three reasons. First, it can be used to assess convergence of the algorithm. Second, it can be used to assess the correctness of one's code: as with

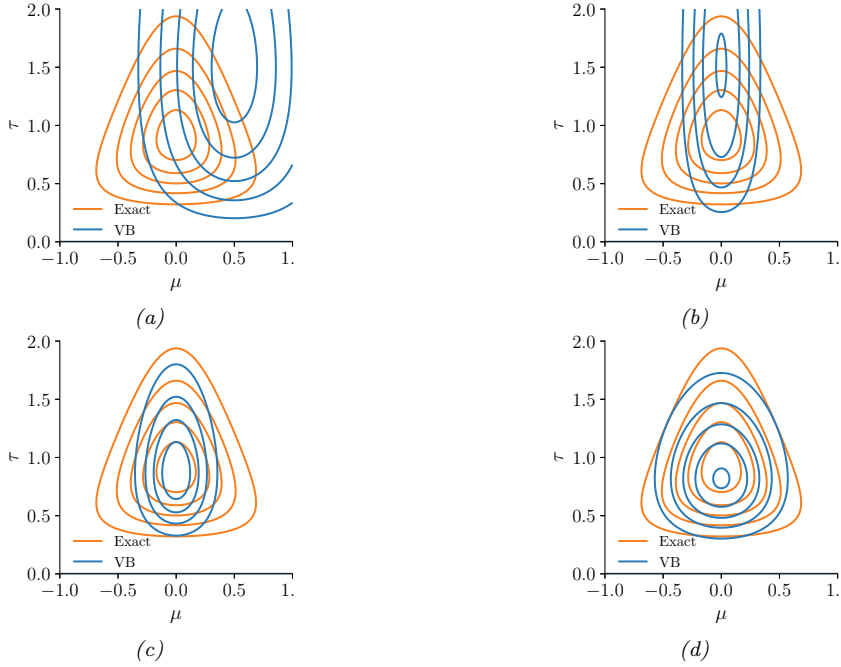


Figure 10.10: Factored variational approximation (orange) to the Gaussian-gamma distribution (blue). (a) Initial guess. (b) After updating $q(\mu|\psi_\mu)$. (c) After updating $q(\lambda|\psi_\lambda)$. (d) At convergence (after 5 iterations). Adapted from Fig. 10.4 of [Bis06]. Generated by unigauss_vb_demo.ipynb.

EM, if we use CAVI to optimize the objective, the bound should increase monotonically at each iteration, otherwise there must be a bug. Third, the bound can be used as an approximation to the marginal likelihood, which can be used for Bayesian model selection or empirical Bayes (see Section 10.1.3). In the case of the current model, one can show that the lower bound has the following form:

$$Ł = \text{const} + \frac{1}{2}\ln\frac{1}{\kappa_N} + \ln\Gamma(a_N) - a_N \ln b_N \tag{10.124}$$

10.3.5 Variational Bayes EM

In Bayesian latent variable models, we have two forms of hidden variables: local (or per example) hidden variables $\boldsymbol{z}_n$, and global (shared) hidden variables $\boldsymbol{\theta}$, which represent the parameters of the model. See Figure 10.5b for an illustration. (Note that the parameters, which are fixed in number, are sometimes called **intrinsic variables**, whereas the local hidden variables are called **extrinsic variables**.) If $\boldsymbol{h} = (\boldsymbol{\theta}, \boldsymbol{z}_{1:N})$ represents all the hidden variables, then the joint distribution is given by

$$p(\boldsymbol{h}, \mathcal{D}) = p(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\boldsymbol{z}_n|\boldsymbol{\theta}) p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}) \tag{10.125}$$

We will make the following mean field assumption:

$$q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}}) = q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) \prod_{n=1}^{N} q(\boldsymbol{z}_n|\boldsymbol{\psi}_n) \tag{10.126}$$

where $\boldsymbol{\psi} = (\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}})$.

We will use VI to maximize the ELBO:

$$Ł(\boldsymbol{\psi}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}})} \left[\log p(\boldsymbol{z}_{1:N}, \boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N})\right] \tag{10.127}$$

If we use the mean field assumption, then we can apply the CAVI approach to optimize each set of variational parameters. In particular, we can alternate between optimizing the $q_n(\boldsymbol{z}_n)$ in parallel, independently of each other, with $q(\boldsymbol{\theta})$ held fixed, and then optimizing $q(\boldsymbol{\theta})$ with the q_n held fixed. This is known as **variational Bayes EM** [BG06]. It is similar to regular EM, except in the E step, we infer an approximate posterior for $\boldsymbol{z}_n$ averaging out the parameters (instead of plugging in a point estimate), and in the M step, we update the parameter posterior parameters using the expected sufficient statistics.

Now suppose we approximate $q(\boldsymbol{\theta})$ by a delta function, $q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$. The Bayesian LVM ELBO objective from Equation (10.127) simplifies to the "LVM ELBO":

$$Ł(\boldsymbol{\theta}, \boldsymbol{\psi}_{1:N}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N})} \left[\log p(\boldsymbol{\theta}, \mathcal{D}, \boldsymbol{z}_{1:N}) - \log q(\boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N})\right] \tag{10.128}$$

We can optimize this using the **variational EM** algorithm, which is a CAVI algorithm which updates the $\boldsymbol{\psi}_n$ in parallel in the variational E step, and then updates $\boldsymbol{\theta}$ in the M step.

VEM is simpler than VBEM since in the variational E step, we compute $q(\boldsymbol{z}_n|\boldsymbol{x}_n, \hat{\boldsymbol{\theta}})$, instead of $\mathbb{E}_{\boldsymbol{\theta}}[q(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{\theta})]$; that is, we plug in a point estimate of the model parameters, rather than averaging over the parameters. For more details on VEM, see Section 10.1.3.

10.3.6 Example: VBEM for a GMM

Consider a standard Gaussian mixture model (GMM):

$$p(\boldsymbol{z}, \boldsymbol{x}|\boldsymbol{\theta}) = \prod_n \prod_k \pi_k^{z_{nk}} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}} \tag{10.129}$$

where $z_{nk} = 1$ if datapoint n belongs to cluster k, and $z_{nk} = 0$ otherwise. Our goal is to approximate the posterior $p(\boldsymbol{z}, \boldsymbol{\theta}|\boldsymbol{x})$ under the following conjugate prior

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\, \breve{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k|\, \breve{\boldsymbol{m}}, (\breve{\kappa}\, \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k|\, \breve{\mathbf{L}}, \breve{\nu}) \tag{10.130}$$

where $\boldsymbol{\Lambda}_k$ is the precision matrix for cluster k. For the mixing weights, we usually use a symmetric prior, $\breve{\boldsymbol{\alpha}} = \alpha_0 \mathbf{1}$.

The exact posterior $p(\boldsymbol{z}, \boldsymbol{\theta}|\mathcal{D})$ is a mixture of K^N distributions, corresponding to all possible labelings $\boldsymbol{z}$, which is intractable to compute. In this section, we derive a VBEM algorithm, which will approximate the posterior around a local mode. We follow the presentation of [Bis06, Sec 10.2]. (See also Section 10.2.1.5 and Section 10.2.2.3, where we discuss variational approximations based on stochastic gradient descent, which can scale better to large datasets compared to VBEM.)

10.3.6.1 The variational posterior

We will use the standard mean field approximation to the posterior: $q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}) = q(\boldsymbol{\theta}) \prod_n q_n(\boldsymbol{z}_n)$. At this stage we have not specified the forms of the q functions; these will be determined by the form of the likelihood and prior. Below we will show that the optimal forms are as follows:

$$q_n(z_n) = \text{Cat}(\boldsymbol{z}_n|\boldsymbol{r}_n) \tag{10.131}$$

$$q(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\,\widehat{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k|\,\widehat{\boldsymbol{m}}_k, (\widehat{\kappa}_k\,\boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k|\,\widehat{\mathbf{L}}_k, \widehat{\nu}_k) \tag{10.132}$$

where $\boldsymbol{r}_n$ are the posterior responsibilities, and the parameters with hats on them are the hyperparameters from the prior updated with data.

10.3.6.2 Derivation of $q(\boldsymbol{\theta})$ (variational M step)

Using the mean field recipe in Algorithm 10.4, we write down the log joint, and take expectations over all variables except $\boldsymbol{\theta}$, so we average out the $\boldsymbol{z}_n$ wrt $q(\boldsymbol{z}_n) = \text{Cat}(z_n|\boldsymbol{r}_n)$:

$$\log q(\boldsymbol{\theta}) = \underbrace{\log p(\boldsymbol{\pi}) + \sum_n \mathbb{E}_{q(z_n)}\left[\log p(\boldsymbol{z}_n|\boldsymbol{\pi})\right]}_{L_{\boldsymbol{\pi}}} + \sum_k \left[\underbrace{\log p(\mu_k, \boldsymbol{\Lambda}_k) \sum_n \mathbb{E}_{q(z_n)}\left[z_{nk}\right] \log \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})}_{L_{\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k}} \right] + \text{const} \tag{10.133}$$

Since the expected log joint factorizes into a term involving $\boldsymbol{\pi}$ and terms involving $(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$, we see that the variational posterior also factorizes into the form

$$q(\boldsymbol{\theta}) = q(\boldsymbol{\pi}) \prod_k q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \tag{10.134}$$

For the $\boldsymbol{\pi}$ term, we have

$$\log q(\boldsymbol{\pi}) = (\alpha_0 - 1) \sum_k \log \pi_k + \sum_k \sum_n r_{nk} \log \pi_k + \text{const} \tag{10.135}$$

Exponentiating, we recognize this as a Dirichlet distribution:

$$q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi}|\,\widehat{\boldsymbol{\alpha}}) \tag{10.136}$$

$$\widehat{\alpha}_k = \alpha_0 + N_k \tag{10.137}$$

$$N_k = \sum_n r_{nk} \tag{10.138}$$

For the $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$ terms, we have

$$q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}(\boldsymbol{\mu}_k | \, \widehat{\boldsymbol{m}}_k, (\widehat{\kappa}_k \, \boldsymbol{\Lambda}_k)^{-1}) \mathrm{Wi}(\boldsymbol{\Lambda}_k | \, \widehat{\mathbf{L}}_k, \widehat{\nu}_k) \tag{10.139}$$

$$\widehat{\kappa}_k = \breve{\kappa} + N_k \tag{10.140}$$

$$\widehat{\boldsymbol{m}}_k = (\breve{\kappa} \, \breve{\boldsymbol{m}} + N_k \overline{\boldsymbol{x}}_k) / \, \widehat{\kappa}_k \tag{10.141}$$

$$\widehat{\mathbf{L}}_k^{-1} = \breve{\mathbf{L}}^{-1} + N_k \mathbf{S}_k + \frac{\breve{\kappa} \, N_k}{\breve{\kappa} + N_k} (\overline{\boldsymbol{x}}_k - \breve{\boldsymbol{m}})(\overline{\boldsymbol{x}}_k - \breve{\boldsymbol{m}})^{\mathsf{T}} \tag{10.142}$$

$$\widehat{\nu}_k = \breve{\nu} + N_k \tag{10.143}$$

$$\overline{\boldsymbol{x}}_k = \frac{1}{N_k} \sum_n r_{nk} \boldsymbol{x}_n \tag{10.144}$$

$$\mathbf{S}_k = \frac{1}{N_k} \sum_n r_{nk} (\boldsymbol{x}_n - \overline{\boldsymbol{x}}_k)(\boldsymbol{x}_n - \overline{\boldsymbol{x}}_k)^{\mathsf{T}} \tag{10.145}$$

This is very similar to the M step for MAP estimation for GMMs, except here we are computing the parameters of the posterior for $\boldsymbol{\theta}$ rather than a point estimate $\hat{\boldsymbol{\theta}}$.

10.3.6.3 Derivation of $q(\boldsymbol{z})$ (variational E step)

The variational E step is more interesting, since it is quite different from the E step in regular EM, because we need to average over the parameters, rather than condition on them. In particular, we have

$$\log q(\boldsymbol{z}) = \sum_n \sum_k z_{nk} \left(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k] + \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] - \frac{D}{2} \log(2\pi) \right.$$
$$\left. - \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} \left[(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}} \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right] \right) + \text{const} \tag{10.146}$$

Using the fact that $q(\boldsymbol{\pi}) = \mathrm{Dir}(\boldsymbol{\pi} | \, \widehat{\boldsymbol{\alpha}})$, one can show that

$$\exp(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k]) = \frac{\exp(\psi(\widehat{\alpha}_k))}{\exp(\psi(\sum_{k'} \widehat{\alpha}_{k'}))} \triangleq \tilde{\pi}_k \tag{10.147}$$

where ψ is the **digamma function**:

$$\psi(x) = \frac{d}{dx} \log \Gamma(x) \tag{10.148}$$

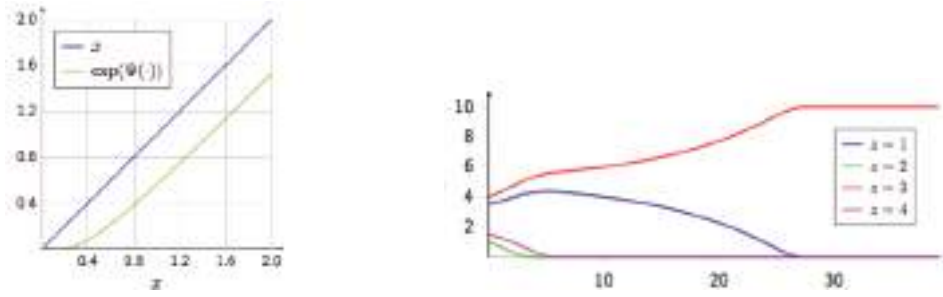
This takes care of the first term.

For the second term, one can show

$$\mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] = \sum_{j=1}^{D} \psi \left(\frac{\widehat{\nu}_k + 1 - j}{2} \right) + D \log 2 + \log | \, \widehat{\mathbf{L}}_k \, | \tag{10.149}$$

Finally, for the expected value of the quadratic form, one can show

$$\mathbb{E}_{q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)} \left[(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}} \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right] = D \, \widehat{\kappa}_k^{-1} + \widehat{\nu}_k \, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^{\mathsf{T}} \, \widehat{\mathbf{L}}_k \, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k) \triangleq \tilde{\Lambda}_k \tag{10.150}$$



(a) (b)

Figure 10.11: (a) We plot $\exp(\psi(x))$ *vs* x*. We see that this function performs a form of shrinkage, so that small values get set to zero. (b) We plot* N_k *vs time for 4 different states (*z *values), starting from random initial values. We perform a series of VBEM updates, ignoring the likelihood term. We see that states that initially had higher counts get reinforced, and sparsely populated states get killed off. From [LK07]. Used with kind permission of Percy Liang.*

Thus we get that the posterior responsibility of cluster k for datapoint n is

$$r_{nk} \propto \tilde{\pi}_k \tilde{\Lambda}_k^{\frac{1}{2}} \exp\left(-\frac{D}{2\,\widehat{\kappa}_k} - \frac{\widehat{\nu}_k}{2}(\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^{\mathsf{T}}\, \widehat{\boldsymbol{\Lambda}}_k\, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)\right) \tag{10.151}$$

Compare this to the expression used in regular EM:

$$r_{nk}^{EM} \propto \hat{\pi}_k |\hat{\boldsymbol{\Lambda}}_k|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_k)^{\mathsf{T}} \hat{\boldsymbol{\Lambda}}_k (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_k)\right) \tag{10.152}$$

where $\hat{\pi}_k$ is the MAP estimate for π_k. The significance of this difference is discussed in Section 10.3.6.4.

10.3.6.4 Automatic sparsity inducing effects of VBEM

In regular EM, the E step has the form given in Equation (10.152), whereas in VBEM, the E step has the form given in Equation (10.151). Although they look similar, they differ in an important way. To understand this, let us ignore the likelihood term, and just focus on the prior. From Equation (10.147) we have

$$r_{nk}^{VB} = \tilde{\pi}_k = \frac{\exp(\psi(\widehat{\alpha}_k))}{\exp(\psi(\sum_{k'} \widehat{\alpha}_{k'}))} \tag{10.153}$$

And from the usual EM MAP estimation equations for GMM mixing weights (see e.g., [Mur22, Sec 8.7.3.4]) we have

$$r_{nk}^{EM} = \hat{\pi}_k = \frac{\widehat{\alpha}_k - 1}{\sum_{k'}(\widehat{\alpha}_{k'} - 1)} \tag{10.154}$$

where $\widehat{\alpha}_k = \alpha_0 + N_k$, and $N_k = \sum_n r_{nk}$ is the expected number of assignments to cluster k.

We know from Figure 2.6 that using $\alpha_0 \ll 1$ causes $\boldsymbol{\pi}$ to be sparse, which will encourage $\boldsymbol{r}_n$ to be sparse, which will "kill off" unnecessary mixture components (i.e., ones for which $N_k \ll N$, meaning very few datapoints are assigned to cluster k). To encourage this sparsity promoting effect, let us set $\alpha_0 = 0$. In this case, the updated parameters for the mixture weights are given by the following:

$$\tilde{\pi}_k = \frac{\exp(\psi(N_k))}{\exp(\psi(\sum_{k'} N_{k'}))} \tag{10.155}$$

$$\hat{\pi}_k = \frac{N_k - 1}{\sum_{k'}(N_{k'} - 1)} \tag{10.156}$$

Now consider a cluster which has no assigned data, so $N_k = 0$. In regular EM, $\hat{\pi}_k$ might end up negative, as pointed out in [FJ02]. (This will not occur if we use maximum likelihood training, which corresponds to $\alpha_0 = 1$, but this will not induce any sparsity, either.) This problem does not arise in VBEM, since we use the digamma function, which is always positive, as shown in Figure 10.11(a).

More interestingly, let us consider the effect of these updates on clusters that have unequal, but non-zero, number of assignments. Suppose we start with a random assignment of counts to 4 clusters, and iterate the VBEM algorithm, ignoring the contribution from the likelihood for simplicity. Figure 10.11(b) shows how the counts N_k evolve over time. We notice that clusters that started out with small counts end up with zero counts, and clusters that started out with large counts end up with even larger counts. In other words, the initially popular clusters get more and more members. This is called the **rich get richer** phenomenon; we will encounter it again in Supplementary Section 31.2, when we discuss Dirichlet process mixture models.

The reason for this effect is shown in Figure 10.11(a): we see that $\exp(\psi(N_k)) < N_k$, and is zero if N_k is sufficiently small, similar to the soft-thresholding behavior induced by ℓ_1-regularization (see Section 15.2.6). Importantly, this effect of reducing N_k is greater on clusters with small counts.

We now demonstrate this automatic pruning method on a real example. We fit a mixture of 6 Gaussians to the Old Faithful dataset, using $\alpha_0 = 0.001$. Since the data only really "needs" 2 clusters, the remaining 4 get "killed off", as shown in Figure 10.12. In Figure 10.13, we plot the initial and final values of α_k; we see that $\widehat{\alpha}_k = 0$ for all but two of the components k.

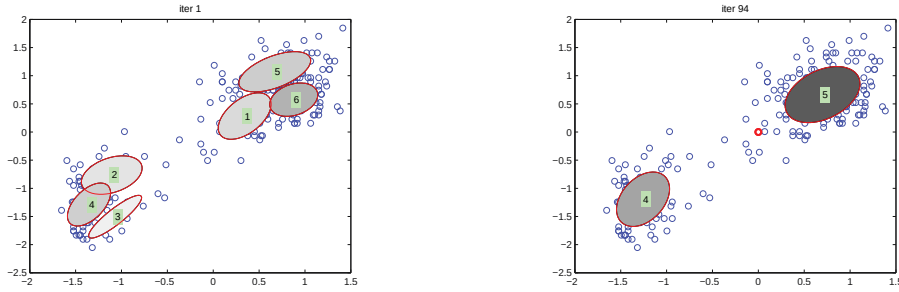
Thus we see that VBEM for GMMs with a sparse Dirichlet prior provides an efficient way to choose the number of clusters. Similar techniques can be used to choose the number of states in an HMM and other latent variable models. However, this **variational pruning effect** (also called **posterior collapse**), is not always desirable, since it can cause the model to "ignore" the latent variables $\boldsymbol{z}$ if the likelihood function $p(\boldsymbol{x}|\boldsymbol{z})$ is sufficiently powerful. We discuss this more in Section 21.4.

10.3.6.5 Lower bound on the marginal likelihood

The VBEM algorithm is maximizing the following lower bound

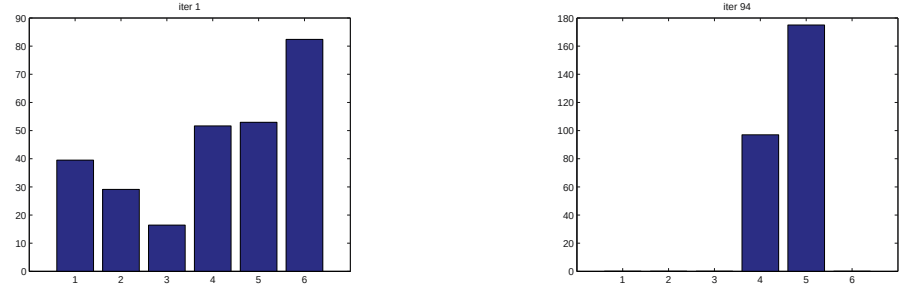
$$\mathcal{L} = \sum_{\boldsymbol{z}} \int d\boldsymbol{\theta}\, q(\boldsymbol{z}, \boldsymbol{\theta}) \log \frac{p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta})}{q(\boldsymbol{z}, \boldsymbol{\theta})} \leq \log p(\boldsymbol{x}) \tag{10.157}$$

This quantity increases monotonically with each iteration, as shown in Figure 10.14.



(a) (b)

Figure 10.12: We visualize the posterior mean parameters at various stages of the VBEM algorithm applied to a mixture of Gaussians model on the Old Faithful data. Shading intensity is proportional to the mixing weight. We initialize with K-means and use $\alpha_0 = 0.001$ as the Dirichlet hyper-parameter. (The red dot on the right panel represents all the unused mixture components, which collapse to the prior at 0.) Adapted from Figure 10.6 of [Bis06]. Generated by gmm_vb_em.ipynb.



(a) (b)

Figure 10.13: We visualize the posterior values of $\boldsymbol{\alpha}_k$ for the model in Figure 10.12 after the first and last iteration of the algorithm. We see that unnecessary components get "killed off". (Interestingly, the initially large cluster 6 gets "replaced" by cluster 5.) Generated by gmm_vb_em.ipynb.

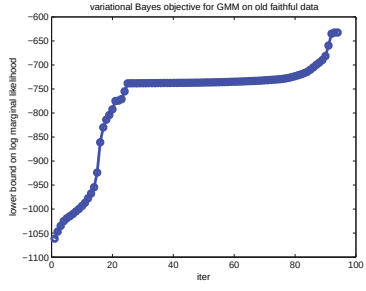


Figure 10.14: Lower bound vs iterations for the VB algorithm in Figure 10.12. The steep parts of the curve correspond to places where the algorithm figures out that it can increase the bound by "killing off" unnecessary mixture components, as described in Section 10.3.6.6. The plateaus correspond to slowly moving the clusters around. Generated by gmm_vb_em.ipynb.

10.3.6.6 Model selection using VBEM

Section 10.3.6.4 discusses a way to choose K automatically, during model fitting, by "killing off" unneeded clusters. An alternative approach is to fit several models, and then to use the variational lower bound to the log marginal likelihood, $\mathcal{L}(K) \leq \log p(\mathcal{D}|K)$, to approximate $p(K|\mathcal{D})$. In particular, if we have a uniform prior, we get the posterior

$$p(K|\mathcal{D}) = \frac{p(\mathcal{D}|K)}{\sum_{K'} p(\mathcal{D}|K')} \approx \frac{e^{\mathcal{L}(K)}}{\sum_{K'} e^{\mathcal{L}(K')}} \tag{10.158}$$

It is shown in [BG06] that the VB approximation to the marginal likelihood is more accurate than BIC [BG06]. However, the lower bound needs to be modified somewhat to take into account the lack of identifiability of the parameters. In particular, although VB will approximate the volume occupied by the parameter posterior, it will only do so around one of the local modes. With K components, there are $K!$ equivalent modes, which differ merely by permuting the labels. Therefore a more accurate approximation to the log marginal likelihood is to use $\log p(\mathcal{D}|K) \approx \mathcal{L}(K) + \log(K!)$.

10.3.7 Variational message passing (VMP)

In this section, we describe the CAVI algorithm for a generic model in which each complete conditional, $p(\boldsymbol{z}_j|\boldsymbol{z}_{-j}, \boldsymbol{x})$, is in the exponential family, i.e.,

$$p(\boldsymbol{z}_j|\boldsymbol{z}_{-j}, \boldsymbol{x}) = h(\boldsymbol{z}_j) \exp[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j) - A_j(\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x}))] \tag{10.159}$$

where $\mathcal{T}(\boldsymbol{z}_j)$ is the vector of sufficient statistics, $\boldsymbol{\eta}_j$ are the natural parameters, A_j is the log partition function, and $h(\boldsymbol{z}_j)$ is the base distribution. This assumption holds if the prior $p(\boldsymbol{z}_j)$ is conjugate to the likelihood, $p(\boldsymbol{z}_{-j}, \boldsymbol{x}|\boldsymbol{z}_j)$.

If Equation (10.159) holds, the mean field update node j becomes

$$q_j(\boldsymbol{z}_j) \propto \exp\left[\mathbb{E}\left[\log p(\boldsymbol{z}_j|\boldsymbol{z}_{-j}, \boldsymbol{x})\right]\right] \tag{10.160}$$

$$= \exp\left[\log h(\boldsymbol{z}_j) + \mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right]^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j) - \mathbb{E}\left[A_j(\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x}))\right]\right] \tag{10.161}$$

$$\propto h(\boldsymbol{z}_j) \exp\left[\mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right]^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j)\right] \tag{10.162}$$

Thus we update the local natural parameters using the expected values of the other nodes. These become the new variational parameters:

$$\boldsymbol{\psi}_j = \mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right] \tag{10.163}$$

We can generalize the above approach to work with any model where each full conditional is conjugate. The resulting algorithm is known as **variational message passing** or **VMP** [WB05] that works for any directed graphical model. VMP is similar to belief propagation (Section 9.3): at each iteration, each node collects all the messages from its parents, and all the messages from its children (which might require the children to get messages from their co-parents), and combines them to compute the expected value of the node's sufficient statistics. The messages that are sent are the expected sufficient statistics of a node, rather than just a discrete or Gaussian distribution (as in BP). Several software libraries have implemented this framework (see e.g., [Win; Min+18; Lut16; Wan17]).

VMP can be extended to the case where each full conditional is conditionally conjugate using the CVI framework in Supplementary Section 10.3.1. See also [ABV21], where they use local Laplace approximations to intractable factors inside of a message passing framework.

10.3.8 Autoconj

The VMP method requires the user to manually specify a graphical model; the corresponding node update equations are then computed for each node using a lookup table, for each possible combination of node types. It is possible to automatically derive these update equations for any conditionally conjugate directed graphical model using a technique called **autoconj** [HJT18]. This is analogous to the use of automatic differentiation (autodiff) to derive the gradient for any differentiable function. (Note that autoconj uses autodiff internally.) The resulting full conditionals can be used for CAVI, and also for Gibbs sampling (Section 12.3).

10.4 More accurate variational posteriors

In general, we can improve the tightness of the ELBO lower bound, and hence reduce the KL divergence of our posterior approximation, if we use more flexible posterior families (although optimizing within more flexible families may be slower, and can incur statistical error if the sample size is low [Bha+21]). In this section, we give several examples of more accurate variational posteriors, going beyond fully factored mean field approximations, or simple unimodal Gaussian approximations.

10.4.1 Structured mean field

The mean field assumption is quite strong, and can sometimes give poor results. Fortunately, sometimes we can exploit **tractable substructure** in our problem, so that we can efficiently handle some kinds of dependencies between the variables in the posterior in an analytic way, rather than assuming they are all independent. This is called the **structured mean field** approach [SJ95].

A common example arises when appling VI to time series models, such as HMMs, where the latent variables within each sequence are usually highly correlated across time. Rather than assuming a fully factorized posterior, we can treat each sequence $\boldsymbol{z}_{n,1:T}$ as a block, and just assume independence between blocks and the parameters: $q(\boldsymbol{z}_{1:N,1:T}, \boldsymbol{\theta}) = q(\boldsymbol{\theta}) \prod_{n=1}^{N} q(\boldsymbol{z}_{n,1:T})$, where $q(\boldsymbol{z}_{n,1:T}) = \prod_t q(\boldsymbol{z}_{n,t}|\boldsymbol{z}_{n,t-1})$. We can compute the joint distribution $q(\boldsymbol{z}_{n,1:T})$, taking into account the dependence between time steps, using the forwards-backwards algorithm. For details, see [JW14; Fot+14]. A similar approach was applied to the factorial HMM model, as we discuss in Supplementary Section 10.3.2.

An automatic way to derive a structured variational approximation to a probabilistic model, specified by a probabilistic programming language, is discussed in [AHG20].

10.4.2 Hierarchical (auxiliary variable) posteriors

Suppose $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \prod_k q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x})$ is a factorized distribution, such as a diagonal Gaussian. This does not capture dependencies between the latent variables (components of $\boldsymbol{z}$). We could of course use a full covariance matrix, but this might be too expensive.

An alternative approach is to use a hierarchical model, in which we add **auxiliary latent variables** $\boldsymbol{a}$, which are used to increase the flexibility of the variational posterior. In particular, we can still assume $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{a})$ is conditionally factorized, but when we marginalize out $\boldsymbol{a}$, we induce dependencies between the elements of $\boldsymbol{z}$, i.e.,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{a}) q_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{x}) d\boldsymbol{a} \neq \prod_k q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}) \tag{10.164}$$

This is called a **hierarchical variational model** [Ran16], or an **auxiliary variable deep generative model** [Maa+16].

In [TRB16], they model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{a})$ as a Gaussian process, which is a flexible nonparametric distribution (see Chapter 18), where $\boldsymbol{a}$ are the inducing points. This combination is called a **variational GP**.

10.4.3 Normalizing flow posteriors

Normalizing flows are a class of probability models which work by passing a simple source distribution, such as a diagonal Gaussian, through a series of nonlinear, but invertible, mappings f to create a more complex distribution. This can be used to get more accurate posterior approximations than standard Gaussian VI, as we discuss in Section 23.1.2.2.

10.4.4 Implicit posteriors

In Chapter 26, we discuss implicit probability distributions, which are models which we can sample from, but which we cannot evaluate pointwise. For example, consider passing a Gaussian noise term, $\boldsymbol{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, through a nonlinear, *non-invertible* mapping f to create $\boldsymbol{z} = f(\boldsymbol{z}_0)$; it is easy to sample from $q(\boldsymbol{z})$, but it is intractable to evaluate the density $q(\boldsymbol{z})$ (unlike with flows). This makes it hard to evaluate the log density ratio $\log p_{\boldsymbol{\theta}}(\boldsymbol{z})/q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})$, which is needed to compute the ELBO. However, we can use the same method as is used in GANs (generative adversarial networks, Chapter 26), in which we train a classifier that discriminates prior samples from samples from the variational posterior by evaluating $T(\boldsymbol{x}, \boldsymbol{z}) = \log q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z})$. See e.g., [TR19] for details.

10.4.5 Combining VI with MCMC inference

There are various ways to combine variational inference with MCMC to get an improved approximate posterior. In [SKW15], they propose **Hamiltonian variational inference**, in which they train an inference network to initialize an HMC sampler (Section 12.5). The gradient of the log posterior (wrt the latents), which is needed by HMC, is given by

$$\nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \nabla_{\boldsymbol{z}} \log \left[p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] = \nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \tag{10.165}$$

This is easy to compute. They use the final sample to approximate the posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. To compute the entropy of this distribution, they also learn an auxiliary inverse inference network to reverse the HMC Markov chain.

A simpler approach is proposed in [Hof17]. Here they train an inference network to initialize an HMC sampler, using the standard ELBO for $\boldsymbol{\phi}$, but they optimize the generative parameters $\boldsymbol{\theta}$ using

a stochastic approximation to the log marginal likelihood, given by $\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$ where $\boldsymbol{z}$ is a sample from the HMC chain. This does not require learning a reverse inference network, and avoids problems with variational pruning, since it does not use the ELBO for training the generative model.

10.5 Tighter bounds

Another way to improve the quality of the posterior approximation is to optimize q wrt a bound that is a tighter approximation to the log marginal likelihood compared to the standard ELBO. We give some examples below.

10.5.1 Multi-sample ELBO (IWAE bound)

In this section, we discuss a method known as the **importance weighted autoencoder** or **IWAE** [BGS16], which is a way to tighten the variational lower bound by using self-normalized importance sampling (Section 11.5.2). (It can also be interpreted as standard ELBO maximization in an expanded model, where we add extra auxiliary variables [CMD17; DS18; Tuc+19].)

Let the inference network $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ be viewed as a proposal distribution for the target posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. Define $w_s^* = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})}$ as the unnormalized importance weight for a sample, and $w_s = w_s^*/(\sum_{s'=1}^S w_{s'}^*)$ as the normalized importance weights. From Equation (11.43) we can compute an estimate of the marginal likelihood $p(\boldsymbol{x})$ using

$$\hat{p}_S(\boldsymbol{x}|\boldsymbol{z}_{1:S}) \triangleq \frac{1}{S}\sum_{k=1}^{S}\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})} = \frac{1}{S}\sum_{k=1}^{S} w_s \tag{10.166}$$

This is unbiased, i.e., $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\hat{p}_S(\boldsymbol{x}|\boldsymbol{z}_{1:S})\right] = p(\boldsymbol{x})$, where $q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x}) = \prod_{s=1}^S q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})$. In addition, since the estimator is always positive, we can take logarithms, and thus obtain a stochastic lower bound on the log likelihood:

$$Ł_S(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{x}) \triangleq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\log\left(\frac{1}{S}\sum_{s=1}^{S} w_s\right)\right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\log \hat{p}_S(\boldsymbol{z}_{1:S})\right] \tag{10.167}$$

$$\leq \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\hat{p}_S(\boldsymbol{z}_{1:S})\right] = \log p(\boldsymbol{x}) \tag{10.168}$$

where we used Jensen's inequality in the penultimate line, and the unbiased property in the last line. This is called the **multi-sample ELBO** or **IWAE bound** [BGS16]. The gradients of this expression wrt $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are given in Equation (10.179). If $S = 1$, $Ł_S$ reduces to the standard ELBO:

$$Ł_1(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log w\right] = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \tag{10.169}$$

One can show [BGS16] that increasing the number of samples S is guaranteed to make the bound tighter, thus making it a better proxy for the log likelihood. Intuitively, averaging the S samples inside the log removes the need for every sample $\boldsymbol{z}_s$ to explain the data $\boldsymbol{x}$. This encourages the proposal distribution q to be less concentrated than the single-sample variational posterior.

10.5.1.1 Pathologies of optimizing the IWAE bound

Unfortunately, increasing the number of samples in the IWAE bound can decrease the signal to noise ratio, resulting in learning a worse model [Rai+18a]. Intuitively, the reason this happens is that increasing S reduces the dependence of the bound on the quality of the inference network, which makes the gradient of the ELBO wrt $\boldsymbol{\phi}$ less informative (higher variance).

One solution to this is to use the **doubly reparameterized gradient estimator** [TL18b]. Another approach is to use alternative estimation methods that avoid ELBO maximization, such as using the thermodynamic variational objective (see Section 10.5.2) or the reweighted wake-sleep algorithm (see Section 10.6).

10.5.2 The thermodynamic variational objective (TVO)

In [MLW19; Bre+20b], they present the **thermodynamic variational objective** or **TVO**. This is an alternative to IWAE for creating tighter variational bounds, which has certain advantages, particularly for posteriors that are not reparameterizable (e.g., discrete latent variables). The framework also has close connections with the reweighted wake-sleep algorithm from Section 10.6, as we will see in Section 10.5.3.

The TVO technique uses **thermodynamic integration**, also called **path sampling**, which is a technique used in physics and phylogenetics to approximate intractable normalization constants of high dimensional distributions (see e.g., [GM98; LP06; FP08]). This is based on the insight that it is easier to calculate the ratio of two unknown constants than to calculate the constants themselves. This is similar to the idea behind annealed importance sampling (Section 11.5.4), but TI is deterministic. For details, see [MLW19; Bre+20b].

10.5.3 Minimizing the evidence upper bound

Recall that the evidence lower bound or ELBO is given by

$$\text{Ł}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{\mathbb{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.170}$$

By analogy, we can define the **evidence upper bound** or **EUBO** as follows:

$$\text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) + D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right) \geq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.171}$$

Minimizing this wrt the variational parameters $\boldsymbol{\phi}$, as an alternative to maxmimizing the ELBO, was proposed in [MLW19], where they showed that it can sometimes converge to the true $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ faster.

The above bound is for a specific input $\boldsymbol{x}$. If we sample $\boldsymbol{x}$ from the generative model, and minimize $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x})\right]$ wrt $\boldsymbol{\phi}$, we recover the sleep phase of the wake-sleep algorithm (see Section 10.6.2).

Now suppose we sample $\boldsymbol{x}$ from the empirical distribution, and minimize $\mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[\text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x})\right]$ wrt $\boldsymbol{\phi}$. To approximate the expectation, we can use self-normalized importance sampling, as in Equation (10.188), to get

$$\nabla_{\boldsymbol{\phi}} \text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}^s|\boldsymbol{x}) \tag{10.172}$$

where $\overline{w}_s = w^{(s)}/(\sum_{s'} w^{(s')})$, and $w^{(s)} = \frac{p(\boldsymbol{x},\boldsymbol{z}^s)}{q(\boldsymbol{z}^s|\boldsymbol{\phi}_t)}$. This is equivalent to the "daydream" update (aka "wake-phase $\boldsymbol{\phi}$ update") of the wake-sleep algorithm (see Section 10.6.3).

10.6 Wake-sleep algorithm

So far in this chapter we have focused on fitting latent variable models by maximizing the ELBO. This has two main drawbacks. First, it does not work well when we have discrete latent variables, because in such cases we cannot use the reparameterization trick; thus we have to use higher variance estimators, such as REINFORCE (see Section 10.2.3). Second, even in the case where we can use the reparameterization trick, the lower bound may not be very tight. We can improve the tightness by using the IWAE multi-sample bound (Section 10.5.1), but paradoxically this may not result in learning a better model, for reasons discussed in Section 10.5.1.1.

In this section, we discuss a different way to jointly train generative and inference models, which avoids some of the problems with ELBO maximization. The method is known as the **wake-sleep algorithm** [Hin+95; BB15b; Le+19; FT19]. because it alternates between two steps: in the wake phase, we optimize the generative model parameters $\boldsymbol{\theta}$ to maximize the marginal likelihood of the observed data (we approximate $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ by drawing importance samples from the inference network), and in the sleep phase, we optimize the inference model parameters $\boldsymbol{\phi}$ to learn to invert the generative model by training the inference network on labeled $(\boldsymbol{x}, \boldsymbol{z})$ pairs, where $\boldsymbol{x}$ are samples generated by the current model parameters. This can be viewed as a form of **adaptive importance sampling**, which iteratively improves its proposal, while simultaneously optimizing the model. We give further details below.

10.6.1 Wake phase

In the **wake phase**, we minimize the KL divergence from the empirical distribution to the model's distribution:

$$\mathcal{L}(\boldsymbol{\theta}) = D_{\mathbb{KL}}\left(p_{\mathcal{D}}(\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x})\right) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] + \text{const} \tag{10.173}$$

where $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z}$. This is equivalent to maximizing the likelihood of the observed data:

$$\ell(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \tag{10.174}$$

Since the log marginal likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ cannot be computed exactly, we will approximate it. In the original wake-sleep paper, they proposed to use the ELBO lower bound. In the **reweighted wake-sleep** (RWS) algorithm of [BB15b; Le+19], they propose to use the IWAE bound from Section 10.5.1 instead. In particular, if we draw S samples from the inference network, $\boldsymbol{z}_s \sim q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$, we get the following estimator:

$$\ell(\boldsymbol{\theta}|\boldsymbol{\phi}, \boldsymbol{x}) = \log\left(\frac{1}{S}\sum_{s=1}^{S} w_s\right) \tag{10.175}$$

where $w_s = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})}$. Note that this is the same as the IWAE bound in Equation (10.168).

We now discuss how to compute the gradient of this objective wrt $\boldsymbol{\theta}$ or $\boldsymbol{\phi}$. Using the log-derivative trick, we have that

$$\nabla \log w_s = \frac{1}{w_s} \nabla w_s \tag{10.176}$$

Hence

$$\nabla \ell(\boldsymbol{\theta}|\boldsymbol{\phi}, \boldsymbol{x}) = \frac{1}{\frac{1}{S}\sum_{s=1}^{S} w_s} \left(\frac{1}{S} \sum_{s=1}^{S} \nabla w_s \right) \tag{10.177}$$

$$= \frac{1}{\sum_{s=1}^{S} w_s} \left(\sum_{s=1}^{S} w_s \nabla \log w_s \right) \tag{10.178}$$

$$= \sum_{s=1}^{S} \overline{w}_s \nabla \log w_s \tag{10.179}$$

where $\overline{w}_s = w_s / (\sum_{s'=1}^{S} w_{s'})$.

In the case of the derivatives wrt $\boldsymbol{\theta}$, we have

$$\nabla_{\boldsymbol{\theta}} \log w_s = \frac{1}{w_s} \nabla_{\boldsymbol{\theta}} w_s = \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)} \nabla_{\boldsymbol{\theta}} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})} = \frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s) = \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s) \tag{10.180}$$

and hence we get

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}|\boldsymbol{\phi}, \boldsymbol{x}) \sum_{s=1}^{S} \overline{w}_s \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s) \tag{10.181}$$

10.6.2 Sleep phase

In the **sleep phase**, we try to minimize the KL divergence between the true posterior (under the current model) and the inference network's approximation to that posterior:

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \left[D_{\mathbb{KL}} \left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right) \right] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})} \left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] + \text{const} \tag{10.182}$$

Equivalently, we can maximize the following log likelihood objective:

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{z},\boldsymbol{x}) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})} \left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] \tag{10.183}$$

where $p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. We see that the sleep phase amounts to maximum likelihood training of the inference network based on samples from the generative model. These "fantasy samples", created while the network "dreams", can be easily generated using ancestral sampling (Section 4.2.5). If we use S such samples, the objective becomes

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s) \tag{10.184}$$

where $(\boldsymbol{z}'_s, \boldsymbol{x}'_s) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$. The gradient of this is given by

$$\nabla_{\boldsymbol{\phi}} \ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s) \tag{10.185}$$

We do not require $q_{\boldsymbol{\phi}}(\boldsymbol{z}'|\boldsymbol{x})$ to be reparameterizable, since the samples are drawn from a distribution that is independent of $\boldsymbol{\phi}$. This means it is easy to apply this method to models with discrete latent variables.

10.6.3 Daydream phase

The disadvantage of the sleep phase is that the inference network, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, is trying to follow a moving target, $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. Furthermore, it is only being trained on synthetic data from the model, not on real data. The reweighted wake-sleep algorithm of [BB15b] proposed to learn the inference network by using real data from the empirical distribution, in addition to fantasy data. They call the case where you use real data the "**wake-phase q update**", but we will call it the "**daydream phase**", since, unlike sleeping, the system uses real data $\boldsymbol{x}$ to update the inference model, instead of fantasies.[1] [Le+19] went further, and proposed to only use the wake and daydream phases, and to skip the sleep phase entirely.

In more detail, the new objective which we want to minimize becomes

$$\mathcal{L}(\boldsymbol{\phi}|\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})} \left[D_{\mathbb{KL}} \left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right) \right] \tag{10.186}$$

We can compute a single sample approximation to the negative of the above expression as follows:

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] \tag{10.187}$$

where $\boldsymbol{x} \sim p_{\mathcal{D}}$. We can approximate this expectation using importance sampling, with $q_{\boldsymbol{\phi}}$ as the proposal. This results in the following estimator of the gradient for each datapoint:

$$\nabla_{\boldsymbol{\phi}} \ell(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} \approx \sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x}) \tag{10.188}$$

where $\boldsymbol{z}_s \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})$ and $\overline{w}_s$ are the normalized weights.

We see that Equation (10.188) is very similar to Equation (10.185). The key difference is that in the daydream phase, we sample from $(\boldsymbol{x}, \boldsymbol{z}_s) \sim p_{\mathcal{D}}(\boldsymbol{x}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, where $\boldsymbol{x}$ is a real datapoint, whereas in the sleep phase, we sample from $(\boldsymbol{x}'_s, \boldsymbol{z}'_s) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$, where $\boldsymbol{x}'_s$ is generated datapoint.

10.6.4 Summary of algorithm

We summarize the RWS algorithm in Algorithm 10.5. The disadvantage of the RWS algorithm is that it does not optimize a single well-defined objective, so it is not clear if the method will converge, in contrast to ELBO maximization. On the other hand, the method is fairly simple, since it consists of two alternating weighted maximum likelihood problems. It can also be shown to "sandwich" a

1. We thank Rif A. Saurous for suggesting this term.

Algorithm 10.5: One SGD update using wake-sleep algorithm.

1 Sample $\boldsymbol{x}_n$ from dataset
2 Draw S samples from inference network: $\boldsymbol{z}_s \sim q(\boldsymbol{z}|\boldsymbol{x}_n)$
3 Compute unnormalized weights: $w_s = \frac{p(\boldsymbol{x}_n, \boldsymbol{z}_s)}{q(\boldsymbol{z}_s|\boldsymbol{x}_n)}$
4 Compute normalized weights: $\overline{w}_s = \frac{w_s}{\sum_{s'=1}^{S} w_{s'}}$
5 Optional: Compute estimate of log likelihood: $\log p(\boldsymbol{x}_n) = \log(\frac{1}{S}\sum_{s=1}^{S} w_s)$
6 Wake phase: Update $\boldsymbol{\theta}$ using $\sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_s, \boldsymbol{x}_n)$
7 Daydream phase: Update $\boldsymbol{\phi}$ using $\sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x}_n)$
8 Optional sleep phase: Draw S samples from model, $(\boldsymbol{x}'_s, \boldsymbol{z}'_s) \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$ and update $\boldsymbol{\phi}$ using $\frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s)$
9 b

lower and upper bound of the log marginal likelihood. We can think of this in terms of the two joint distributions $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ and $q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x})q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$:

$$\text{wake phase } \min_{\boldsymbol{\theta}} D_{\mathbb{KL}}\left(q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right) \tag{10.189}$$

$$\text{daydream phase } \min_{\boldsymbol{\phi}} D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \parallel q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})\right) \tag{10.190}$$

10.7 Expectation propagation (EP)

One problem with lower bound maximization (i.e., standard VI) is that we are minimizing $D_{\mathbb{KL}}(q \parallel p)$, which induces **zero-forcing** behavior, as we discussed in Section 5.1.4.1. This means that $q(\boldsymbol{z}|\boldsymbol{x})$ tends to be too compact (over-confident), to avoid the situation in which $q(\boldsymbol{z}|\boldsymbol{x}) > 0$ but $p(\boldsymbol{z}|\boldsymbol{x}) = 0$, which would incur infinite KL penalty.

Although zero-forcing can be desirable behavior for some multi-modal posteriors (e.g., mixture models), it is not so reasonable for many unimodal posteriors (e.g., Bayesian logistic regression, or GPs with log-concave likelihoods). One way to avoid this problem is to minimize $D_{\mathbb{KL}}(p \parallel q)$, which is zero-avoiding, as we discussed in Section 5.1.4.1. This tends to result in broad posteriors, which avoids overconfidence. In this section, we discuss **expectation propagation** or **EP** [Min01b], which can be seen as a local approximation to $D_{\mathbb{KL}}(p \parallel q)$.

10.7.1 Algorithm

We assume the exact posterior can be written as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z_p}\hat{p}(\boldsymbol{\theta}), \;\; \hat{p}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^{K} f_k(\boldsymbol{\theta}) \tag{10.191}$$

where $\hat{p}(\boldsymbol{\theta})$ is the unnormalized posterior, p_0 is the prior, f_k corresponds to the k'th likelihood term or **local factor** (also called a **site potential**). Here $Z_p = p(\mathcal{D})Z_0$ is the normalization constant for

the posterior, where Z_0 is the normalization constant for the prior. To simplify notation, we let $f_0(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta})$ be the prior.

We will approximate the posterior as follows:

$$q(\boldsymbol{\theta}) = \frac{1}{Z_q}\hat{q}(\boldsymbol{\theta}), \quad \hat{q}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^{K} \tilde{f}_k(\boldsymbol{\theta}) \tag{10.192}$$

where $\tilde{f}_k \in \mathcal{Q}$ is the approximate local factor, and $\mathcal{Q}$ is some tractable family in the exponential family, usually a Gaussian [Gel+14b].

We will optimize each $\tilde{f}_i$ in turn, keeping the others fixed. We initialize each $\tilde{f}_i$ using an uninformative distribution from the family $\mathcal{Q}$. so $q(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta})$.

To compute the new local factor $\tilde{f}_i^{\text{new}}$, we proceed as follows. First we compute the **cavity distribution** by deleting the $\tilde{f}_i$ from the approximate posterior by dividing it out:

$$q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta})}{\tilde{f}_i(\boldsymbol{\theta})} \propto \prod_{k \neq i} \tilde{f}_k(\boldsymbol{\theta}) \tag{10.193}$$

This division operation can be implemented by subtracting the natural parameters, as explained in Section 2.3.3.2. The cavity distribution represents the effect of all the factors except for f_i (which is approximated by $\tilde{f}_i$).

Next we (conceptually) compute the **tilted distribution** by multiplying the exact factor f_i onto the cavity distribution:

$$q_i^{\text{tilted}}(\boldsymbol{\theta}) = \frac{1}{Z_i} f_i(\boldsymbol{\theta}) q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) \tag{10.194}$$

where $Z_i = \int q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) f_i(\boldsymbol{\theta}) d\boldsymbol{\theta}$ is the normalization constant for the tilted distribution. This is the result of combining the current approximation, excluding factor i, with the exact f_i term.

Unfortunately, the resulting tilted distribution may be outside of our model family (e.g., if we combine a Gaussian prior with a non-Gaussian likelihood). So we will approximate the tilted distribution as follows:

$$q_i^{\text{proj}}(\boldsymbol{\theta}) = \text{proj}(q_i^{\text{tilted}}) \triangleq \underset{\tilde{q} \in \mathcal{Q}}{\text{argmin}}\, D(q_i^{\text{tilted}} || \tilde{q}) \tag{10.195}$$

This can be thought of as projecting the tilted distribution into the approximation family. If $D(q_i^{\text{tilted}}||q) = D_{\mathbb{KL}}\left(q_i^{\text{tilted}} \parallel q\right)$, this can be done by moment matching, as shown in Section 5.1.4.2. For example, suppose the cavity distribution is Gaussian, $q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) = \mathcal{N}_c(\boldsymbol{\theta}|\boldsymbol{r}_{-i}, \mathbf{Q}_{-i})$, using the canonical parameterization. Then the log of the tilted distribution is given by

$$\log q_i^{\text{tilted}}(\boldsymbol{\theta}) = \alpha \log f_i(\boldsymbol{\theta}) - \frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}}\mathbf{Q}_{-i}\boldsymbol{\theta} + \boldsymbol{r}_{-i}^{\mathsf{T}}\boldsymbol{\theta} + \text{const} \tag{10.196}$$

Let $\hat{\boldsymbol{\theta}}$ be a local maximum of this objective. If $\mathcal{Q}$ is the set of Gaussians, we can compute the projected tilted distribution as a Gaussian with the following parameters:

$$\mathbf{Q}_{\backslash i} = -\nabla^2_{\boldsymbol{\theta}} \log q_i^{\text{tilted}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, \quad \boldsymbol{r}_{\backslash i} = \mathbf{Q}_{\backslash i}\hat{\boldsymbol{\theta}} \tag{10.197}$$

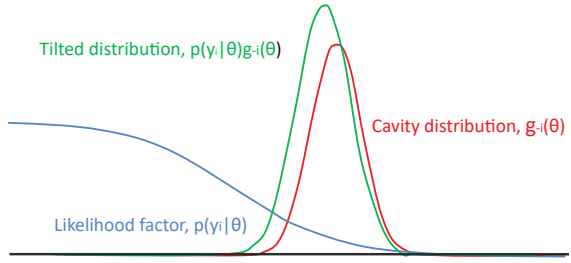


Figure 10.15: Combining a logistic likelihood factor $f_i = p(y_i|\boldsymbol{\theta})$ with the cavity prior, $q_{-i}^{cavity} = g_{-i}(\boldsymbol{\theta})$, to get the tilted distribution, $q_i^{tilted} = p(y_i|\boldsymbol{\theta})g_{-i}(\boldsymbol{\theta})$. Adapted from Figure 2 of [Gel+14b].

This is called **Laplace propagation** [SVE04]. For more general distributions, we can use Monte Carlo approximations; this is known as **blackbox EP** [HL+16a; Li+18c].

Finally, we compute a local factor that, if combined with the cavity distribution, would give the same results as this projected distribution:

$$\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = \frac{q_i^{\text{proj}}(\boldsymbol{\theta})}{q_{-i}^{\text{cavity}}(\boldsymbol{\theta})} \tag{10.198}$$

We see that $q_{-i}^{\text{cavity}}(\boldsymbol{\theta})\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = q_i^{\text{proj}}(\boldsymbol{\theta})$, so combining this approximate factor with the cavity distribution results in a distribution which is the best possible approximation (within $\mathcal{Q}$) to the results of using the exact factor.

10.7.2 Example

Figure 10.15 illustrates the process of combining a very non-Gaussian likelihood f_i with a Gaussian cavity prior q_{-i}^{cavity} to yield a nearly Gaussian tilted distribution q_i^{tilted}, which can then be approximated by a Gaussian using projection.

Thus instead of trying to "Gaussianize" each likelihood term f_i in isolation (as is done, e.g., in EKF), we try to find the best local factor $\tilde{f}_i$ (within some family) that achieves approximately the same effect, when combined with all the other terms (represented by the cavity distribution, q_{-i}), as using the exact factor f_i. That is, we choose a local factor that works well in the context of all the other factors.

10.7.3 EP as generalized ADF

We can view EP as a generalization of the ADF algorithm discussed in Section 8.6. ADF is a form of sequential Bayesian inference. At each step, it maintains a tractable approximation to the posterior, $q_t(\boldsymbol{z}) \in \mathcal{Q}$, updates it with the likelihood from the next observation, $\hat{p}_{t+1}(\boldsymbol{z}) \propto q_t(\boldsymbol{z})p(\boldsymbol{x}_t|\boldsymbol{z})$, and then projects the resulting updated posterior back to the tractable family using $q_{t+1} = \text{argmin}_{q\in\mathcal{Q}} D_{\mathbb{KL}}(\hat{p}_{t+1} \parallel q)$. ADF minimizes KL in the desired direction. However, it is a sequential algorithm, designed for the online setting. In the batch setting, the method can given

different results depending on the order in which the updates are performed. In addition, if we perform multiple passes over the data, we will include the same likelihood terms multiple times, resulting in an overconfident posterior. EP overcomes this problem.

10.7.4 Optimization issues

In practice, EP can be numerically unstable. For example, if we use Gaussians as our local factors, we might end up with negative variance when we subtract the natural parameters. To reduce the chance of this, it is common to use damping, in which we perform a partial update of each factor with a step size of δ. More precisely, we change the final step to be the following:

$$\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = \left(\tilde{f}_i(\boldsymbol{\theta})\right)^{1-\delta} \left(\frac{q_i^{\text{proj}}(\boldsymbol{\theta})}{q_{-i}^{\text{cavity}}}\right)^{\delta} \tag{10.199}$$

This can be implemented by scaling the natural parameters by δ. [ML02] suggest $\delta = 1/K$ as a safe strategy (where K is the number of factors), but this results in very slow convergence. [Gel+14b] suggest starting with $\delta = 0.5$, and then reducing to $\delta = 1/K$ over K iterations.

In addition to numerical stability, there is no guarantee that EP will converge in its vanilla form, although empirically it can work well, especially with log-concave factors f_i (e.g., as in GP classifiers).

10.7.5 Power EP and α-divergence

We also have a choice about what divergence measure $D(q_i^{\text{tilted}}||q)$ to use when we approximate the tilted distribution. If we use $D_{\mathbb{KL}}\left(q_i^{\text{tilted}} \parallel q\right)$, we recover classic EP, as described above. If we use $D_{\mathbb{KL}}\left(q \parallel q_i^{\text{tilted}}\right)$, we recover the reverse KL used in standard variational inference. We can generalize the above results by using α-divergences (Section 2.7.1.2), which allow us to interpolate between mode seeking and mode covering behavior, as shown in Figure 2.20. We can optimize the α-divergence by using the **power EP** method of [Min04].

Algorithmically, this is a fairly small modification to regular EP. In particular, we first compute the cavity distribution, $q_{-i}^{\text{cavity}} \propto \frac{q}{\tilde{f}_i^{\alpha}}$; we then approximate the tilted distribution, $q_i^{\text{proj}} = \text{proj}(q_{-i}^{\text{cavity}} f_i^{\alpha})$; and finally we compute the new factor $\tilde{f}_i^{\text{new}} \propto \left(\frac{q_i^{\text{proj}}}{q_{-i}^{\text{cavity}}}\right)^{1/\alpha}$.

10.7.6 Stochastic EP

The main disadvantage of EP in the big data setting is that we need to store the $\tilde{f}_n(\boldsymbol{\theta})$ terms for each datapoint n, so we can compute the cavity distribution. If $\boldsymbol{\theta}$ has D dimensions, and we use full covariance Gaussians, this requires $O(ND^2)$ memory.

The idea behind **stochastic EP** [LHLT15] is to approximate the local factors with a shared factor that acts like an aggregated likelihood, i.e.,

$$\prod_{n=1}^{N} f_n(\boldsymbol{\theta}) \approx \tilde{f}(\boldsymbol{\theta})^N \tag{10.200}$$

where typically $f_n(\boldsymbol{\theta}) = p(\boldsymbol{x}_n|\boldsymbol{\theta})$. This exploits the fact that the posterior only cares about approximating the product of the likelihoods, rather than each likelihood separately. Hence it suffices for $\tilde{f}(\boldsymbol{\theta})$ to approximate the average likelihood.

We can modify EP to this setting as follows. First, when computing the cavity distribution, we use

$$q_{-1}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})/\tilde{f}(\boldsymbol{\theta}) \tag{10.201}$$

We then compute the tilted distribution

$$q_{\backslash n}(\boldsymbol{\theta}) \propto f_n(\boldsymbol{\theta}) q_{-1}(\boldsymbol{\theta}) \tag{10.202}$$

Next we derive the new local factor for this datapoint using moment matching:

$$\tilde{f}_n(\boldsymbol{\theta}) = \text{proj}(q_{\backslash n}(\boldsymbol{\theta}))/q_{-1}(\boldsymbol{\theta}) \tag{10.203}$$

Finally, we perform a damped update of the average likelihood $\tilde{f}(\boldsymbol{\theta})$ using this new local factor:

$$\tilde{f}_{\text{new}}(\boldsymbol{\theta}) = \tilde{f}_{\text{old}}(\boldsymbol{\theta})^{1-1/N} \tilde{f}_n(\boldsymbol{\theta})^{1/N} \tag{10.204}$$

The ADF algorithm is similar to SEP, in that we compute the tilted distribution $q_{\backslash t} \propto f_t q_{t-1}$ and then project it, without needing to keep the f_t factors. The difference is that instead of using the cavity distribution $q_{-1}(\boldsymbol{\theta})$ as a prior, it uses the posterior from the previous time step, q_{t-1}. This avoids the need to compute and store $\tilde{f}$, but results in overconfidence in the batch setting.

11 Monte Carlo methods

11.1 Introduction

In this chapter, we discuss **Monte Carlo methods**, which are a stochastic approach to solving numerical integration problems. The name refers to the "Monte Carlo" casino in Monaco; this was used as a codename by von Neumann and Ulam, who invented the technique while working on the atomic bomb during WWII. Since then, the technique has become widely adopted in physics, statistics, machine learning, and many areas of science and engineering.

In this chapter, we give a brief introduction to some key concepts. In Chapter 12, we discuss MCMC, which is the most widely used MC method for high-dimensional problems. In Chapter 13, we discuss SMC, which is widely used for MC inference in state space models, but can also be applied more generally. For more details on MC methods, see e.g., [Liu01; RC04; KTB11; BZ20].

11.2 Monte Carlo integration

We often want to compute the expected value of some function of a random variable, $\mathbb{E}[f(\mathbf{X})]$. This requires computing the following integral:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} \tag{11.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}^m$, and $p(\boldsymbol{x})$ is the target distribution of $\mathbf{X}$.[1] In low dimensions (up to, say, 3), we can compute the above integral efficiently using **numerical integration**, which (adaptively) computes a grid, and then evaluates the function at each point on the grid.[2] But this does not scale to higher dimensions.

An alternative approach is to draw multiple random samples, $\boldsymbol{x}_n \sim p(\boldsymbol{x})$, and then to compute

$$\mathbb{E}[f(\boldsymbol{x})] \approx \frac{1}{N_s} \sum_{n=1}^{N_s} f(\boldsymbol{x}_n) \tag{11.2}$$

This is called **Monte Carlo integration**. It has the advantage over numerical integration that the function is only evaluated in places where there is non-negligible probability, so it does not

1. In many cases, the target distribution may be the posterior $p(\boldsymbol{x}|\boldsymbol{y})$, which can be hard to compute; in such problems, we often work with the unnormalized distribution, $\tilde{p}(\boldsymbol{x}) = p(\boldsymbol{x}, \boldsymbol{y})$, instead, and then normalize the results using $Z = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} = p(\boldsymbol{y})$.
2. In 1d, numerical integration is called **quadrature**; in higher dimensions, it is called **cubature** [Sar13].

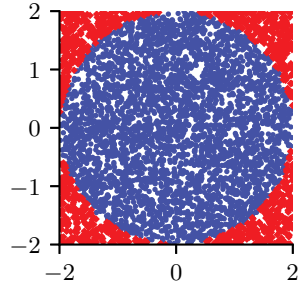


Figure 11.1: Estimating π by Monte Carlo integration using 5000 samples. Blue points are inside the circle, red points are outside. Generated by mc_estimate_pi.ipynb.

need to uniformly cover the entire space. In particular, it can be shown that the accuracy is in principle independent of the dimensionality of $\boldsymbol{x}$, and only depends on the number of samples N_s (see Section 11.2.2 for details). The catch is that we need a way to generate the samples $\boldsymbol{x}_n \sim p(\boldsymbol{x})$ in the first place. In addition, the estimator may have high variance. We will discuss this topic at length in the sections below.

11.2.1 Example: estimating π by Monte Carlo integration

MC integration can be used for many applications, not just in ML and statistics. For example, suppose we want to estimate π. We know that the area of a circle with radius r is πr^2, but it is also equal to the following definite integral:

$$I = \int_{-r}^{r} \int_{-r}^{r} \mathbb{I}\left(x^2 + y^2 \leq r^2\right) dxdy \tag{11.3}$$

Hence $\pi = I/(r^2)$. Let us approximate this by Monte Carlo integration. Let $f(x, y) = \mathbb{I}\left(x^2 + y^2 \leq r^2\right)$ be an indicator function that is 1 for points inside the circle, and 0 outside, and let $p(x)$ and $p(y)$ be uniform distributions on $[-r, r]$, so $p(x) = p(y) = 1/(2r)$. Then

$$I = (2r)(2r) \int \int f(x, y)p(x)p(y)dxdy \tag{11.4}$$

$$= 4r^2 \int \int f(x, y)p(x)p(y)dxdy \tag{11.5}$$

$$\approx 4r^2 \frac{1}{N_s} \sum_{n=1}^{N_s} f(x_n, y_n) \tag{11.6}$$

Using 5000 samples, we find $\hat{\pi} = 3.10$ with standard error 0.09 compared to the true value of $\pi = 3.14$. We can plot the points that are accepted or rejected as in Figure 11.1.

11.2.2 Accuracy of Monte Carlo integration

The accuracy of an MC approximation increases with sample size. This is illustrated in Figure 11.2. On the top line, we plot a histogram of samples from a Gaussian distribution. On the bottom line,

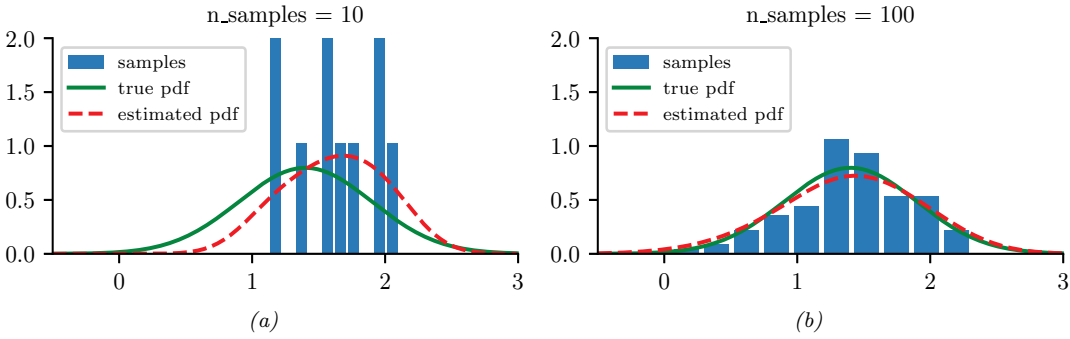


Figure 11.2: 10 and 100 samples from a Gaussian distribution, $\mathcal{N}(\mu = 1.5, \sigma^2 = 0.25)$. A dotted red line denotes kernel density estimate derived from the samples. Generated by mc_accuracy_demo.ipynb.

we plot a smoothed version of these samples, created using a kernel density estimate. This smoothed distribution is then evaluated on a dense grid of points and plotted. Note that this smoothing is just for the purposes of plotting, it is not used for the Monte Carlo estimate itself.

If we denote the exact mean by $\mu = \mathbb{E}[f(X)]$, and the MC approximation by $\hat{\mu}$, one can show that, with independent samples,

$$(\hat{\mu} - \mu) \to \mathcal{N}(0, \frac{\sigma^2}{N_s}) \tag{11.7}$$

where

$$\sigma^2 = \mathbb{V}[f(X)] = \mathbb{E}\left[f(X)^2\right] - \mathbb{E}[f(X)]^2 \tag{11.8}$$

This is a consequence of the central limit theorem. Of course, σ^2 is unknown in the above expression, but it can be estimated by MC:

$$\hat{\sigma}^2 = \frac{1}{N_s}\sum_{n=1}^{N_s}(f(x_n) - \hat{\mu})^2 \tag{11.9}$$

Thus for large enough N_s we have

$$P\left\{\hat{\mu} - 1.96\frac{\hat{\sigma}}{\sqrt{N_s}} \leq \mu \leq \hat{\mu} + 1.96\frac{\hat{\sigma}}{\sqrt{N_s}}\right\} \approx 0.95 \tag{11.10}$$

The term $\sqrt{\frac{\hat{\sigma}^2}{N_s}}$ is called the (numerical or empirical) **standard error**, and is an estimate of our uncertainty about our estimate of μ.

If we want to report an answer which is accurate to within $\pm\epsilon$ with probability at least 95%, we need to use a number of samples N_s which satisfies $1.96\sqrt{\hat{\sigma}^2/N_s} \leq \epsilon$. We can approximate the 1.96 factor by 2, yielding $N_s \geq \frac{4\hat{\sigma}^2}{\epsilon^2}$.

The remarkable thing to note about the above results is that the error in the estimate, σ^2/N_s, is theoretically independent of the dimensionality of the integral. The catch is that sampling from high dimensional distributions can be hard. We turn to that topic next.

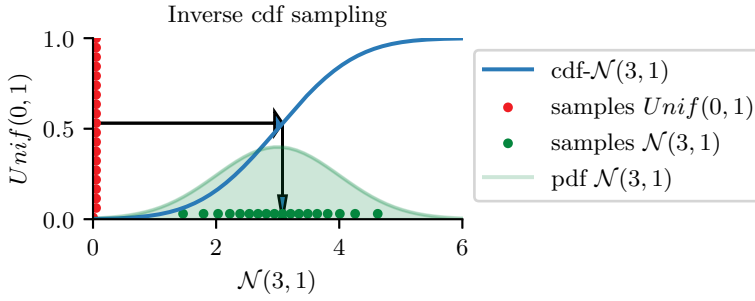


Figure 11.3: Sampling from $\mathcal{N}(3,1)$ using an inverse cdf.

11.3 Generating random samples from simple distributions

We saw in Section 11.2 how we can evaluate $\mathbb{E}[f(X)]$ for different functions f of a random variable X using Monte Carlo integration. The main computational challenge is to efficiently generate samples from the probability distribution $p^*(\boldsymbol{x})$ (which may be a posterior, $p^*(\boldsymbol{x}) \propto p(\boldsymbol{x}|\mathcal{D})$). In this section, we discuss sampling methods that are suitable for parametric univariate distributions. These can be used as building blocks for sampling from more complex multivariate distributions.

11.3.1 Sampling using the inverse cdf

The simplest method for sampling from a univariate distribution is based on the **inverse probability transform**. Let F be a cdf of some distribution we want to sample from, and let F^{-1} be its inverse. Then we have the following result.

Theorem 11.3.1. *If $U \sim U(0,1)$ is a uniform rv, then $F^{-1}(U) \sim F$.*

Proof.

$$\Pr(F^{-1}(U) \leq x) = \Pr(U \leq F(x)) \quad \text{(applying } F \text{ to both sides)} \tag{11.11}$$

$$= F(x) \quad \text{(because } \Pr(U \leq y) = y) \tag{11.12}$$

where the first line follows since F is a monotonic function, and the second line follows since U is uniform on the unit interval. □

Hence we can sample from any univariate distribution, for which we can evaluate its inverse cdf, as follows: generate a random number $u \sim U(0,1)$ using a **pseudorandom number generator** (see e.g., [Pre+88] for details). Let u represent the height up the y axis. Then "slide along" the x axis until you intersect the F curve, and then "drop down" and return the corresponding x value. This corresponds to computing $x = F^{-1}(u)$. See Figure 11.3 for an illustration.

For example, consider the exponential distribution

$$\text{Expon}(x|\lambda) \triangleq \lambda e^{-\lambda x}\, \mathbb{I}(x \geq 0) \tag{11.13}$$

The cdf is

$$F(x) = 1 - e^{-\lambda x}\, \mathbb{I}(x \geq 0) \tag{11.14}$$

whose inverse is the quantile function

$$F^{-1}(p) = -\frac{\ln(1-p)}{\lambda} \tag{11.15}$$

By the above theorem, if $U \sim \text{Unif}(0,1)$, we know that $F^{-1}(U) \sim \text{Expon}(\lambda)$. So we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using $-\ln(1-u)/\lambda$. (In fact, since $1-U \sim \text{Unif}(0,1)$, we can just use $-\ln(u)/\lambda$.)

11.3.2 Sampling from a Gaussian (Box-Muller method)

In this section, we describe a method to sample from a Gaussian. The idea is we sample uniformly from a unit radius circle, and then use the change of variables formula to derive samples from a spherical 2d Gaussian. This can be thought of as two samples from a 1d Gaussian.

In more detail, sample $z_1, z_2 \in (-1,1)$ uniformly, and then discard pairs that do not satisfy $z_1^2 + z_2^2 \leq 1$. The result will be points uniformly distributed inside the unit circle, so $p(\boldsymbol{z}) = \frac{1}{\pi}\mathbb{I}(z \text{ inside circle})$. Now define

$$x_i = z_i \left(\frac{-2\ln r^2}{r^2}\right)^{\frac{1}{2}} \tag{11.16}$$

for $i = 1:2$, where $r^2 = z_1^2 + z_2^2$. Using the multivariate change of variables formula, we have

$$p(x_1, x_2) = p(z_1, z_2)\left|\frac{\partial(z_1, z_2)}{\partial(x_1, x_2)}\right| = \left[\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}x_1^2)\right]\left[\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}x_2^2)\right] \tag{11.17}$$

Hence x_1 and x_2 are two independent samples from a univariate Gaussian. This is known as the **Box-Muller** method.

To sample from a multivariate Gaussian, we first compute the Cholesky decomposition of its covariance matrix, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\mathsf{T}$, where $\mathbf{L}$ is lower triangular. Next we sample $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using the Box-Muller method. Finally we set $\boldsymbol{y} = \mathbf{L}\boldsymbol{x} + \boldsymbol{\mu}$. This is valid since

$$\text{Cov}[\boldsymbol{y}] = \mathbf{L}\text{Cov}[\boldsymbol{x}]\mathbf{L}^\mathsf{T} = \mathbf{L}\,\mathbf{I}\,\mathbf{L}^\mathsf{T} = \boldsymbol{\Sigma} \tag{11.18}$$

11.4 Rejection sampling

Suppose we want to sample from the **target distribution**

$$p(\boldsymbol{x}) = \tilde{p}(\boldsymbol{x})/Z_p \tag{11.19}$$

where $\tilde{p}(\boldsymbol{x})$ is the unnormalized version, and

$$Z_p = \int \tilde{p}(\boldsymbol{x})\, d\boldsymbol{x} \tag{11.20}$$

is the (possibly unknown) normalization constant. One of the simplest approaches to this problem is **rejection sampling**, which we now explain.

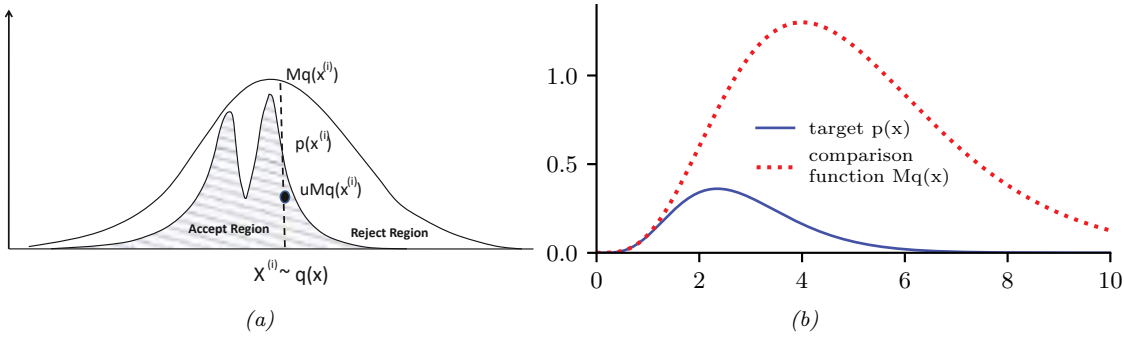


Figure 11.4: (a) Schematic illustration of rejection sampling. From Figure 2 of [And+03]. Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\text{Ga}(\alpha = 5.7, \lambda = 2)$ *distribution (solid blue) using a proposal of the form* $M\text{Ga}(k, \lambda - 1)$ *(dotted red), where* $k = \lfloor 5.7 \rfloor = 5$*. The curves touch at* $\alpha - k = 0.7$*. Generated by rejection_ sampling_ demo.ipynb.*

11.4.1 Basic idea

In rejection sampling, we require access to a **proposal distribution** $q(\boldsymbol{x})$ which satisfies $Cq(\boldsymbol{x}) \geq \tilde{p}(\boldsymbol{x})$, for some constant C. The function $Cq(\boldsymbol{x})$ provides an upper envelope for $\tilde{p}$.

We can use the proposal distribution to generate samples from the target distribution as follows. We first sample $\boldsymbol{x}_0 \sim q(\boldsymbol{x})$, which corresponds to picking a random $\boldsymbol{x}$ location, and then we sample $u_0 \sim \text{Unif}(0, Cq(\boldsymbol{x}_0))$, which corresponds to picking a random height (y location) under the envelope. If $u_0 > \tilde{p}(\boldsymbol{x}_0)$, we reject the sample, otherwise we accept it. This process is illustrated in 1d in Figure 11.4(a): the acceptance region is shown shaded, and the rejection region is the white region between the shaded zone and the upper envelope.

We now prove this procedure is correct. First note that the probability of any given sample $\boldsymbol{x}_0$ being accepted equals the probability of a sample $u_0 \sim \text{Unif}(0, Cq(\boldsymbol{x}_0))$ being less than or equal to $\tilde{p}(\boldsymbol{x}_0)$, i.e.,

$$q(\text{accept}|\boldsymbol{x}_0) = \int_0^{\tilde{p}(\boldsymbol{x}_0)} \frac{1}{Cq(\boldsymbol{x}_0)}\, du = \frac{\tilde{p}(\boldsymbol{x}_0)}{Cq(\boldsymbol{x}_0)} \tag{11.21}$$

Therefore

$$q(\text{propose and accept } \boldsymbol{x}_0) = q(\boldsymbol{x}_0)q(\text{accept}|\boldsymbol{x}_0) = q(\boldsymbol{x}_0)\frac{\tilde{p}(\boldsymbol{x}_0)}{Cq(\boldsymbol{x}_0)} = \frac{\tilde{p}(\boldsymbol{x}_0)}{C} \tag{11.22}$$

Integrating both sides give

$$\int q(\boldsymbol{x}_0)q(\text{accept}|\boldsymbol{x}_0)\, d\boldsymbol{x}_0 = q(\text{accept}) = \frac{\int \tilde{p}(\boldsymbol{x}_0)\, d\boldsymbol{x}_0}{C} = \frac{Z_p}{C} \tag{11.23}$$

Hence we see that the distribution of accepted points is given by the target distribution:

$$q(\boldsymbol{x}_0|\text{accept}) = \frac{q(\boldsymbol{x}_0, \text{accept})}{q(\text{accept})} = \frac{\tilde{p}(\boldsymbol{x}_0)}{C}\frac{C}{Z_p} = \frac{\tilde{p}(\boldsymbol{x}_0)}{Z_p} = p(\boldsymbol{x}_0) \tag{11.24}$$

How efficient is this method? If $\tilde{p}$ is a normalized target distribution, the acceptance probability is $1/C$. Hence we want to choose C as small as possible while still satisfying $Cq(x) \geq \tilde{p}(x)$.

11.4.2 Example

For example, suppose we want to sample from a gamma distribution:[3]

$$\text{Ga}(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} \lambda^{\alpha} \exp(-\lambda x) \tag{11.25}$$

where $\Gamma(\alpha)$ is the gamma function. One can show that if $X_i \overset{iid}{\sim} \text{Expon}(\lambda)$, and $Y = X_1 + \cdots + X_k$, then $Y \sim \text{Ga}(k, \lambda)$. For non-integer shape parameters α, we cannot use this trick. However, we can use rejection sampling using a $\text{Ga}(k, \lambda - 1)$ distribution as a proposal, where $k = \lfloor \alpha \rfloor$. The ratio has the form

$$\frac{p(x)}{q(x)} = \frac{\text{Ga}(x|\alpha, \lambda)}{\text{Ga}(x|k, \lambda - 1)} = \frac{x^{\alpha-1} \lambda^{\alpha} \exp(-\lambda x)/\Gamma(\alpha)}{x^{k-1}(\lambda - 1)^k \exp(-(\lambda - 1)x)/\Gamma(k)} \tag{11.26}$$

$$= \frac{\Gamma(k)\lambda^{\alpha}}{\Gamma(\alpha)(\lambda - 1)^k} x^{\alpha-k} \exp(-x) \tag{11.27}$$

This ratio attains its maximum when $x = \alpha - k$. Hence

$$C = \frac{\text{Ga}(\alpha - k|\alpha, \lambda)}{\text{Ga}(\alpha - k|k, \lambda - 1)} \tag{11.28}$$

See Figure 11.4(b) for a plot.

11.4.3 Adaptive rejection sampling

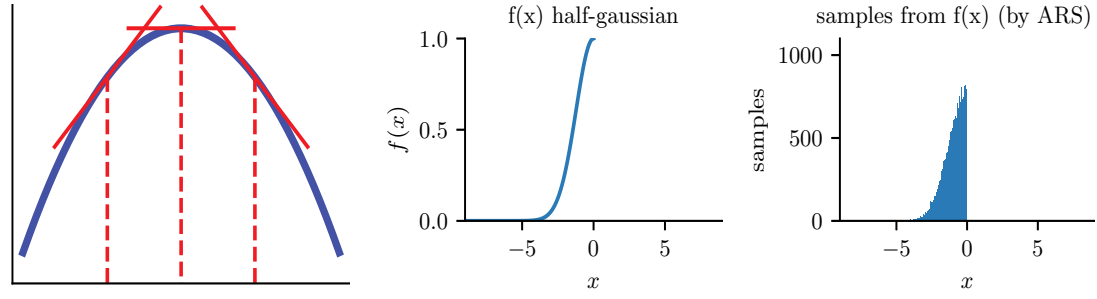
We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to any log concave 1d density $p(x)$. The idea is to upper bound the log density with a piecewise linear function, as illustrated in Figure 11.5(a). We choose the initial locations for the pieces based on a fixed grid over the support of the distribution. We then evaluate the gradient of the log density at these locations, and make the lines be tangent at these points.

Since the log of the envelope is piecewise linear, the envelope itself is piecewise exponential:

$$q(x) = C_i \lambda_i \exp(-\lambda_i (x - x_{i-1})), \quad x_{i-1} < x \leq x_i \tag{11.29}$$

where x_i are the grid points. It is relatively straightforward to sample from this distribution. If the sample x is rejected, we create a new grid point at x, and thereby refine the envelope. As the number of grid points is increased, the tightness of the envelope improves, and the rejection rate goes down. This is known as **adaptive rejection sampling** (ARS) [GW92]. Figure 11.5(b-c) gives an example of the method in action. As with standard rejection sampling, it can be applied to unnormalized distributions.

3. This section is based on notes by Ioana A. Cosma, available at http://users.aims.ac.za/~ioana/cp2.pdf.



(a) (b) (c)

Figure 11.5: (a) Idea behind adaptive rejection sampling. We place piecewise linear upper (and lower) bounds on the log-concave density. Adapted from Figure 1 of [GW92]. Generated by ars_envelope.ipynb. (b-c) Using ARS to sample from a half-Gaussian. Generated by ars_demo.ipynb.

11.4.4 Rejection sampling in high dimensions

It is clear that we want to make our proposal $q(\boldsymbol{x})$ as close as possible to the target distribution $p(\boldsymbol{x})$, while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To see this, consider sampling from $p(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I})$ using as a proposal $q(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \sigma_q^2 \mathbf{I})$. Obviously we must have $\sigma_q^2 \geq \sigma_p^2$ in order to be an upper bound. In D dimensions, the optimum value is given by $C = (\sigma_q/\sigma_p)^D$. The acceptance rate is $1/C$ (since both p and q are normalized), which decreases exponentially fast with dimension. For example, if σ_q exceeds σ_p by just 1%, then in 1000 dimensions the acceptance ratio will be about 1/20,000. This is a fundamental weakness of rejection sampling.

11.5 Importance sampling

In this section, we describe a Monte Carlo method known as **importance sampling** for approximating integrals of the form

$$\mathbb{E}\left[\varphi(\boldsymbol{x})\right] = \int \varphi(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x} \tag{11.30}$$

where φ is called a **target function**, and $\pi(\boldsymbol{x})$ is the **target distribution**, often a conditional distribution of the form $\pi(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{y})$. Since in general it is difficult to draw from the target distribution, we will instead draw from some **proposal distribution** $q(\boldsymbol{x})$ (which will usually depend on $\boldsymbol{y}$). We then adjust for the inaccuracies of this by associating weights with each sample, so we end up with a weighted MC approximation:

$$\mathbb{E}\left[\varphi(\boldsymbol{x})\right] \approx \sum_{n=1}^{N} W_n \varphi(\boldsymbol{x}_n) \tag{11.31}$$

We discuss two cases, first when the target is normalized, and then when it is unnormalized. This will affect the ways the weights are computed, as well as statistical properties of the estimator.

11.5.1 Direct importance sampling

In this section, we assume that we can *evaluate* the normalized target distribution $\pi(\boldsymbol{x})$, but we cannot sample from it. So instead we will sample from the proposal $q(\boldsymbol{x})$. We can then write

$$\int \varphi(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x} = \int \varphi(\boldsymbol{x})\frac{\pi(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \tag{11.32}$$

We require that the proposal be non-zero whenever the target is non-zero, i.e., the support of $q(\boldsymbol{x})$ needs to be greater or equal to the support of $\pi(\boldsymbol{x})$. If we draw N_s samples $\boldsymbol{x}_n \sim q(\boldsymbol{x})$, we can write

$$\mathbb{E}\left[\varphi(\boldsymbol{x})\right] \approx \frac{1}{N_s}\sum_{n=1}^{N_s}\frac{\pi(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)}\varphi(\boldsymbol{x}_n) = \frac{1}{N_s}\sum_{n=1}^{N_s}\tilde{w}_n\varphi(\boldsymbol{x}_n) \tag{11.33}$$

where we have defined the **importance weights** as follows:

$$\tilde{w}_n = \frac{\pi(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)} \tag{11.34}$$

The result is an unbiased estimate of the true mean $\mathbb{E}\left[\varphi(\boldsymbol{x})\right]$.

11.5.2 Self-normalized importance sampling

The disadvantage of direct importance sampling is that we need a way to evaluate the normalized target distribution π in order to compute the weights. It is often much easier to evaluate the **unnormalized target distribution**

$$\tilde{\gamma}(\boldsymbol{x}) = Z\pi(\boldsymbol{x}) \tag{11.35}$$

where

$$Z = \int \tilde{\gamma}(\boldsymbol{x})d\boldsymbol{z} \tag{11.36}$$

is the normalization constant. (For example, if $\pi(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{y})$, then $\tilde{\gamma}(\boldsymbol{x}) = p(\boldsymbol{x}, \boldsymbol{y})$ and $Z = p(\boldsymbol{y})$.) The key idea is to also approximate the normalization constant Z with importance sampling. This method is called **self-normalized importance sampling**. The resulting estimate is a ratio of two estimates, and hence is biased. However as $N_s \to \infty$, the bias goes to zero, under some weak assumptions (see e.g., [RC04] for details).

In more detail, SNIS is based on this approximation:

$$\mathbb{E}\left[\varphi(\boldsymbol{x})\right] = \int \varphi(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x} = \frac{\int \varphi(\boldsymbol{x})\tilde{\gamma}(\boldsymbol{x})d\boldsymbol{x}}{\int \tilde{\gamma}(\boldsymbol{x})d\boldsymbol{x}} = \frac{\int \left[\frac{\tilde{\gamma}(\boldsymbol{x})}{q(\boldsymbol{x})}\varphi(\boldsymbol{x})\right] q(\boldsymbol{x})d\boldsymbol{x}}{\int \left[\frac{\tilde{\gamma}(\boldsymbol{x})}{q(\boldsymbol{x})}\right] q(\boldsymbol{x})d\boldsymbol{x}} \tag{11.37}$$

$$\approx \frac{\frac{1}{N_s}\sum_{n=1}^{N_s}\tilde{w}_n\varphi(\boldsymbol{x}_n)}{\frac{1}{N_s}\sum_{n=1}^{N_s}\tilde{w}_n} \tag{11.38}$$

where we have defined the **unnormalized weights**

$$\tilde{w}_n = \frac{\tilde{\gamma}(\boldsymbol{x}_n)}{q(\boldsymbol{x}_n)} \tag{11.39}$$

We can write Equation (11.38) more compactly as

$$\mathbb{E}\left[\varphi(\boldsymbol{x})\right] \approx \sum_{n=1}^{N_s} W_n \varphi(\boldsymbol{x}_n) \tag{11.40}$$

where we have defined the **normalized weights** by

$$W_n = \frac{\tilde{w}_n}{\sum_{n'=1}^{N_s} \tilde{w}_{n'}} \tag{11.41}$$

This is equivalent to approximating the target distribution using a weighted sum of delta functions:

$$\pi(\boldsymbol{x}) \approx \sum_{n=1}^{N_s} W_n \delta(\boldsymbol{x} - \boldsymbol{x}_n) \triangleq \hat{\pi}(\boldsymbol{x}) \tag{11.42}$$

As a byproduct of this algorithm we get the following appoximation to the normalization constant:

$$Z \approx \frac{1}{N_s} \sum_{n=1}^{N_s} \tilde{w}_n \triangleq \hat{Z} \tag{11.43}$$

11.5.3 Choosing the proposal

The performance of importance sampling depends crucially on the quality of the proposal distribution. As we mentioned, we require that the support of q cover the support of the target (i.e., $\tilde{\gamma}(\boldsymbol{x}) > 0 \implies q(\boldsymbol{x}) > 0$). However, we also want the proposal to not be too "loose" of a "covering". Ideally it should also take into account properties of the target function φ as well, as shown in Figure 11.6. This can yield subsantial benefits, as shown in the "**target aware Bayesian inference**" scheme of [Rai+20]. However, usually the target function φ is unknown or ignored, so we just try to find a "generally useful" approximation to the target.

One way to come up with a good proposal is to learn one, by optimizing the variational lower bound or ELBO (see Section 10.1.1.2). Indeed, if we fix the parameters of the generative model, we can think of importance weighted autoencoders (Section 10.5.1) as learning a good IS proposal. More details on this connection can be found in [DS18].

11.5.4 Annealed importance sampling (AIS)

In this section, we describe a method known as **annealed importance sampling** [Nea01] for sampling from complex, possibly multimodal distributions. Assume we want to sample from some target distribution $p_0(\boldsymbol{x}) \propto f_0(\boldsymbol{x})$ (where $f_0(\boldsymbol{x})$ is the unnormalized version), but we cannot easily do so, because p_0 is complicated in some way (e.g., high dimensional and/or multi-modal). However, suppose that there is an easier distribution which we *can* sample from, call it $p_n(\boldsymbol{x}) \propto f_n(\boldsymbol{x})$; for

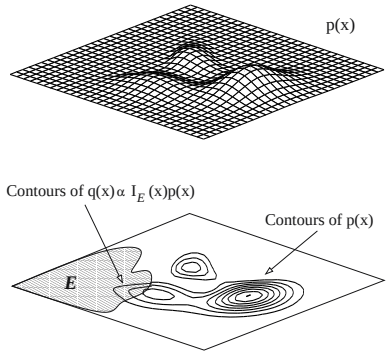


Figure 11.6: In importance sampling, we should sample from a distribution that takes into account regions where $\pi(\boldsymbol{x})$ has high probability and *where $\varphi(\boldsymbol{x})$ is large. Here the function to be evaluated is an indicator function of a set, corresponding to a set of rare events in the tail of the distribution. From Figure 3 of [And+03]. Used with kind permission of Nando de Freitas.*

example, this might be the prior. We now construct a sequence of intermediate distributions than move slowly from p_n to p_0 as follows:

$$f_j(\boldsymbol{x}) = f_0(\boldsymbol{x})^{\beta_j} f_n(\boldsymbol{x})^{1-\beta_j} \tag{11.44}$$

where $1 = \beta_0 > \beta_1 > \cdots > \beta_n = 0$, where β_j is an inverse temperature. We will sample a set of points from f_n, and then from f_{n-1}, and so on, until we eventually sample from f_0.

To sample from each f_j, suppose we can define a Markov chain $T_j(\boldsymbol{x}, \boldsymbol{x}') = p_j(\boldsymbol{x}'|\boldsymbol{x})$, which leaves p_0 invariant (i.e., $\int p_j(\boldsymbol{x}'|\boldsymbol{x})p_0(\boldsymbol{x})d\boldsymbol{x} = p_0(\boldsymbol{x}')$). (See Chapter 12 for details on how to construct such chains.) Given this, we can sample $\boldsymbol{x}$ from p_0 as follows: sample $\boldsymbol{v}_n \sim p_n$; sample $\boldsymbol{v}_{n-1} \sim T_{n-1}(\boldsymbol{v}_n, \cdot)$; and continue in this way until we sample $\boldsymbol{v}_0 \sim T_0(\boldsymbol{v}_1, \cdot)$; finally we set $\boldsymbol{x} = \boldsymbol{v}_0$ and give it weight

$$w = \frac{f_{n-1}(\boldsymbol{v}_{n-1})}{f_n(\boldsymbol{v}_{n-1})} \frac{f_{n-2}(\boldsymbol{v}_{n-2})}{f_{n-1}(\boldsymbol{v}_{n-2})} \cdots \frac{f_1(\boldsymbol{v}_1)}{f_2(\boldsymbol{v}_1)} \frac{f_0(\boldsymbol{v}_0)}{f_1(\boldsymbol{v}_0)} \tag{11.45}$$

This can be shown to be correct by viewing the algorithm as a form of importance sampling in an extended state space $\boldsymbol{v} = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n)$. Consider the following distribution on this state space:

$$p(\boldsymbol{v}) \propto \varphi(\boldsymbol{v}) = f_0(\boldsymbol{v}_0)\tilde{T}_0(\boldsymbol{v}_0, \boldsymbol{v}_1)\tilde{T}_2(\boldsymbol{v}_1, \boldsymbol{v}_2) \cdots \tilde{T}_{n-1}(\boldsymbol{v}_{n-1}, \boldsymbol{v}_n) \tag{11.46}$$

$$\propto p(\boldsymbol{v}_0)p(\boldsymbol{v}_1|\boldsymbol{v}_0) \cdots p(\boldsymbol{v}_n|\boldsymbol{v}_{n-1}) \tag{11.47}$$

where $\tilde{T}_j$ is the reversal of T_j:

$$\tilde{T}_j(\boldsymbol{v}, \boldsymbol{v}') = T_j(\boldsymbol{v}', \boldsymbol{v})p_j(\boldsymbol{v}')/p_j(\boldsymbol{v}) = T_j(\boldsymbol{v}', \boldsymbol{v})f_j(\boldsymbol{v}')/f_j(\boldsymbol{v}) \tag{11.48}$$

It is clear that $\sum_{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n} \varphi(\boldsymbol{v}) = f_0(\boldsymbol{v}_0)$, so by sampling from $p(\boldsymbol{v})$, we can effectively sample from $p_0(\boldsymbol{x})$.

We can sample on this extended state space using the above algorithm, which corresponds to the following proposal:

$$q(\boldsymbol{v}) \propto g(\boldsymbol{v}) = f_n(\boldsymbol{v}_n)T_{n-1}(\boldsymbol{v}_n, \boldsymbol{v}_{n-1}) \cdots T_2(\boldsymbol{v}_2, \boldsymbol{v}_1)T_0(\boldsymbol{v}_1, \boldsymbol{v}_0) \tag{11.49}$$

$$\propto p(\boldsymbol{v}_n)p(\boldsymbol{v}_{n-1}|\boldsymbol{v}_n) \cdots p(\boldsymbol{v}_1|\boldsymbol{v}_0) \tag{11.50}$$

One can show that the importance weights $w = \frac{\varphi(\boldsymbol{v}_0,\ldots,\boldsymbol{v}_n)}{g(\boldsymbol{v}_0,\ldots,\boldsymbol{v}_n)}$ are given by Equation (11.45). Since marginals of the sampled sequences from this extended model are equivalent to samples from $p_0(\boldsymbol{x})$, we see that we are using the correct weights.

11.5.4.1 Estimating normalizing constants using AIS

An important application of AIS is to evaluate a ratio of partition functions. Notice that $Z_0 = \int f_0(\boldsymbol{x})d\boldsymbol{x} = \int \varphi(\boldsymbol{v})d\boldsymbol{v}$, and $Z_n = \int f_n(\boldsymbol{x})d\boldsymbol{x} = \int g(\boldsymbol{v})d\boldsymbol{v}$. Hence

$$\frac{Z_0}{Z_n} = \frac{\int \varphi(\boldsymbol{v})d\boldsymbol{v}}{\int g(\boldsymbol{v})d\boldsymbol{v}} = \frac{\int \frac{\varphi(\boldsymbol{v})}{g(\boldsymbol{v})} g(\boldsymbol{v})d\boldsymbol{v}}{\int g(\boldsymbol{v})d\boldsymbol{v}} = \mathbb{E}_g\left[\frac{\varphi(\boldsymbol{v})}{g(\boldsymbol{v})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} w_s \tag{11.51}$$

where $w_s = \varphi(\boldsymbol{v}_s)/g(\boldsymbol{v}_s)$. If f_0 is a prior and f_n is the posterior, we can estimate $Z_n = p(\mathcal{D})$ using the above equation, provided the prior has a known normalization constant Z_0. This is generally considered the method of choice for evaluating difficult partition functions. See e.g., [GM98] for more details.

11.6 Controlling Monte Carlo variance

As we mentioned in Section 11.2.2, the standard error in a Monte Carlo estimate is $O(1/\sqrt{S})$, where S is the number of (independent) samples. Consequently it may take many samples to reduce the variance to a sufficiently small value. In this section, we discuss some ways to reduce the variance of sampling methods. For more details, see e.g., [KTB11].

11.6.1 Common random numbers

When performing Monte Carlo optimization, we often want to compare $\mathbb{E}_{p(\boldsymbol{z})}[f(\boldsymbol{\theta}, \boldsymbol{z})]$ to $\mathbb{E}_{p(\boldsymbol{z})}[f(\boldsymbol{\theta}', \boldsymbol{z})]$ for different values of the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. To reduce the variance of this comparison, we can use the same random samples $\boldsymbol{z}_s$ for evaluating both functions. In this way, differences in the outcome can be ascribed to differences in the parameters $\boldsymbol{\theta}$, rather than to the noise terms. This is called the **common random numbers** trick, and is widely used in ML (see e.g., [GBJ18; NJ00]), since it can often convert a stochastic optimization problem into a deterministic one, enabling the us of more powerful optimization methods. For more details on CRN, see e.g., https://en.wikipedia.org/wiki/Variance_reduction#Common_Random_Numbers_(CRN).

11.6.2 Rao-Blackwellization

In this section, we discuss a useful technique for reducing the variance of MC estimators known as **Rao-Blackwellization**. To explain the method, suppose we have two rv's, X and Y, and we want

to estimate $\overline{f} = \mathbb{E}[f(X,Y)]$. The naive approach is to use an MC approximation

$$\hat{f}_{MC} = \frac{1}{S}\sum_{s=1}^{S} f(X_s, Y_s) \tag{11.52}$$

where $(X_s, Y_s) \sim p(X,Y)$. This is an unbiased estimator of $\overline{f}$. However, it may have high variance.

Now suppose we can analytically marginalize out Y, provided we know X, i.e., we can tractably compute

$$f_X(X_s) = \int dY p(Y|X_s) f(X_s, Y) = \mathbb{E}[f(X,Y)|X = X_s] \tag{11.53}$$

Let us define the Rao-Blackwellized estimator

$$\hat{f}_{RB} = \frac{1}{S}\sum_{s=1}^{S} f_X(X_s) \tag{11.54}$$

where $X_s \sim p(X)$. This is an unbiased estimator, since $\mathbb{E}\left[\hat{f}_{RB}\right] = \mathbb{E}[\mathbb{E}[f(X,Y)|X]] = \overline{f}$. However, this estimate can have lower variance than the naive estimator. The intuitive reason is that we are now sampling in a reduced dimensional space. Formally we can see this by using the law of iterated variance to get

$$\mathbb{V}[\mathbb{E}[f(X,Y)|X]] = \mathbb{V}[f(X,Y)] - \mathbb{E}[\mathbb{V}[f(X,Y)]|X] \leq \mathbb{V}[f(X,Y)] \tag{11.55}$$

For some examples of this in practice, see Section 6.3.4.2, Section 13.4, and Section 12.3.8.

11.6.3 Control variates

Suppose we want to estimate $\mu = \mathbb{E}[f(X)]$ using an unbiased estimator $m(\mathcal{X}) = \frac{1}{S}\sum_{s=1}^{S} m(x_s)$, where $x_s \sim p(X)$ and $\mathbb{E}[m(X)] = \mu$. (We abuse notation slightly and use m to refer to a function of a single random variable as well as a set of samples.) Now consider the alternative estimator

$$m^*(\mathcal{X}) = m(\mathcal{X}) + c\,(b(\mathcal{X}) - \mathbb{E}[b(\mathcal{X})]) \tag{11.56}$$

This is called a **control variate**, and b is called a **baseline**. (Once again we abuse notation and use $b(\mathcal{X}) = \frac{1}{S}\sum_{s=1}^{S} b(x_s)$ and $m^*(\mathcal{X}) = \frac{1}{S}\sum_{s=1}^{S} m^*(x_s)$.)

It is easy to see that $m^*(\mathcal{X})$ is an unbiased estimator, since $\mathbb{E}[m^*(X)] = \mathbb{E}[m(X)] = \mu$. However, it can have lower variance, provided b is correlated with m. To see this, note that

$$\mathbb{V}[m^*(X)] = \mathbb{V}[m(X)] + c^2\mathbb{V}[b(X)] + 2c\mathrm{Cov}[m(X), b(X)] \tag{11.57}$$

By taking the derivative of $\mathbb{V}[m^*(X)]$ wrt c and setting to 0, we find that the optimal value is

$$c^* = -\frac{\mathrm{Cov}[m(X), b(X)]}{\mathbb{V}[b(X)]} \tag{11.58}$$

The corresponding variance of the new estimator is now

$$\mathbb{V}\left[m^*(X)\right] = \mathbb{V}\left[m(X)\right] - \frac{\mathrm{Cov}\left[m(X), b(X)\right]^2}{\mathbb{V}\left[b(X)\right]} = (1 - \rho_{m,b}^2)\mathbb{V}\left[m(X)\right] \leq \mathbb{V}\left[m(X)\right] \tag{11.59}$$

where $\rho_{m,b}^2$ is the correlation of the basic estimator and the baseline function. If we can ensure this correlation is high, we can reduce the variance. Intuitively, the CV estimator is exploiting information about the errors in the estimate of a known quantity, namely $\mathbb{E}\left[b(X)\right]$, to reduce the errors in estimating the unknown quantity, namely μ.

We give a simple worked example in Section 11.6.3.1. See Section 10.2.3 for an example of this technique applied to blackbox variational inference.

11.6.3.1 Example

We now give a simple worked example of control variates.[4] Consider estimating $\mu = \mathbb{E}\left[f(X)\right]$ where $f(X) = 1/(1+X)$ and $X \sim \mathrm{Unif}(0,1)$. The exact value is

$$\mu = \int_0^1 \frac{1}{1+x} dx = \ln 2 \approx 0.693 \tag{11.60}$$

The naive MC estimate, using S samples, is $m(\mathcal{X}) = \frac{1}{S}\sum_{s=1}^{S} f(x_s)$. Using $S = 1500$, we find $\mathbb{E}\left[m(\mathcal{X})\right] = 0.6935$ with standard error $\mathrm{se} = 0.0037$.

Now let us use $b(X) = 1 + X$ as a baseline, so $b(\mathcal{X}) = (1/S)\sum_s (1 + x_s)$. This has expectation $\mathbb{E}\left[b(X)\right] = \int_0^1 (1+x)dx = \frac{3}{2}$. The control variate estimator is given by

$$m^*(\mathcal{X}) = \frac{1}{S}\sum_{s=1}^{S} f(x_s) + c\left(\frac{1}{S}\sum_{s=1}^{S} b(x_s) - \frac{3}{2}\right) \tag{11.61}$$

The optimal value can be estimated from the samples of $m(x_s)$ and $b(x_s)$, and plugging into Equation (11.58) to get $c^* \approx 0.4773$. Using $S = 1500$, we find $\mathbb{E}\left[m^*(\mathcal{X})\right] = 0.6941$ and $\mathrm{se} = 0.0007$.

See also Section 11.6.4.1, where we analyze this example using antithetic sampling.

11.6.4 Antithetic sampling

In this section, we discuss **antithetic sampling**, which is a simple way to reduce variance.[5] Suppose we want to estimate $\theta = \mathbb{E}\left[Y\right]$. Let Y_1 and Y_2 be two samples. An unbiased estimate of θ is given by $\hat{\theta} = (Y_1 + Y_2)/2$. The variance of this estimate is

$$\mathbb{V}\left[\hat{\theta}\right] = \frac{\mathbb{V}\left[Y_1\right] + \mathbb{V}\left[Y_2\right] + 2\mathrm{Cov}\left[Y_1, Y_2\right]}{4} \tag{11.62}$$

so the variance is reduced if $\mathrm{Cov}\left[Y_1, Y_2\right] < 0$. So whenever we sample Y_1, we should set Y_2 to be its "opposite", but with the same mean.

For example, suppose $Y \sim \mathrm{Unif}(0,1)$. If we let $y_1, \ldots, y_n$ be iid samples from $\mathrm{Unif}(0,1)$, then we can define $y_i' = 1 - y_i$. The distribution of y_i' is still $\mathrm{Unif}(0,1)$, but $\mathrm{Cov}\left[y_i, y_i'\right] < 1$.

4. The example is from https://en.wikipedia.org/wiki/Control_variates, with modified notation. See control_variates.ipynb for some code.

5. Our presentation is based on https://en.wikipedia.org/wiki/Antithetic_variates. See antithetic_sampling.ipynb for the code.

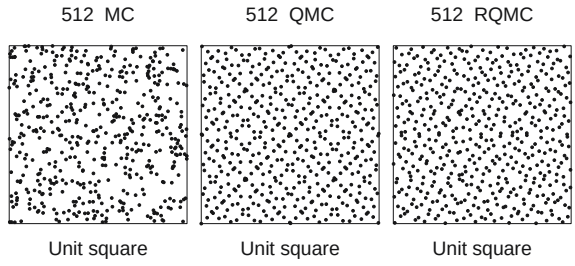


Figure 11.7: Illustration of Monte Carlo (MC), Quasi-MC (QMC) from a Sobol sequence, and randomized QMC using a scrambling method. Adapted from Figure 1 of [OR20]. Used with kind permission of Art Owen.

11.6.4.1 Example

To see why this can be useful, consider the example from Section 11.6.3.1. Let $\hat{\mu}_{\text{mc}}$ be the classic MC estimate using $2N$ samples from $\text{Unif}(0,1)$, and let $\hat{\mu}_{\text{anti}}$ be the MC estimate using the above antithetic sampling scheme applied to N base samples from $\text{Unif}(0,1)$. The exact value is $\mu = \ln 2 \approx 0.6935$. For the classical method, with $N = 750$, we find $\mathbb{E}[\hat{\mu}_{\text{mc}}] = 0.69365$ with a standard error of 0.0037. For the antithetic method, we find $\mathbb{E}[\hat{\mu}_{\text{anti}}] = 0.6939$ with a standard error of 0.0007, which matches the control variate method of Section 11.6.3.1.

11.6.5 Quasi-Monte Carlo (QMC)

Quasi-Monte Carlo (see e.g., [Lem09; Owe13]) is an approach to numerical integration that replaces random samples with **low discrepancy sequences**, such as the **Halton sequence** (see e.g., [Owe17]) or **Sobol sequence**. Intuitively, these are **space filling** sequences of points, constructed to reduce the unwanted gaps and clusters that would arise among randomly chosen inputs. See Figure 11.7 for an example.[6]

More precisely, consider the problem of evaluating the following D-dimensional integral:

$$\overline{f} = \int_{[0,1]^D} f(\boldsymbol{x}) d\boldsymbol{x} \approx \hat{f}_N = \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x}_n) \tag{11.63}$$

Let $\epsilon_N = |\overline{f} - \hat{f}_N|$ be the error. In standard Monte Carlo, if we draw N independent samples, then we have $\epsilon_N \sim O\left(\frac{1}{\sqrt{N}}\right)$. In QMC, it can be shown that $\epsilon_N \sim O\left(\frac{(\log N)^D}{N}\right)$. For $N > 2^D$, the latter is smaller than the former.

One disadvantage of QMC is that it just provides a point estimate of $\overline{f}$, and does not give an uncertainty estimate. By contrast, in regular MC, we can estimate the MC standard error, discussed in Section 11.2.2. **Randomized QMC** (see e.g., [L'E18]) provides a solution to this problem. The basic idea is to repeat the QMC method R times, by perturbing the sequence of N points by a

6. More details on QMC can be found at http://roth.cs.kuleuven.be/wiki/Main_Page. For connections to Bayesian quadrature, see e.g., [DKS13; HKO22].

random amount. In particular, define

$$\boldsymbol{y}_{i,r} = \boldsymbol{x}_i + \boldsymbol{u}_r \pmod 1 \tag{11.64}$$

where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ is a low-discrepancy sequence, and $\boldsymbol{u}_r \sim \text{Unif}(0,1)^D$ is a random perturbation. The set $\{\boldsymbol{y}_j\}$ is low discrepancy, and satisfies that each $\boldsymbol{y}_j \sim \text{Unif}(0,1)^D$, for $j = 1 : N \times R$. This has much lower variance than standard MC. (Typically we take R to be a power of 2.) Recently, [OR20] proved a strong law of large numbers for RQMC.

QMC and RQMC can be used inside of MCMC inference (see e.g., [OT05]) and variational inference (see e.g., [BWM18]). It is also commonly used to select the initial set of query points for Bayesian optimization (Section 6.6).

Another technique that can be used is **orthogonal Monte Carlo**, where the samples are conditioned to be pairwise orthogonal, but with the marginal distributions matching the original ones (see e.g., [Lin+20]).

12 Markov chain Monte Carlo

12.1 Introduction

In Chapter 11, we considered non-iterative Monte Carlo methods, including rejection sampling and importance sampling, which generate independent samples from some target distribution. The trouble with these methods is that they often do not work well in high dimensional spaces. In this chapter, we discuss a popular method for sampling from high-dimensional distributions known as **Markov chain Monte Carlo** or **MCMC**. In a survey by *SIAM News*[1], MCMC was placed in the top 10 most important algorithms of the 20th century.

The basic idea behind MCMC is to construct a Markov chain (Section 2.6) on the state space $\mathcal{X}$ whose stationary distribution is the target density $p^*(\boldsymbol{x})$ of interest. (In a Bayesian context, this is usually a posterior, $p^*(\boldsymbol{x}) \propto p(\boldsymbol{x}|\mathcal{D})$, but MCMC can be applied to generate samples from any kind of distribution.) That is, we perform a random walk on the state space, in such a way that the fraction of time we spend in each state $\boldsymbol{x}$ is proportional to $p^*(\boldsymbol{x})$. By drawing (correlated) samples $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, from the chain, we can perform Monte Carlo integration wrt p^*.

Note that the initial samples from the chain do not come from the stationary distribution, and should be discarded; the amount of time it takes to reach stationarity is called the **mixing time** or **burn-in time**; reducing this is one of the most important factors in making the algorithm fast, as we will see.

The MCMC algorithm has an interesting history. It was discovered by physicists working on the atomic bomb at Los Alamos during World War II, and was first published in the open literature in [Met+53] in a chemistry journal. An extension was published in the statistics literature in [Has70], but was largely unnoticed. A special case (Gibbs sampling, Section 12.3) was independently invented in [GG84] in the context of Ising models (Section 4.3.2.1). But it was not until [GS90] that the algorithm became well-known to the wider statistical community. Since then it has become wildly popular in Bayesian statistics, and is becoming increasingly popular in machine learning.

In the rest of this chapter, we give a brief introduction to MCMC methods. For more details on the theory, see e.g., [GRS96; BZ20]. For more details on the implementation side, see e.g., [Lao+20]. And for an interactive visualization of many of these algorithsm in 2d, see http://chi-feng.github.io/mcmc-demo/app.html.

1. Source: http://www.siam.org/pdf/news/637.pdf.

12.2 Metropolis-Hastings algorithm

In this section, we describe the simplest kinds of MCMC algorithm known as the **Metropoli-Hastings** or **MH** algorithm.

12.2.1 Basic idea

The basic idea in MH is that at each step, we propose to move from the current state $\boldsymbol{x}$ to a new state $\boldsymbol{x}'$ with probability $q(\boldsymbol{x}'|\boldsymbol{x})$, where q is called the **proposal distribution** (also called the **kernel**). The user is free to use any kind of proposal they want, subject to some conditions which we explain below. This makes MH quite a flexible method.

Having proposed a move to $\boldsymbol{x}'$, we then decide whether to **accept** this proposal, or to reject it, according to some formula, which ensures that the long-term fraction of time spent in each state is proportional to $p^*(\boldsymbol{x})$. If the proposal is accepted, the new state is $\boldsymbol{x}'$, otherwise the new state is the same as the current state, $\boldsymbol{x}$ (i.e., we repeat the sample).

If the proposal is symmetric, so $q(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}|\boldsymbol{x}')$, the acceptance probability is given by the following formula:

$$A = \min\left(1, \frac{p^*(\boldsymbol{x}')}{p^*(\boldsymbol{x})}\right) \tag{12.1}$$

We see that if $\boldsymbol{x}'$ is more probable than $\boldsymbol{x}$, we definitely move there (since $\frac{p^*(\boldsymbol{x}')}{p^*(\boldsymbol{x})} > 1$), but if $\boldsymbol{x}'$ is less probable, we may still move there anyway, depending on the relative probabilities. So instead of greedily moving to only more probable states, we occasionally allow "downhill" moves to less probable states. In Section 12.2.2, we prove that this procedure ensures that the fraction of time we spend in each state $\boldsymbol{x}$ is equal to $p^*(\boldsymbol{x})$.

If the proposal is asymmetric, so $q(\boldsymbol{x}'|\boldsymbol{x}) \neq q(\boldsymbol{x}|\boldsymbol{x}')$, we need the **Hastings correction**, given by the following:

$$A = \min(1, \alpha) \tag{12.2}$$

$$\alpha = \frac{p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})} = \frac{p^*(\boldsymbol{x}')/q(\boldsymbol{x}'|\boldsymbol{x})}{p^*(\boldsymbol{x})/q(\boldsymbol{x}|\boldsymbol{x}')} \tag{12.3}$$

This correction is needed to compensate for the fact that the proposal distribution itself (rather than just the target distribution) might favor certain states.

An important reason why MH is a useful algorithm is that, when evaluating α, we only need to know the target density up to a normalization constant. In particular, suppose $p^*(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x})$, where $\tilde{p}(\boldsymbol{x})$ is an unnormalized distribution and Z is the normalization constant. Then

$$\alpha = \frac{(\tilde{p}(\boldsymbol{x}')/Z)\ q(\boldsymbol{x}|\boldsymbol{x}')}{(\tilde{p}(\boldsymbol{x})/Z)\ q(\boldsymbol{x}'|\boldsymbol{x})} \tag{12.4}$$

so the Z's cancel. Hence we can sample from p^* even if Z is unknown.

A proposal distribution q is valid or admissible if it "covers" the support of the target. Formally, we can write this as

$$\text{supp}(p^*) \subseteq \cup_x \text{supp}(q(\cdot|x)) \tag{12.5}$$

With this, we can state the overall algorithm as in Algorithm 12.1.

Algorithm 12.1: Metropolis-Hastings algorithm

Initialize x^0
for $s = 0, 1, 2, \ldots$ **do**
  Define $x = x^s$
  Sample $x' \sim q(x'|x)$
  Compute acceptance probability

$$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$$

  Compute $A = \min(1, \alpha)$
  Sample $u \sim U(0, 1)$
  Set new sample to

$$x^{s+1} = \begin{cases} x' & \text{if } u \leq A \text{ (accept)} \\ x^s & \text{if } u > A \text{ (reject)} \end{cases}$$

12.2.2 Why MH works

To prove that the MH procedure generates samples from p^*, we need a bit of Markov chain theory, as discussed in Section 2.6.4.

The MH algorithm defines a Markov chain with the following transition matrix:

$$p(\boldsymbol{x}'|\boldsymbol{x}) = \begin{cases} q(\boldsymbol{x}'|\boldsymbol{x})A(\boldsymbol{x}'|\boldsymbol{x}) & \text{if } \boldsymbol{x}' \neq \boldsymbol{x} \\ q(\boldsymbol{x}|\boldsymbol{x}) + \sum_{\boldsymbol{x}' \neq \boldsymbol{x}} q(\boldsymbol{x}'|\boldsymbol{x})(1 - A(\boldsymbol{x}'|\boldsymbol{x})) & \text{otherwise} \end{cases} \tag{12.6}$$

This follows from a case analysis: if you move to $\boldsymbol{x}'$ from $\boldsymbol{x}$, you must have proposed it (with probability $q(\boldsymbol{x}'|\boldsymbol{x})$) and it must have been accepted (with probability $A(\boldsymbol{x}'|\boldsymbol{x})$); otherwise you stay in state $\boldsymbol{x}$, either because that is what you proposed (with probability $q(\boldsymbol{x}|\boldsymbol{x})$), or because you proposed something else (with probability $q(\boldsymbol{x}'|\boldsymbol{x})$) but it was rejected (with probability $1 - A(\boldsymbol{x}'|\boldsymbol{x})$).

Let us analyze this Markov chain. Recall that a chain satisfies **detailed balance** if

$$p(\boldsymbol{x}'|\boldsymbol{x})p^*(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{x}')p^*(\boldsymbol{x}') \tag{12.7}$$

This means in the in-flow to state $\boldsymbol{x}'$ from $\boldsymbol{x}$ is equal to the out-flow from state $\boldsymbol{x}'$ back to $\boldsymbol{x}$, and vice versa. We also showed that if a chain satisfies detailed balance, then p^* is its stationary distribution. Our goal is to show that the MH algorithm defines a transition function that satisfies detailed balance and hence that p^* is its stationary distribution. (If Equation (12.7) holds, we say that p^* is an **invariant** distribution wrt the Markov transition kernel q.)

Theorem 12.2.1. *If the transition matrix defined by the MH algorithm (given by Equation (12.6)) is ergodic and irreducible, then p^* is its unique limiting distribution.*

Proof. Consider two states $\boldsymbol{x}$ and $\boldsymbol{x}'$. Either

$$p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x}) < p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}') \tag{12.8}$$

or

$$p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x}) \geq p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}') \tag{12.9}$$

Without loss of generality, assume that $p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x}) > p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')$. Hence

$$\alpha(\boldsymbol{x}'|\boldsymbol{x}) = \frac{p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})} < 1 \tag{12.10}$$

Hence we have $A(\boldsymbol{x}'|\boldsymbol{x}) = \alpha(\boldsymbol{x}'|\boldsymbol{x})$ and $A(\boldsymbol{x}|\boldsymbol{x}') = 1$.

Now to move from $\boldsymbol{x}$ to $\boldsymbol{x}'$ we must first propose $\boldsymbol{x}'$ and then accept it. Hence

$$p(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}'|\boldsymbol{x})A(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}'|\boldsymbol{x})\frac{p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}')}{p^*(\boldsymbol{x})q(\boldsymbol{x}'|\boldsymbol{x})} = \frac{p^*(\boldsymbol{x}')}{p^*(\boldsymbol{x})}q(\boldsymbol{x}|\boldsymbol{x}') \tag{12.11}$$

Hence

$$p^*(\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x}) = p^*(\boldsymbol{x}')q(\boldsymbol{x}|\boldsymbol{x}') \tag{12.12}$$

The backwards probability is

$$p(\boldsymbol{x}|\boldsymbol{x}') = q(\boldsymbol{x}|\boldsymbol{x}')A(\boldsymbol{x}|\boldsymbol{x}') = q(\boldsymbol{x}|\boldsymbol{x}') \tag{12.13}$$

since $A(\boldsymbol{x}|\boldsymbol{x}') = 1$. Inserting this into Equation (12.12) we get

$$p^*(\boldsymbol{x})p(\boldsymbol{x}'|\boldsymbol{x}) = p^*(\boldsymbol{x}')p(\boldsymbol{x}|\boldsymbol{x}') \tag{12.14}$$

so detailed balance holds wrt p^*. Hence, from Theorem 2.6.3, p^* is a stationary distribution. Furthermore, from Theorem 2.6.2, this distribution is unique, since the chain is ergodic and irreducible. □

12.2.3 Proposal distributions

In this section, we discuss some common proposal distributions. Note, however, that good proposal design is often intimately dependent on the form of the target distribution (most often the posterior).

12.2.3.1 Independence sampler

If we use a proposal of the form $q(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}')$, where the new state is independent of the old state, we get a method known as the **independence sampler**, which is similar to importance sampling (Section 11.5). The function $q(\boldsymbol{x}')$ can be any suitable distribution, such as a Gaussian. This has non-zero probability density on the entire state space, and hence is a valid proposal for any unconstrained continuous state space.

12.2.3.2 Random walk Metropolis (RWM) algorithm

The **random walk Metropolis** algorithm corresponds to MH with the following proposal distribution:

$$q(\boldsymbol{x}'|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}'|\boldsymbol{x}, \tau^2\mathbf{I}) \tag{12.15}$$

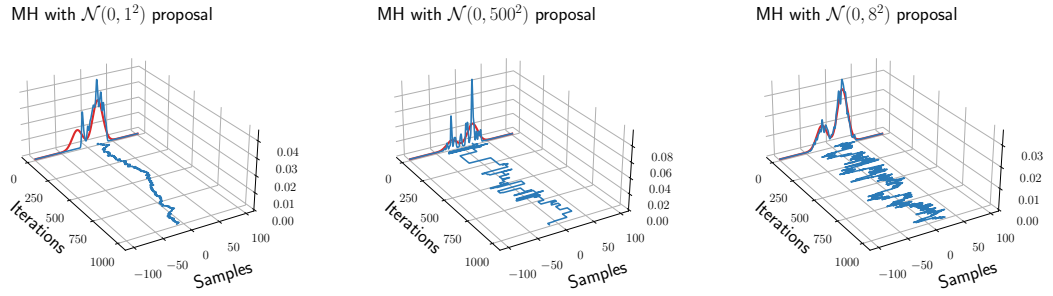


Figure 12.1: An example of the Metropolis-Hastings algorithm for sampling from a mixture of two 1d Gaussians ($\boldsymbol{\mu} = (-20, 20)$, $\boldsymbol{\pi} = (0.3, 0.7)$, $\boldsymbol{\Sigma} = (100, 100)$), using a Gaussian proposal with standard deviation of $\tau \in \{1, 8, 500\}$. (a) When $\tau = 1$, the chain gets trapped near the starting state and fails to sample from the mode at $\mu = -20$. (b) When $\tau = 500$, the chain is very "sticky", so its effective sample size is low (as reflected by the rough histogram approximation at the end). (c) Using a variance of $\tau = 8$ is just right and leads to a good approximation of the true distribution (shown in red). Compare to Figure 12.4. Generated by mcmc_gmm_demo.ipynb.

Here τ is a scale factor chosen to facilitate rapid mixing. [RR01b] prove that, if the posterior is Gaussian, the asymptotically optimal value is to use $\tau^2 = 2.38^2/D$, where D is the dimensionality of $\boldsymbol{x}$; this results in an acceptance rate of 0.234, which (in this case) is the optimal tradeoff between exploring widely enough to cover the distribution without being rejected too often. (See [Béd08] for a more recent account of optimal acceptance rates for random walk Metropolis methods.)

Figure 12.1 shows an example where we use RWM to sample from a mixture of two 1D Gaussians. This is a somewhat tricky target distribution, since it consists of two somewhat separated modes. It is very important to set the variance of the proposal τ^2 correctly: if the variance is too low, the chain will only explore one of the modes, as shown in Figure 12.1(a), but if the variance is too large, most of the moves will be rejected, and the chain will be very **sticky**, i.e., it will stay in the same state for a long time. This is evident from the long stretches of repeated values in Figure 12.1(b). If we set the proposal's variance just right, we get the trace in Figure 12.1(c), where the samples clearly explore the support of the target distribution.

12.2.3.3 Composing proposals

If there are several proposals that might be useful, one can combine them using a **mixture proposal**, which is a convex combination of base proposals:

$$q(\boldsymbol{x}'|\boldsymbol{x}) = \sum_{k=1}^{K} w_k q_k(\boldsymbol{x}'|\boldsymbol{x}) \tag{12.16}$$

where w_k are the mixing weights that sum to one. As long as each q_k is an individually valid proposal, and each $w_k > 0$, then the overall mixture proposal will also be valid. In particular, if each proposal is reversible, so it satisfies detailed balance (Section 2.6.4.4), then so does the mixture.

It is also possible to compose individual proposals by chaining them together to get

$$q(\boldsymbol{x}'|\boldsymbol{x}) = \sum_{\boldsymbol{x}_1} \cdots \sum_{\boldsymbol{x}_{K-1}} q_1(\boldsymbol{x}_1|\boldsymbol{x}) q_2(\boldsymbol{x}_2|\boldsymbol{x}_1) \cdots q_K(\boldsymbol{x}|\boldsymbol{x}_{K-1}) \tag{12.17}$$

A common example is where each base proposal only updates a subset of the variables (see e.g., Section 12.3).

12.2.3.4 Data-driven MCMC

In the case where the target distribution is a posterior, $p^*(\boldsymbol{x}) = p(\boldsymbol{x}|\mathcal{D})$, it is helpful to condition the proposal not just on the previous hidden state, but also the visible data, i.e., to use $q(\boldsymbol{x}'|\boldsymbol{x}, \mathcal{D})$. This is called **data-driven MCMC** (see e.g., [TZ02; Jih+12]).

One way to create such a proposal is to train a recognition network to propose states using $q(\boldsymbol{x}'|\boldsymbol{x}, \mathcal{D}) = f(\boldsymbol{x})$. If the state space is high-dimensional, it might be hard to predict all the hidden components, so we can alternatively train individual "experts" to predict specific pieces of the hidden state. For example, in the context of estimating the 3d pose of a person from an image, we might combine a face detector with a limb detector. We can then use a mixture proposal of the form

$$q(\boldsymbol{x}'|\boldsymbol{x}, \mathcal{D}) = \pi_0 q_0(\boldsymbol{x}'|\boldsymbol{x}) + \sum_k \pi_k q_k(x'_k|f_k(\mathcal{D})) \tag{12.18}$$

where q_0 is a standard data-independent proposal (e.g., random walk), and q_k updates the k'th component of the state space.

The overall procedure is a form of **generate and test**: the discriminative proposals $q(\boldsymbol{x}'|\boldsymbol{x}, \mathcal{D})$ generate new hypotheses, which are then "tested" by computing the posterior ratio $\frac{p(\boldsymbol{x}'|\mathcal{D})}{p(\boldsymbol{x}|\mathcal{D})}$, to see if the new hypothesis is better or worse. (See also Section 13.3, where we discuss learning proposal distributions for particle filters.)

12.2.3.5 Adaptive MCMC

One can change the parameters of the proposal as the algorithm is running to increase efficiency. This is called **adaptive MCMC**. This allows one to start with a broad covariance (say), allowing large moves through the space until a mode is found, followed by a narrowing of the covariance to ensure careful exploration of the region around the mode.

However, one must be careful not to violate the Markov property; thus the parameters of the proposal should not depend on the entire history of the chain. It turns out that a sufficient condition to ensure this is that the adaption is "faded out" gradually over time. See e.g., [AT08] for details.

12.2.4 Initialization

It is necessary to start MCMC in an initial state that has non-zero probability. A natural approach is to first use an optimizer to find a local mode. However, at such points the gradients of the log joint are zero, which can cause problems for some gradient-based MCMC methods, such as HMC (Section 12.5), so it can be better to start "close" to a MAP estimate (see e.g., [HFM17, Sec 7.]).

12.3 Gibbs sampling

The major problems with MH are the need to choose the proposal distribution, and the fact that the acceptance rate may be low. In this section, we describe an MH method that exploits conditional independence properties of a graphical model to automatically create a good proposal, with acceptance probability 1. This method is known as **Gibbs sampling**.[2] (In physics, this method is known as **Glauber dynamics** or the **heat bath** method.) This is the MCMC analog of coordinate descent.[3]

12.3.1 Basic idea

The idea behind Gibbs sampling is to sample each variable in turn, conditioned on the values of all the other variables in the distribution. For example, if we have $D = 3$ variables, we use

- $x_1^{s+1} \sim p(x_1|x_2^s, x_3^s)$
- $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_3^s)$
- $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1})$

This readily generalizes to D variables. (Note that if x_i is a known variable, we do not sample it, but it may be used as input to th another conditional distribution.)

The expression $p(x_i|\boldsymbol{x}_{-i})$ is called the **full conditional** for variable i. In general, x_i may only depend on some of the other variables. If we represent $p(\boldsymbol{x})$ as a graphical model, we can infer the dependencies by looking at i's Markov blanket, which are its neighbors in the graph (see Section 4.2.4.3), so we can write

$$x_i^{s+1} \sim p(x_i|\boldsymbol{x}_{-i}^s) = p(x_i|\boldsymbol{x}_{\text{mb}(i)}^s) \tag{12.19}$$

(Compare to the equation for mean field variational inference in Equation (10.87).)

We can sample some of the nodes in parallel, without affecting correctness. In particular, suppose we can create a **coloring** of the (moralized) undirected graph, such that no two neighboring nodes have the same color. (In general, computing an optimal coloring is NP-complete, but we can use efficient heuristics such as those in [Kub04].) Then we can sample all the nodes of the same color in parallel, and cycle through the colors sequentially [Gon+11].

12.3.2 Gibbs sampling is a special case of MH

It turns out that Gibbs sampling is a special case of MH where we use a sequence of proposals of the form

$$q_i(\boldsymbol{x}'|\boldsymbol{x}) = p(x_i'|\boldsymbol{x}_{-i})\mathbb{I}\left(\boldsymbol{x}_{-i}' = \boldsymbol{x}_{-i}\right) \tag{12.20}$$

That is, we move to a new state where x_i is sampled from its full conditional, but $\boldsymbol{x}_{-i}$ is left unchanged.

2. Josiah Willard Gibbs, 1839–1903, was an American physicist.
3. Several software libraries exist for applying Gibbs sampling to general graphical models, including Nimble, which is a C++ library with an R wrapper, and which replaces older programs such as BUGS and JAGS.

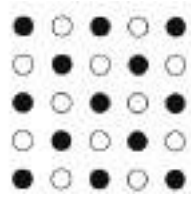


Figure 12.2: Illustration of checkerboard pattern for a 2d MRF. This allows for parallel updates.

We now prove that the acceptance rate of each such proposal is 100%, so the overall algorithm also has an acceptance rate of 100%. We have

$$\alpha = \frac{p(\boldsymbol{x}')q_i(\boldsymbol{x}|\boldsymbol{x}')}{p(\boldsymbol{x})q_i(\boldsymbol{x}'|\boldsymbol{x})} = \frac{p(x_i'|\boldsymbol{x}'_{-i})p(\boldsymbol{x}'_{-i})p(x_i|\boldsymbol{x}'_{-i})}{p(x_i|\boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})p(x_i'|\boldsymbol{x}_{-i})} \tag{12.21}$$

$$= \frac{p(x_i'|\boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})p(x_i|\boldsymbol{x}_{-i})}{p(x_i|\boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})p(x_i'|\boldsymbol{x}_{-i})} = 1 \tag{12.22}$$

where we exploited the fact that $\boldsymbol{x}'_{-i} = \boldsymbol{x}_{-i}$.

The fact that the acceptance rate is 100% does not necessarily mean that Gibbs will converge rapidly, since it only updates one coordinate at a time (see Section 12.3.7). However, if we can group together correlated variables, then we can sample them as a group, which can significantly help mixing.

12.3.3 Example: Gibbs sampling for Ising models

In Section 4.3.2.1, we discuss Ising models and Potts models, which are pairwise MRFs with a 2d grid structure. The joint distribution has the form

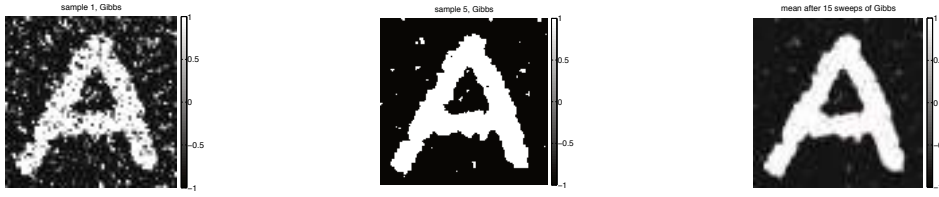
$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_{i\sim j}\psi_{ij}(x_i, x_j|\boldsymbol{\theta}) \tag{12.23}$$

where $i \sim j$ means i and j are neighbors in the graph.

To apply Gibbs sampling to such a model, we just need to iteratively sample from each full conditional:

$$p(x_i|\boldsymbol{x}_{-i}) \propto \prod_{j\in \mathrm{nbr}(i)} \psi_{ij}(x_i, x_j) \tag{12.24}$$

Note that although Gibbs sampling is a sequential algorithm, we can sometimes exploit conditional independence properties to perform parallel updates [RS97a]. In the case of a 2d grid, we can color code nodes using a checkerboard pattern shown in Figure 12.2. This has the property that the black nodes are conditionally independent of each other given the white nodes, and vice versa. Hence we can sample all the black nodes in parallel (as a single group), and then sample all the white nodes, etc.



(a) (b) (c)

Figure 12.3: Example of image denoising using Gibbs sampling. We use an Ising prior with $J = 1$ and a Gaussian noise model with $\sigma = 2$. (a) Sample from the posterior after one sweep over the image. (b) Sample after 5 sweeps. (c) Posterior mean, computed by averaging over 15 sweeps. Compare to Figure 10.9 which shows the results of mean field inference. Generated by ising_image_denoise_demo.ipynb.

To perform the sampling, we need to compute the full conditional in Equation (12.24). In the case of an Ising model with edge potentials $\psi(x_i, x_j) = \exp(Jx_ix_j)$, where $x_i \in \{-1, +1\}$, the full conditional becomes

$$p(x_i = +1|\boldsymbol{x}_{-i}) = \frac{\prod_{j\in\text{nbr}(i)} \psi_{ij}(x_i = +1, x_j)}{\prod_{j\in\text{nbr}(i)} \psi(x_i = +1, x_j) + \prod_{j\in\text{nbr}(i)} \psi(x_i = -1, x_i)} \tag{12.25}$$

$$= \frac{\exp[J \sum_{j\in\text{nbr}(i)} x_j]}{\exp[J \sum_{j\in\text{nbr}(i)} x_j] + \exp[-J \sum_{j\in\text{nbr}(i)} x_j]} \tag{12.26}$$

$$= \frac{\exp[J\eta_i]}{\exp[J\eta_i] + \exp[-J\eta_i]} = \sigma(2J\eta_i) \tag{12.27}$$

where J is the coupling strength, $\eta_i \triangleq \sum_{j\in\text{nbr}(i)} x_j$, and $\sigma(u) = 1/(1 + e^{-u})$ is the sigmoid function. (If we use $x_i \in \{0, 1\}$, this becomes $p(x_i = +1|\boldsymbol{x}_{-i}) = \sigma(J\eta_i)$.) It is easy to see that $\eta_i = x_i(a_i - d_i)$, where a_i is the number of neighbors that agree with (have the same sign as) node i, and d_t is the number of neighbors who disagree. If this number is equal, the "forces" on x_i cancel out, so the full conditional is uniform. Some samples from this model are shown in Figure 4.17.

One application of Ising models is as a prior for binary image denoising problems. In particular, suppose $\boldsymbol{y}$ is a noisy version of $\boldsymbol{x}$, and we wish to compute the posterior $p(\boldsymbol{x}|\boldsymbol{y}) \propto p(\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x})$, where $p(\boldsymbol{x})$ is an Ising prior, and $p(\boldsymbol{y}|\boldsymbol{x}) = \prod_i p(y_i|x_i)$ is a per-site likelihood term. Suppose this is a Gaussian. Let $\psi_i(x_i) = \mathcal{N}(y_i|x_i, \sigma^2)$ be the corresponding "local evidence" term. The full conditional becomes

$$p(x_i = +1|\boldsymbol{x}_{-i}, \boldsymbol{y}) = \frac{\exp[J\eta_i]\psi_i(+1)}{\exp[J\eta_i]\psi_i(+1) + \exp[-J\eta_i]\psi_i(-1)} \tag{12.28}$$

$$= \sigma\left(2J\eta_i - \log\frac{\psi_i(+1)}{\psi_i(-1)}\right) \tag{12.29}$$

Now the probability of x_i entering each state is determined both by compatibility with its neighbors (the Ising prior) and compatibility with the data (the local likelihood term).

See Figure 12.3 for an example of this algorithm applied to a simple image denoising problem. The results are similar to the mean field results in Figure 10.9.

12.3.4 Example: Gibbs sampling for Potts models

We can extend Section 12.3.3 to the Potts models as follows. Recall that the model has the following form:

$$p(\boldsymbol{x}) = \frac{1}{Z}\exp(-\mathcal{E}(\boldsymbol{x})) \tag{12.30}$$

$$\mathcal{E}(\boldsymbol{x}) = -J\sum_{i\sim j}\mathbb{I}(x_i = x_j) \tag{12.31}$$

For a node i with neighbors $\text{nbr}(i)$, the full conditional is thus given by

$$p(x_i = k|\boldsymbol{x}_{-i}) = \frac{\exp(J\sum_{n\in\text{nbr}(i)}\mathbb{I}(x_n = k))}{\sum_{k'}\exp(J\sum_{n\in\text{nbr}(i)}\mathbb{I}(x_n = k'))} \tag{12.32}$$

So if $J > 0$, a node i is more likely to enter a state k if most of its neighbors are already in state k, corresponding to an attractive MRF. If $J < 0$, a node i is more likely to enter a different state from its neighbors, corresponding to a repulsive MRF. See Figure 4.18 for some samples from this model created using this method.

12.3.5 Example: Gibbs sampling for GMMs

In this section, we consider sampling from a Bayesian Gaussian mixture model of the form

$$p(z = k, \boldsymbol{x}|\boldsymbol{\theta}) = \pi_k\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{12.33}$$

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha})\prod_{k=1}^{K}\mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{m}_0, \mathbf{V}_0)\text{IW}(\boldsymbol{\Sigma}_k, \mathbf{S}_0, \nu_0) \tag{12.34}$$

12.3.5.1 Known parameters

Suppose, initially, that the parameteters $\boldsymbol{\theta}$ are known. We can easily draw independent samples from $p(\boldsymbol{x}|\boldsymbol{\theta})$ by using ancestral sampling: first sample z and then $\boldsymbol{x}$. However, for illustrative purposes, we will use Gibbs sampling to draw correlated samples. The full conditional for $p(\boldsymbol{x}|z = k, \boldsymbol{\theta})$ is just $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, and the full conditional for $p(z = k|\boldsymbol{x})$ is given by Bayes' rule:

$$p(z = k|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{\pi_k\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'}\pi_{k'}\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \tag{12.35}$$

An example of this procedure, applied to a mixture of two 1d Gaussians with means at -20 and $+20$, is shown in Figure 12.4. We see that the samples are auto correlated, meaning that if we are in state 1, we will likely stay in that state for a while, and generate values near μ_1; then we will stochastically jump to state 2, and stay near there for a while, etc. (See Section 12.6.3 for a way to measure this.) By contrast, independent samples from the joint would not be correlated at all.

In Section 12.3.5.2, we modify this example to sample the parameters of the GMM from their posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, instead of sampling from $p(\mathcal{D}|\boldsymbol{\theta})$.

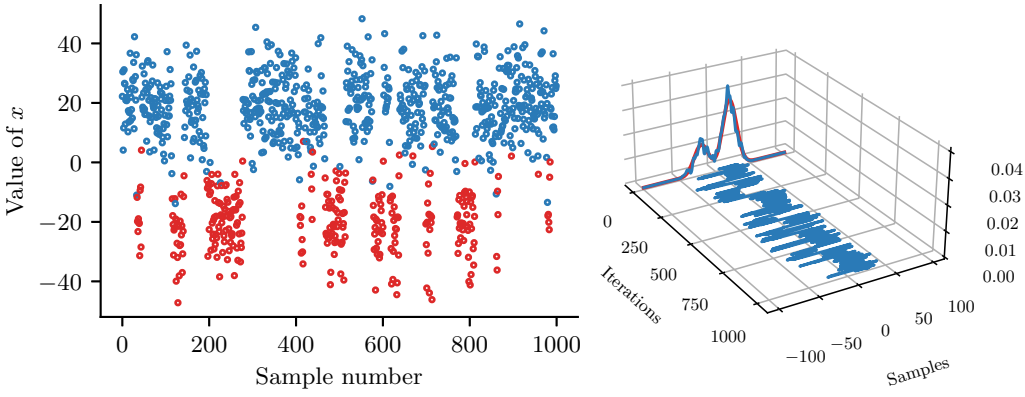


Figure 12.4: (a) Some samples from a mixture of two 1d Gaussians generated using Gibbs sampling. Color denotes the value of z, vertical location denotes the value of x. Horizontal axis represents time (sample number). (b) Traceplot of x over time, and the resulting empirical distribution is shown in blue. The true distribution is shown in red. Compare to Figure 12.1. Generated by mcmc_gmm_demo.ipynb.

12.3.5.2 Unknown parameters

Now suppose the parameters are unknown, so we want to fit the model to data. If we use a conditionally conjugate factored prior, then the full joint distribution is given by

$$p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\boldsymbol{z}|\boldsymbol{\pi}) p(\boldsymbol{\pi}) \prod_{k=1}^{K} p(\boldsymbol{\mu}_k) p(\boldsymbol{\Sigma}_k) \tag{12.36}$$

$$= \left(\prod_{i=1}^{N} \prod_{k=1}^{K} (\pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times \tag{12.37}$$

$$\text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_0, \nu_0) \tag{12.38}$$

We use the same prior for each mixture component.

The full conditionals are as follows. For the discrete indicators, we have

$$p(z_i = k|\boldsymbol{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \propto \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{12.39}$$

For the mixing weights, we have (using results from Section 3.4.2)

$$p(\boldsymbol{\pi}|\boldsymbol{z}) = \text{Dir}(\{\alpha_k + \sum_{i=1}^{N} \mathbb{I}(z_i = k)\}_{k=1}^{K}) \tag{12.40}$$

For the means, we have (using results from Section 3.4.4.1)

$$p(\boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k, \boldsymbol{z}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{m}_k, \mathbf{V}_k) \tag{12.41}$$

$$\mathbf{V}_k^{-1} = \mathbf{V}_0^{-1} + N_k\boldsymbol{\Sigma}_k^{-1} \tag{12.42}$$

$$\boldsymbol{m}_k = \mathbf{V}_k(\boldsymbol{\Sigma}_k^{-1}N_k\overline{\boldsymbol{x}}_k + \mathbf{V}_0^{-1}\boldsymbol{m}_0) \tag{12.43}$$

$$N_k \triangleq \sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right) \tag{12.44}$$

$$\overline{\boldsymbol{x}}_k \triangleq \frac{\sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right)\boldsymbol{x}_i}{N_k} \tag{12.45}$$

For the covariances, we have (using results from Section 3.4.4.2)

$$p(\boldsymbol{\Sigma}_k|\boldsymbol{\mu}_k, \boldsymbol{z}, \boldsymbol{x}) = \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_k, \nu_k) \tag{12.46}$$

$$\mathbf{S}_k = \mathbf{S}_0 + \sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^{\mathsf{T}} \tag{12.47}$$

$$\nu_k = \nu_0 + N_k \tag{12.48}$$

12.3.6 Metropolis within Gibbs

When implementing Gibbs sampling, we have to sample from the full conditionals. If the distributions are conjugate, we can compute the full conditional in closed form, but in the general case, we will need to devise special algorithms to sample from the full conditionals.

One approach is to use the MH algorithm; this is called **Metropolis within Gibbs**. In particular, to sample from $x_i^{s+1} \sim p(x_i|\boldsymbol{x}_{1:i-1}^{s+1}, \boldsymbol{x}_{i+1:D}^{s})$, we proceed in 3 steps:

1. Propose $x_i' \sim q(x_i'|x_i^s)$
2. Compute the acceptance probability $A_i = \min(1, \alpha_i)$ where

$$\alpha_i = \frac{p(\boldsymbol{x}_{1:i-1}^{s+1}, x_i', \boldsymbol{x}_{i+1:D}^{s})/q(x_i'|x_i^s)}{p(\boldsymbol{x}_{1:i-1}^{s+1}, x_i^s, \boldsymbol{x}_{i+1:D}^{s})/q(x_i^s|x_i')} \tag{12.49}$$

3. Sample $u \sim U(0, 1)$ and set $x_i^{s+1} = x_i'$ if $u < A_i$, and set $x_i^{s+1} = x_i^s$ otherwise.

12.3.7 Blocked Gibbs sampling

Gibbs sampling can be quite slow, since it only updates one variable at a time (so-called **single site updating**). If the variables are highly correlated, the chain will move slowly through the state space. This is illustrated in Figure 12.5, where we illustrate sampling from a 2d Gaussian. The ellipse represents the covariance matrix. The size of the moves taken by Gibbs sampling is controlled by the variance of the conditional distributions. If the variance is ℓ along some coordinate direction, but the support of the distribution is L along this dimension, then we need $O((L/\ell)^2)$ steps to obtain an independent sample.

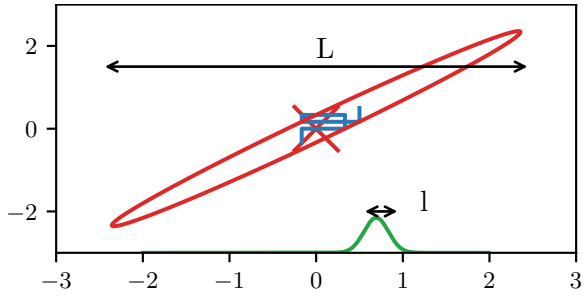


Figure 12.5: Illustration of potentially slow sampling when using Gibbs sampling for a skewed 2d Gaussian. Adapted from Figure 11.11 of [Bis06]. Generated by gibbs_gauss_demo.ipynb.

In some cases we can efficiently sample groups of variables at a time. This is called **blocked Gibbs sampling** [JKK95; WY02], and can make much bigger moves through the state space.

As an example, suppose we want to perform Bayesian inference for a state-space model, such as an HMM, i.e., we want to sample from

$$p(\boldsymbol{\theta}, \boldsymbol{z}|\boldsymbol{x}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{z}_t, \boldsymbol{\theta}) p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{\theta}) \tag{12.50}$$

We can use blocked Gibbs sampling, where we alternate between sampling from $p(\boldsymbol{\theta}|\boldsymbol{z}, \boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta})$. The former is easy to do (assuming conjugate priors), since all variables in the model are observed (see Section 29.8.4.1). The latter can be done using forwards-filtering backwards-sampling (Section 9.2.7).

12.3.8 Collapsed Gibbs sampling

We can sometimes gain even greater speedups by analytically integrating out some of the unknown quantities. This is called a **collapsed Gibbs sampler**, and it tends to be more efficient, since it is sampling in a lower dimensional space. This can result in lower variance, as discussed in Section 11.6.2.

As an example, consider a GMM with a fully conjugate prior. This can be represented as a DPGM as shown in Figure 12.6a. Since the prior is conjugate, we can analytically integrate out the model parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and $\boldsymbol{\pi}$, so the only remaining hidden variables are the discrete indicator variables $\boldsymbol{z}$. However, once we integrate out $\boldsymbol{\pi}$, all the z_i nodes become inter-dependent. Similarly, once we integrate out $\boldsymbol{\theta}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, all the $\boldsymbol{x}_i$ nodes become inter-dependent, as shown in Figure 12.6b. Nevertheless, we can easily compute the full conditionals, and hence implement a Gibbs sampler, as we explain below. In particular, the full conditional for the latent indicators is given by

$$\begin{aligned} p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}, \cancel{\boldsymbol{\beta}}) p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \cancel{\boldsymbol{\alpha}}, \boldsymbol{\beta}) && (12.51) \\ &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) \\ & p(\boldsymbol{x}_{-i}|\cancel{\boldsymbol{\xi}_i} \boldsymbol{\xi} = k, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) && (12.52) \\ &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) && (12.53) \end{aligned}$$

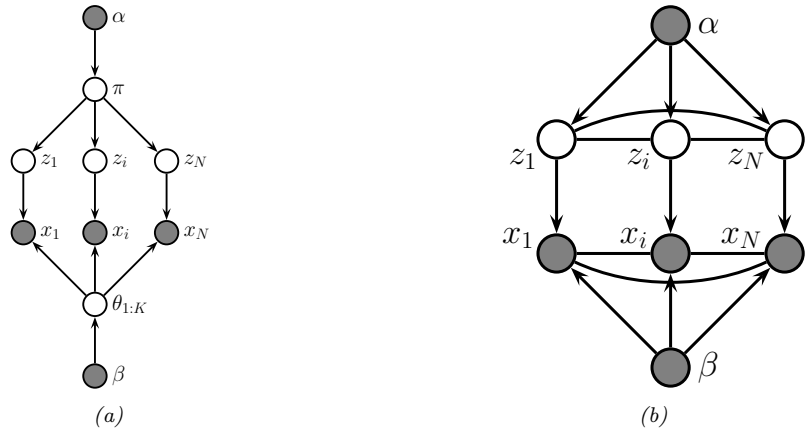


Figure 12.6: (a) A mixture model represented as an "unrolled" DPGM. (b) After integrating out the continuous latent parameters.

where $\boldsymbol{\beta} = (\boldsymbol{m}_0, \mathbf{V}_0, \mathbf{S}_0, \nu_0)$ are the hyper-parameters for the class-conditional densities. We now discuss how to compute these terms.

Suppose we use a symmetric prior of the form $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$, where $\alpha_k = \alpha/K$, for the mixing weights. Then we can obtain the first term in Equation (12.53), from Equation (3.96), where

$$p(z_1, \ldots, z_N|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)} \tag{12.54}$$

Hence

$$p(z_i = k|\boldsymbol{z}_{-i}, \alpha) = \frac{p(\boldsymbol{z}_{1:N}|\alpha)}{p(\boldsymbol{z}_{-i}|\alpha)} = \frac{\frac{1}{\Gamma(N+\alpha)}}{\frac{1}{\Gamma(N+\alpha-1)}} \times \frac{\Gamma(N_k + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} \tag{12.55}$$

$$= \frac{\Gamma(N+\alpha-1)}{\Gamma(N+\alpha)} \frac{\Gamma(N_{k,-i} + 1 + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} = \frac{N_{k,-i} + \alpha}{N + \alpha - 1} \tag{12.56}$$

where $N_{k,-i} \triangleq \sum_{n \neq i} \mathbb{I}(z_n = k) = N_k - 1$, and where we exploited the fact that $\Gamma(x+1) = x\Gamma(x)$.

To obtain the second term in Equation (12.53), which is the posterior predictive distribution for $\boldsymbol{x}_i$ given all the other data and all the assignments, we use the fact that

$$p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, \boldsymbol{z}_{-i}, z_i = k, \boldsymbol{\beta}) = p(\boldsymbol{x}_i|\mathcal{D}_{-i,k}, \boldsymbol{\beta}) \tag{12.57}$$

where $\mathcal{D}_{-i,k} = \{\boldsymbol{x}_j : z_j = k, j \neq i\}$ is all the data assigned to cluster k except for $\boldsymbol{x}_i$. If we use a conjugate prior for $\boldsymbol{\theta}_k$, we can compute $p(\boldsymbol{x}_i|\mathcal{D}_{-i,k}, \boldsymbol{\beta})$ in closed form. Furthermore, we can efficiently update these predictive likelihoods by caching the sufficient statistics for each cluster. To compute the above expression, we remove $\boldsymbol{x}_i$'s statistics from its current cluster (namely z_i), and then evaluate $\boldsymbol{x}_i$ under each cluster's posterior predictive distribution. Once we have picked a new cluster, we add $\boldsymbol{x}_i$'s statistics to this new cluster.

Some pseudo-code for one step of the algorithm is shown in Algorithm 12.2, based on [Sud06, p94]. (We update the nodes in random order to improve the mixing time, as suggested in [RS97b].) We can initialize the sample by sequentially sampling from $p(z_i|\boldsymbol{z}_{1:i-1}, \boldsymbol{x}_{1:i})$. In the case of GMMs, both the naive sampler and collapsed sampler take $O(NKD)$ time per step.

Algorithm 12.2: Collapsed Gibbs sampler for a mixture model

for *each* $i = 1 : N$ *in random order* **do**
  Remove $\boldsymbol{x}_i$'s sufficient statistics from old cluster z_i
  for *each* $k = 1 : K$ **do**
    Compute $p_k(\boldsymbol{x}_i|\boldsymbol{\beta}) = p(\boldsymbol{x}_i|\{\boldsymbol{x}_j : z_j = k, j \neq i\}, \boldsymbol{\beta})$
  Compute $p(z_i = k|\boldsymbol{z}_{-i}, \alpha) \propto (N_{k,-i} + \alpha/K)p_k(\boldsymbol{x}_i)$
  Sample $z_i \sim p(z_i|\cdot)$
  Add $\boldsymbol{x}_i$'s sufficient statistics to new cluster z_i

The primary advantage of using the collapsed sampler is that it extends to the case where we have an "infinite" number of mixture components, as in the Dirichlet process mixture model of Supplementary Section 31.2.2.

12.4 Auxiliary variable MCMC

Sometimes we can dramatically improve the efficiency of sampling by introducing **auxiliary variables**, in order to reduce correlation between the original variables. If the original variables are denoted by $\boldsymbol{x}$, and the auxiliary variables by $\boldsymbol{v}$, then the augmented distribution becomes $p(\boldsymbol{x}, \boldsymbol{v})$. We assume it is easier to sample from this than the marginal distribution $p(\boldsymbol{x})$. If so, we can draw joint samples $(\boldsymbol{x}^s, \boldsymbol{v}^s) \sim p(\boldsymbol{x}, \boldsymbol{v})$, and then just "throw away" the $\boldsymbol{v}^s$, and the result will be samples from the desired marginal, $\boldsymbol{x}^s \sim \sum_{\boldsymbol{v}} p(\boldsymbol{x}, \boldsymbol{v})$. We give some examples of this below.

12.4.1 Slice sampling

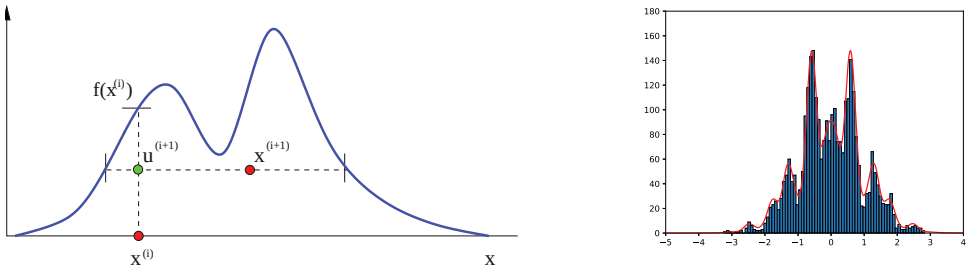
Consider sampling from a univariate, but multimodal, distribution $p(x) = \tilde{p}(x)/Z_p$, where $\tilde{p}(x)$ is unnormalized, and $Z_p = \int \tilde{p}(x)dx$. We can sometimes improve the ability to make large moves by adding a uniform auxiliary variable v. We define the joint distribution as follows:

$$\hat{p}(x, v) = \begin{cases} 1/Z_p & \text{if } 0 \leq v \leq \tilde{p}(x) \\ 0 & \text{otherwise} \end{cases} \tag{12.58}$$

The marginal distribution over x is given by

$$\int \hat{p}(x, v)dv = \int_0^{\tilde{p}(x)} \frac{1}{Z_p} dv = \frac{\tilde{p}(x)}{Z_p} = p(x) \tag{12.59}$$

so we can sample from $p(x)$ by sampling from $\hat{p}(x, v)$ and then ignoring v. To do this, we will use a technique called **slice sampling** [Nea03].



(a) (b)

Figure 12.7: Slice sampling. (a) Illustration of one step of the algorithm in 1d. Given a previous sample x^i, we sample u^{i+1} uniformly on $[0, f(x^i)]$, where $f = \tilde{p}$ is the (unnormalized) target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. From Figure 15 of [And+03]. Used with kind permission of Nando de Freitas. (b) Output of slice sampling applied to a 1d distribution. Generated by slice_ sampling_ demo_ 1d.ipynb.

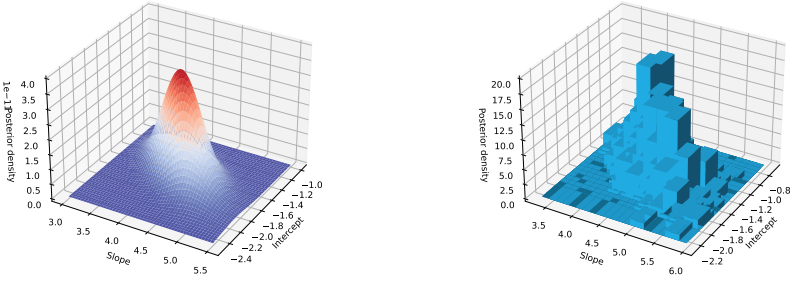


Figure 12.8: Posterior for binomial regression for 1d data. Left: slice sampling approximation. Right: grid approximation. Generated by slice_ sampling_ demo_ 2d.ipynb.

This works as follows. Given previous sample x^i, we sample v^{i+1} from

$$p(v|x^i) = U_{[0,\tilde{p}(x^i)]}(v) \tag{12.60}$$

This amounts to uniformly picking a point on the vertical line between 0 and $\tilde{p}(x^i)$, We use this to construct a "slice" of the density at or above this height, by computing $A^{i+1} = \{x : \tilde{p}(x) \geq v^{i+1}\}$. We then sample x^{i+1} uniformly from ths set. See Figure 12.7(a) for an illustration.

To compute the level set A, we can use an iterative search procedure called **stepping out**, in which we start with an interval $x_{min} \leq x \leq x_{max}$ around the current point x^i of some width, and then we keep extending it until the endpoints fall outside the slice. We can then use rejection sampling to sample from the interval. For the details, see [Nea03].

To apply the method to multivariate distributions, we sample one extra auxiliary variable for each dimension. Thus we perfom $2D$ sampling operations to draw a single joint sample, where

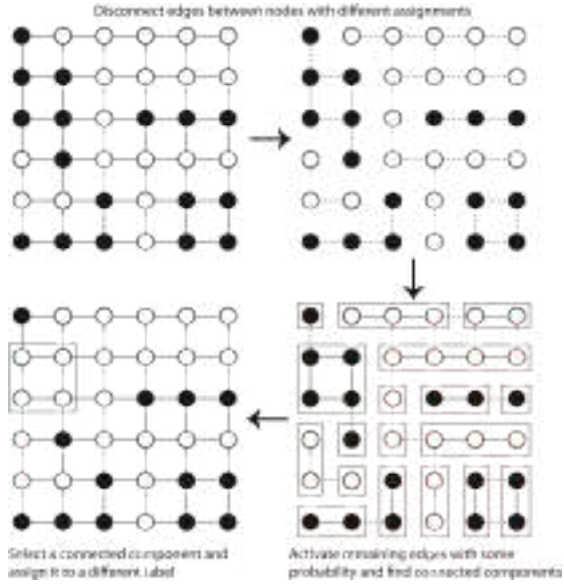


Figure 12.9: Illustration of the Swendsen-Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

D is the number of random variables. The advantage of this over Gibbs sampling applied to the original (non-augmented) distribution is that it only needs access to the unnormalized joint, not the full-conditionals.

Figure 12.7(b) illustrates the algorithm in action on a synthetic 1d problem. Figure 12.8 illustrates its behavior on a slightly harder problem, namely binomial logistic regression. The model has the form $y_i \sim \text{Bin}(n_i, \text{logit}(\beta_1 + \beta_2 x_i))$. We use a vague Gaussian prior for the β_j's. On the left we show the slice sampling approximation to the posterior, and on the right we shpw a grid-based approximation, as a simple deteterministic proxy for the true posterior. We see a close correpondence.

12.4.2 Swendsen-Wang

Consider an Ising model of the following form: $p(\boldsymbol{x}) = \frac{1}{Z}\prod_e \Psi(\boldsymbol{x}_e)$, where $\boldsymbol{x}_e = (x_i, x_j)$ for edge $e = (i, j)$, $x_i \in \{+1, -1\}$, and the edge potential is defined by $\begin{pmatrix} e^J & e^{-J} \\ e^{-J} & e^J \end{pmatrix}$, where J is the edge strength. In Section 12.3.3, we discussed how to apply Gibbs sampling to this model. However, this can be slow when J is large in absolute value, because neighboring states can be highly correlated. The **Swendsen-Wang** algorithm [SW87b] is an auxiliary variable MCMC sampler which mixes much faster, at least for the case of attractive or ferromagnetic models, with $J > 0$.

Suppose we introduce auxiliary binary variables, one per edge.[4] These are called **bond variables**, and will be denoted by $\boldsymbol{v}$. We then define an extended model $p(\boldsymbol{x}, \boldsymbol{v})$ of the form $p(\boldsymbol{x}, \boldsymbol{v}) =$

4. Our presentation of the method is based on notes by David MacKay, available from http://www.inference.phy.cam.ac.uk/mackay/itila/swendsen.pdf.

$\frac{1}{Z'}\prod_e \Psi(\boldsymbol{x}_e, v_e)$, where $v_e \in \{0,1\}$, and we define the new edge potentials as follows:

$$\Psi(\boldsymbol{x}_e, v_e=0) = \begin{pmatrix} e^{-J} & e^{-J} \\ e^{-J} & e^{-J} \end{pmatrix}, \; \Psi(\boldsymbol{x}_e, v_e=1) = \begin{pmatrix} e^{J}-e^{-J} & 0 \\ 0 & e^{J}-e^{-J} \end{pmatrix} \tag{12.61}$$

It is clear that $\sum_{v_e=0}^{1} \Psi(\boldsymbol{x}_e, v_e) = \Psi(\boldsymbol{x}_e)$, and hence that $\sum_{\boldsymbol{v}} p(\boldsymbol{x}, \boldsymbol{v}) = p(\boldsymbol{x})$, as required.

Fortunately, it is easy to apply Gibbs sampling to this extended model. The full conditional $p(\boldsymbol{v}|\boldsymbol{x})$ factorizes over the edges, since the bond variables are conditionally independent given the node variables. Furthermore, the full conditional $p(v_e|\boldsymbol{x}_e)$ is simple to compute: if the nodes on either end of the edge are in the same state ($x_i = x_j$), we set the bond v_e to 1 with probability $p = 1 - e^{-2J}$, otherwise we set it to 0. In Figure 12.9 (top right), the bonds that could be turned on (because their corresponding nodes are in the same state) are represented by dotted edges. In Figure 12.9 (bottom right), the bonds that are randomly turned on are represented by solid edges.

To sample $p(\boldsymbol{x}|\boldsymbol{v})$, we proceed as follows. Find the connected components defined by the graph induced by the bonds that are turned on. (Note that a connected component may consist of a singleton node.) Pick one of these components uniformly at random. All the nodes in each such component must have the same state. Pick a state ± 1 uniformly at random, and set all the variables in this component to adopt this new state. This is illustrated in Figure 12.9 (bottom right), where the green square denotes the selected connected component; we set all the nodes within this square to white, to get the bottom left configuration.

It should be intuitively clear that Swendsen-Wang makes much larger moves through the state space than Gibbs sampling. The gains are exponentially large for certain settings of the edge parameter. More precisely, let the edge strength be parameterized by J/T, where $T > 0$ is a computational temperature. For large T, the nodes are roughly independent, so both methods work equally well. However, as T approaches a **critical temperature** T_c, the typical states of the system have very long correlation lengths, and Gibbs sampling takes a very long time to generate independent samples. As the temperature continues to drop, the typical states are either all on or all off. The frequency with which Gibbs sampling moves between these two modes is exponentially small. By contrast, SW mixes rapidly at all temperatures.

Unfortunately, if any of the edge weights are negative, $J < 0$, the system is **frustrated**, and there are exponentially many modes, even at low temperature. SW does not work very well in this setting, since it tries to force many neighboring variables to have the same state. In fact, sampling from these kinds of frustrated systems is provably computationally hard for any algorithm [JS93; JS96].

12.5 Hamiltonian Monte Carlo (HMC)

Many MCMC algorithms perform poorly in high dimensional spaces, because they rely on a form of random search based on local perturbations. In this section, we discuss a method known as **Hamiltonian Monte Carlo** or **HMC**, that leverages gradient information to guide the local moves. This is an auxiliary variable method (Section 12.4) derived from physics [Dua+87; Nea93; Mac03; Nea10; Bet17].[5] In particular, the method builds on **Hamiltonian mechanics**, which we describe below.

5. The method was originally called **hybrid MC** [Dua+87]. It was introduced to the statistics community in [Nea93], and was renamed to Hamiltonian MC in [Mac03].

12.5.1 Hamiltonian mechanics

Consider a particle rolling around an energy landscape. We can characterize the motion of the particle in terms of its position $\boldsymbol{\theta} \in \mathbb{R}^D$ (often denoted by $\boldsymbol{q}$) and its momentum $\boldsymbol{v} \in \mathbb{R}^D$ (often denoted by $\boldsymbol{p}$). The set of possible values for $(\boldsymbol{\theta}, \boldsymbol{v})$ is called the **phase space**. We define the **Hamiltonian** function for each point in phase space as follows:

$$\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v}) \triangleq \mathcal{E}(\boldsymbol{\theta}) + \mathcal{K}(\boldsymbol{v}) \tag{12.62}$$

where $\mathcal{E}(\boldsymbol{\theta})$ is the **potential energy**, $\mathcal{K}(\boldsymbol{v})$ is the **kinetic energy**, and the Hamiltonian is the total energy. In a physical setting, the potential energy is due to the pull of gravity, and the momentum is due to the motion of the particle. In a statistical setting, we often take the potential energy to be

$$\mathcal{E}(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta}) \tag{12.63}$$

where $\tilde{p}(\boldsymbol{\theta})$ is a possibly unnormalized distribution, such as $p(\boldsymbol{\theta}, \mathcal{D})$, and the kinetic energy to be

$$\mathcal{K}(\boldsymbol{v}) = \frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{\Sigma}^{-1}\boldsymbol{v} \tag{12.64}$$

where $\boldsymbol{\Sigma}$ is a positive definite matrix, known as the **inverse mass matrix**.

Stable orbits are defined by trajectories in phase space that have a constant energy, The trajectory of a particle within an energy level set can be obtained by solving the following continuous time differential equations, known as **Hamilton's equations**:

$$\begin{aligned} \frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial \mathcal{H}}{\partial \boldsymbol{v}} = \frac{\partial \mathcal{K}}{\partial \boldsymbol{v}} \\ \frac{d\boldsymbol{v}}{dt} &= -\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}} = -\frac{\partial \mathcal{E}}{\partial \boldsymbol{\theta}} \end{aligned} \tag{12.65}$$

To see why energy is conserved, note that

$$\frac{d\mathcal{H}}{dt} = \sum_{i=1}^{D}\left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i}\frac{d\boldsymbol{\theta}_i}{dt} + \frac{\partial \mathcal{H}}{\partial \boldsymbol{v}_i}\frac{d\boldsymbol{v}_i}{dt}\right] = \sum_{i=1}^{D}\left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i}\frac{\partial \mathcal{H}}{\partial \boldsymbol{v}_i} - \frac{\partial \mathcal{H}}{\partial \boldsymbol{\theta}_i}\frac{\partial \mathcal{H}}{\partial \boldsymbol{v}_i}\right] = 0 \tag{12.66}$$

Intuitively, we can understand this result as follows: a satellite in orbit around a planet will "want" to continue in a straight line due to its momentum, but will get pulled in towards the planet due to gravity, and if these forces cancel, the orbit is stable. If the satellite starts spiraling towards the planet, its kinetic energy will increase but its potential energy will decrease.

Note that the mapping from $(\boldsymbol{\theta}(t), \boldsymbol{v}(t))$ to $(\boldsymbol{\theta}(t+s), \boldsymbol{v}(t+s))$ for some time increment s is invertible for small enough time steps. Furthermore, this mapping is volume preserving, so has a Jacobian determinant of 1. (See e.g., [BZ20, p287] for a proof.) These facts will be important later when we turn this system into an MCMC algorithm.

12.5.2 Integrating Hamilton's equations

In this section, we discuss how to simulate Hamilton's equations in discrete time.

12.5.2.1 Euler's method

The simplest way to model the time evolution is to update the position and momentum simultaneously by a small amount, known as the step size η:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \eta \frac{d\boldsymbol{v}}{dt}(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = \boldsymbol{v}(t) - \eta \frac{\partial \mathcal{E}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \tag{12.67}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \frac{d\boldsymbol{\theta}}{dt}(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = \boldsymbol{\theta}_t + \eta \frac{\partial \mathcal{K}(\boldsymbol{v}_t)}{\partial \boldsymbol{v}} \tag{12.68}$$

If the kinetic energy has the form in Equation (12.64) then the second expression simplifies to

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \boldsymbol{\Sigma}^{-1} \boldsymbol{v}_{t+1} \tag{12.69}$$

This is known as **Euler's method**.

12.5.2.2 Modified Euler's method

The **modified Euler's method** is slightly more accurate, and works as follows: First update the momentum, and then update the position using the new momentum:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \eta \frac{d\boldsymbol{v}}{dt}(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = \boldsymbol{v}_t - \eta \frac{\partial \mathcal{E}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \tag{12.70}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \frac{d\boldsymbol{\theta}}{dt}(\boldsymbol{\theta}_t, \boldsymbol{v}_{t+1}) = \boldsymbol{\theta}_t + \eta \frac{\partial \mathcal{K}(\boldsymbol{v}_{t+1})}{\partial \boldsymbol{v}} \tag{12.71}$$

Unfortunately, the asymmetry of this method can cause some theoretical problems (see e.g., [BZ20, p287]) which we resolve below.

12.5.2.3 Leapfrog integrator

In this section, we discuss the **leapfrog integrator**, which is a symmetrized version of the modified Euler method. We first perform a "half" update of the momentum, then a full update of the position, and then finally another "half" update of the momentum:

$$\boldsymbol{v}_{t+1/2} = \boldsymbol{v}_t - \frac{\eta}{2} \frac{\partial \mathcal{E}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \tag{12.72}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \frac{\partial \mathcal{K}(\boldsymbol{v}_{t+1/2})}{\partial \boldsymbol{v}} \tag{12.73}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_{t+1/2} - \frac{\eta}{2} \frac{\partial \mathcal{E}(\boldsymbol{\theta}_{t+1})}{\partial \boldsymbol{\theta}} \tag{12.74}$$

If we perform multiple leapfrog steps, it is equivalent to performing a half step update of $\boldsymbol{v}$ at the beginning and end of the trajectory, and alternating between full step updates of $\boldsymbol{\theta}$ and $\boldsymbol{v}$ in between.

12.5.2.4 Higher order integrators

It is possible to define higher order integrators that are more accurate, but take more steps. For details, see [BRSS18].

12.5.3 The HMC algorithm

We now describe how to use Hamiltonian dynamics to define an MCMC sampler in the expanded state space $(\boldsymbol{\theta}, \boldsymbol{v})$. The target distribution has the form

$$p(\boldsymbol{\theta}, \boldsymbol{v}) = \frac{1}{Z} \exp\left[-\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v})\right] = \frac{1}{Z} \exp\left[-\mathcal{E}(\boldsymbol{\theta}) - \frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{v}\right] \tag{12.75}$$

The marginal distribution over the latent variables of interest has the form

$$p(\boldsymbol{\theta}) = \int p(\boldsymbol{\theta}, \boldsymbol{v}) d\boldsymbol{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \int \frac{1}{Z_p} e^{-\frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{v}} d\boldsymbol{v} = \frac{1}{Z_q} e^{-\mathcal{E}(\boldsymbol{\theta})} \tag{12.76}$$

Suppose the previous state of the Markov chain is $(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})$. To sample the next state, we proceed as follows. We set the initial position to $\boldsymbol{\theta}'_0 = \boldsymbol{\theta}_{t-1}$, and sample a new random momentum, $\boldsymbol{v}'_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. We then initialize a random trajectory in the phase space, starting at $(\boldsymbol{\theta}'_0, \boldsymbol{v}'_0)$, and followed for L leapfrog steps, until we get to the final proposed state $(\boldsymbol{\theta}^*, \boldsymbol{v}^*) = (\boldsymbol{\theta}'_L, \boldsymbol{v}'_L)$. If we have simulated Hamiltonian mechanics correctly, the energy should be the same at the start and end of this process; if not, we say the HMC has **diverged**, and we reject the sample. If the energy is constant, we compute the MH acceptance probability

$$\alpha = \min\left(1, \frac{p(\boldsymbol{\theta}^*, \boldsymbol{v}^*)}{p(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})}\right) = \min\left(1, \exp\left[-\mathcal{H}(\boldsymbol{\theta}^*, \boldsymbol{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})\right]\right) \tag{12.77}$$

(The transition probabilities cancel since the proposal is reversible.) Finally, we accept the proposal by setting $(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = (\boldsymbol{\theta}^*, \boldsymbol{v}^*)$ with probability α, otherwise we set $(\boldsymbol{\theta}_t, \boldsymbol{v}_t) = (\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})$. (In practice we don't need to keep the momentum term, it is only used inside of the leapfrog algorithm.) See Algorithm 12.3 for the pseudocode.[6]

Algorithm 12.3: Hamiltonian Monte Carlo

for $t = 1 : T$ **do**
  Generate random momentum $\boldsymbol{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
  Set $(\boldsymbol{\theta}'_0, \boldsymbol{v}'_0) = (\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})$
  Half step for momentum: $\boldsymbol{v}'_{\frac{1}{2}} = \boldsymbol{v}'_0 - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}'_0)$
  for $l = 1 : L - 1$ **do**
    $\boldsymbol{\theta}'_l = \boldsymbol{\theta}'_{l-1} + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}'_{l-1/2}$
    $\boldsymbol{v}'_{l+1/2} = \boldsymbol{v}'_{l-1/2} - \eta\nabla\mathcal{E}(\boldsymbol{\theta}'_l)$
  Full step for location: $\boldsymbol{\theta}'_L = \boldsymbol{\theta}'_{L-1} + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}'_{L-1/2}$
  Half step for momentum: $\boldsymbol{v}'_L = \boldsymbol{v}'_{L-1/2} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}'_L)$
  Compute proposal $(\boldsymbol{\theta}^*, \boldsymbol{v}^*) = (\boldsymbol{\theta}'_L, \boldsymbol{v}'_L)$
  Compute $\alpha = \min\left(1, \exp[-\mathcal{H}(\boldsymbol{\theta}^*, \boldsymbol{v}^*) + \mathcal{H}(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})]\right)$
  Set $\boldsymbol{\theta}_t = \boldsymbol{\theta}^*$ with probability α, otherwise $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$.

6. There are many high-quality implementations of HMC. For example, BlackJAX in JAX.

We need to sample a new momentum at each iteration to satisfy ergodicity. To see why, recall that $\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v})$ stays approximately constant as we move through phase space. If $\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v}) = \mathcal{E}(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{v}$, then clearly $\mathcal{E}(\boldsymbol{\theta}) \leq \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v}) = h$ for all locations $\boldsymbol{\theta}$ along the trajectory. Thus the sampler cannot reach states where $\mathcal{E}(\boldsymbol{\theta}) > h$. To ensure the sampler explores the full space, we must pick a random momentum at the start of each iteration.

12.5.4 Tuning HMC

We need to specify three hyperparameters for HMC: the number of leapfrog steps L, the step size η, and the covariance $\boldsymbol{\Sigma}$.

12.5.4.1 Choosing the number of steps using NUTS

We want to choose the number of leapfrog steps L to be large enough that the algorithm explores the level set of constant energy, but without doubling back on itself, which would waste computation, due to correlated samples. Fortunately, there is an algorithm, known as the **no-U-turn sampler** or **NUTS** algorithm [HG14], which can adaptively choose L for us.

12.5.4.2 Choosing the step size

When $\boldsymbol{\Sigma} = \mathbf{I}$, the ideal step size η should be roughly equal to the width of $\mathcal{E}(\boldsymbol{\theta})$ in the most constrained direction of the local energy landscape. For a locally quadratic potential, this corresponds to the square root of the smallest marginal standard deviation of the local covariance matrix. (If we think of the energy surface as a valley, this corresponds to the direction with the steepest sides.) A step size much larger than this will cause moves that are likely to be rejected because they move to places which increase the potential energy too much. On the other hand, if the step size is too low, the proposal distribution will not move much from the starting position, and the algorithm will be very slow.

In [BZ20, Sec 9.5.4] they recommend the following heuristic for picking η: set $\boldsymbol{\Sigma} = \mathbf{I}$ and $L = 1$, and then vary η until the acceptance rates are in the range of 40%–80%. Of course, different step sizes might be needed in different parts of the state space. In this case, we can use learning rate schedules from the optimization literature, such as cyclical schedules [Zha+20d].

12.5.4.3 Choosing the covariance (inverse mass) matrix

To allow for larger step sizes, we can use a smarter choice for $\boldsymbol{\Sigma}$, also called the **inverse mass matrix**. One way to estimate a fixed $\boldsymbol{\Sigma}$ is to run HMC with $\boldsymbol{\Sigma} = \mathbf{I}$ for a **warm-up** period, until the chain is "burned in" (see Section 12.6); then we run for a few more steps, so we can compute the empirical covariance matrix using $\boldsymbol{\Sigma} = \mathbb{E}\left[(\boldsymbol{\theta} - \overline{\boldsymbol{\theta}})(\boldsymbol{\theta} - \overline{\boldsymbol{\theta}})^\mathsf{T}\right]$. In [Hof+19] they propose a method called the **NeuTra HMC** algorithm which "neutralizes" bad geometry by learning an inverse autoregressive flow model (Section 23.2.4.3) in order to map the warped distribution to an isotropic Gaussian. This is often an order of magnitude faster than vanilla HMC.

12.5.5 Riemann manifold HMC

If we let the covariance matrix change as we move position, so $\boldsymbol{\Sigma}$ is a function of $\boldsymbol{\theta}$, the method is known as **Riemann manifold HMC** or **RM-HMC** [GC11; Bet13], since the moves follow a curved manifold, rather than the flat manifold induced by a constant $\boldsymbol{\Sigma}$.

A natural choice for the covariance matrix is to use the Hessian at the current location, to capture the local geometry:

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \nabla^2 \mathcal{E}(\boldsymbol{\theta}) \tag{12.78}$$

Since this is not always positive definite, an alternative, that can be used for some problems, is to use the Fisher information matrix (Section 3.3.4), given by

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}\left[\nabla^2 \log p(\boldsymbol{x}|\boldsymbol{\theta})\right] \tag{12.79}$$

Once we have computed $\boldsymbol{\Sigma}(\boldsymbol{\theta})$, we can compute the kinetic energy as follows:

$$\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{v}) = \frac{1}{2}\log((2\pi)^D |\boldsymbol{\Sigma}(\boldsymbol{\theta})|) + \frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{\Sigma}(\boldsymbol{\theta})\boldsymbol{v} \tag{12.80}$$

Unfortunately the Hamiltonian updates of $\boldsymbol{\theta}$ and $\boldsymbol{v}$ are no longer separable, which makes the RM-HMC algorithm more complex to implement, so it is not widely used.

12.5.6 Langevin Monte Carlo (MALA)

A special case of HMC occurs when we take $L = 1$ leapfrog steps. This is known as **Langevin Monte Carlo** (**LMC**), or the **Metropolis adjusted Langevin algorithm** (**MALA**) [RT96]. This gives rise to the simplified algorithm shown in Algorithm 12.4.

Algorithm 12.4: Langevin Monte Carlo

for $t = 1 : T$ **do**
- Generate random momentum $\boldsymbol{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
- $\boldsymbol{\theta}^* = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2}\boldsymbol{\Sigma}^{-1}\nabla\mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}_{t-1}$
- $\boldsymbol{v}^* = \boldsymbol{v}_{t-1} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_{t-1}) - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}^*)$
- Compute $\alpha = \min\left(1, \exp[-\mathcal{H}(\boldsymbol{\theta}^*, \boldsymbol{v}^*)] / \exp[-\mathcal{H}(\boldsymbol{\theta}_{t-1}, \boldsymbol{v}_{t-1})]\right)$
- Set $\boldsymbol{\theta}_t = \boldsymbol{\theta}^*$ with probability α, otherwise $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$.

A further simplification is to eliminate the MH acceptance step. In this case, the update becomes

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2}\boldsymbol{\Sigma}^{-1}\nabla\mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}_{t-1} \tag{12.81}$$

$$= \boldsymbol{\theta}_{t-1} - \frac{\eta^2}{2}\boldsymbol{\Sigma}^{-1}\nabla\mathcal{E}(\boldsymbol{\theta}_{t-1}) + \eta\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}_{t-1} \tag{12.82}$$

where $\boldsymbol{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ and $\boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This is just like gradient descent with added noise. If we set $\boldsymbol{\Sigma}$ to be the Fisher information matrix, this becomes natural gradient descent (Section 6.4) with

added noise. If we approximate the gradient with a stochastic gradient, we get a method known as **SGLD**, or **stochastic gradient Langevin descent** (see Section 12.7.1 for details).

Now suppose $\boldsymbol{\Sigma} = \mathbf{I}$, and we set $\eta = \sqrt{2}$. In continuous time, we get the following stochastic differential equation (SDE), known as **Langevin diffusion**:

$$d\boldsymbol{\theta}_t = -\nabla \mathcal{E}(\boldsymbol{\theta}_t)dt + \sqrt{2}d\mathbf{B}_t \tag{12.83}$$

where $\mathbf{B}_t$ represents D-dimensional **Brownian motion**. If we use this to generate the samples, the method is known as the **anadjusted Langevin algorithm** or **ULA** [Par81; RT96].

12.5.7 Connection between SGD and Langevin sampling

In this section, we discuss a deep connection between stochastic gradient descent (SGD) and Langevin sampling, following the presentation of [BZ20, Sec 10.2.3].

Consider the minimization of the additive loss

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta}) \tag{12.84}$$

For example, we may define $\mathcal{L}_n(\boldsymbol{\theta}) = -\log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}.)$ We will use a minibatch approximation to the gradients:

$$\nabla_B \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{B} \sum_{n \in \mathcal{S}} \nabla \mathcal{L}_n(\boldsymbol{\theta}) \tag{12.85}$$

where $\mathcal{S} = \{i_1, \ldots, i_B\}$ is a randomly chosen set of indices of size B. For simplicity of analysis, we assume the indices are chosen with replacement from $\{1, \ldots, N\}$.

Let us define the (scaled) error (due to minibatching) in the estimated gradient by

$$\boldsymbol{v}_t \triangleq \sqrt{\eta}(\nabla \mathcal{L}(\boldsymbol{\theta}_t) - \nabla_B \mathcal{L}(\boldsymbol{\theta}_t)) \tag{12.86}$$

This is called the **diffusion term**. Then we can rewrite the SGD update as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_B \mathcal{L}(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t - \eta \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \sqrt{\eta}\boldsymbol{v}_t \tag{12.87}$$

The diffusion term $\boldsymbol{v}_t$ has mean 0, since

$$\mathbb{E}\left[\nabla_B \mathcal{L}(\boldsymbol{\theta})\right] = \frac{1}{B}\sum_{j=1}^{B} \mathbb{E}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})\right] = \frac{1}{B}\sum_{j=1}^{B} \nabla \mathcal{L}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta}) \tag{12.88}$$

To compute the variance of the diffusion term, note that

$$\mathbb{V}\left[\nabla_B \mathcal{L}(\boldsymbol{\theta})\right] = \frac{1}{B^2}\sum_{j=1}^{B} \mathbb{V}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})\right] \tag{12.89}$$

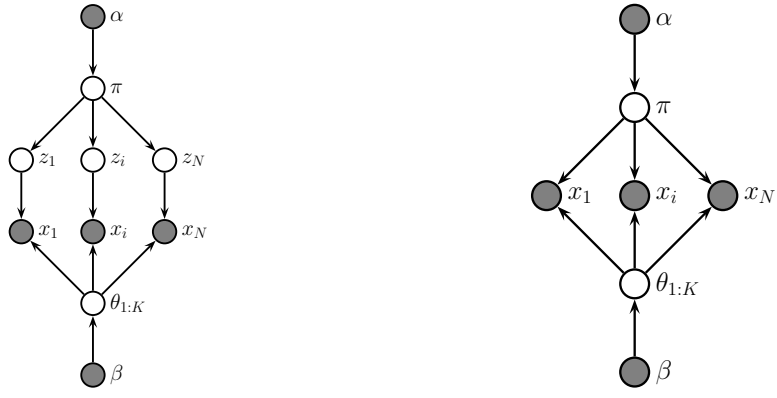


Figure 12.10: (a) A mixture model. (b) After integrating out the discrete latent variables.

where

$$\mathbb{V}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})\right] = \mathbb{E}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta}) \nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})^{\mathsf{T}}\right] - \mathbb{E}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})\right] \mathbb{E}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})^{\mathsf{T}}\right] \tag{12.90}$$

$$= \left(\frac{1}{N}\sum_{n=1}^{N} \nabla \mathcal{L}_n(\boldsymbol{\theta}) \nabla \mathcal{L}_n(\boldsymbol{\theta})^{\mathsf{T}}\right) - \nabla \mathcal{L}(\boldsymbol{\theta}) \nabla \mathcal{L}(\boldsymbol{\theta})^{\mathsf{T}} \triangleq \mathbf{D}(\boldsymbol{\theta}) \tag{12.91}$$

where $\mathbf{D}(\boldsymbol{\theta})$ is called the **diffusion matrix**. Hence $\mathbb{V}[\boldsymbol{v}_t] = \frac{\eta}{B}\mathbf{D}(\boldsymbol{\theta}_t)$.

[LTW15] prove that the following continuous time stochastic differential equation is a first-order approximation of minibatch SGD (assuming the loss function is Lipschitz continuous):

$$d\boldsymbol{\theta}(t) = -\nabla \mathcal{L}(\boldsymbol{\theta}(t))dt + \sqrt{\frac{\eta}{B}\mathbf{D}(\boldsymbol{\theta}_t)}d\mathbf{B}(t) \tag{12.92}$$

where $\mathbf{B}(t)$ is Brownian motion. Thus the noise from minibatching causes SGD to act like a Langevin sampler. (See [Hu+17] for more information.)

The scale factor for the noise, $\tau = \frac{\eta}{B}$, plays the role of **temperature**. Thus we see that using a smaller batch size is like using a larger temperature; the added noise ensures that SGD avoids going into narrow ravines, and instead spends most of its time in flat minima which have better generalization performance [Kes+17]. See Section 17.4.1 for more discussion of this point.

12.5.8 Applying HMC to constrained parameters

To apply HMC, we require that all the latent quantities be continuous (real-valued) and have unconstrained support, i.e., $\boldsymbol{\theta} \in \mathbb{R}^D$, so discrete latent variables need to be marginalized out (although some recent work, such as [NDL20; Zho20], relaxes this requirement).

As an example of how this can be done, consider a GMM. We can easily write the likelihood

without discrete latents as follows:

$$p(\boldsymbol{x}_n|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{12.93}$$

The corresponding "collapsed" model is shown in Figure 12.10(b). (Note that this is the opposite of Section 12.3.8, where we integrated out the continuous parameters in order to apply Gibbs sampling to the discrete latents.) We can apply similar techniques to other discrete latent variable models. For example, to apply HMC to HMMs, we can use the forwards algorithm (Section 9.2.2) to efficiently compute $p(\boldsymbol{x}_n|\boldsymbol{\theta}) = \sum_{\boldsymbol{z}_{1:T}} p(\boldsymbol{x}_n, \boldsymbol{z}_{n,1:T}|\boldsymbol{\theta})$.

In addition to marginalizing out any discrete latent variables, we need to ensure the remaining continuous latent variables are unconstrained. This often requires performing a change of variables using a bijector. For example, instead of sampling the discrete probability vector from the probability simplex $\boldsymbol{\pi} \in \mathbb{S}^K$, we should sample the logits $\boldsymbol{\eta} \in \mathbb{R}^K$. After sampling, we can transform back, since bijectors are invertible. (For a practical example, see change_of_variable_hmc.ipynb.)

12.5.9 Speeding up HMC

Although HMC uses gradient information to explore the typical set, sometimes the geometry of the typical set can be difficult to sample from. See Section 12.5.4.3 for ways to estimate the mass matrix, which can help with such difficult cases.

Another issue is the cost of evaluating the target distribution, $\mathcal{E}(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta})$. For many ML applications, this has the form $\log \tilde{p}(\boldsymbol{\theta}) = \log p_0(\boldsymbol{\theta}) + \sum_{n=1}^{N} \log p(\boldsymbol{\theta}_n|\boldsymbol{\theta})$. This takes $O(N)$ time to compute. We can speed this up by using stochastic gradient methods; see Section 12.7 for details.

12.6 MCMC convergence

We start MCMC from an arbitrary initial state. As we explained in Section 2.6.4, the samples will be coming from the chain's stationary distribution only when the chain has "forgotten" where it started from. The amount of time it takes to enter the stationary distribution is called the mixing time (see Section 12.6.1 for details). Samples collected before the chain has reached its stationary distribution do not come from p^*, and are usually thrown away. The initial period, whose samples will be ignored, is called the **burn-in phase**.

For example, consider a uniform distribution on the integers $\{0, 1, \ldots, 20\}$. Suppose we sample from this using a symmetric random walk. In Figure 12.11, we show two runs of the algorithm. On the left, we start in state 10; on the right, we start in state 17. Even in this small problem it takes over 200 steps until the chain has "forgotten" where it started from. Proposal distributions that make larger changes can converge faster. For example, [BD92; Man] prove that it takes about 7 riffle shuffles to properly mix a deck of 52 cards (i.e., to ensure the distribution is uniform).

In Section 12.6.1 we discuss how to compute the mixing time theoretically. In practice, this can be very hard [BBM10] (this is one of the fundamental weaknesses of MCMC), so in Section 12.6.2, we discuss practical heurstics.

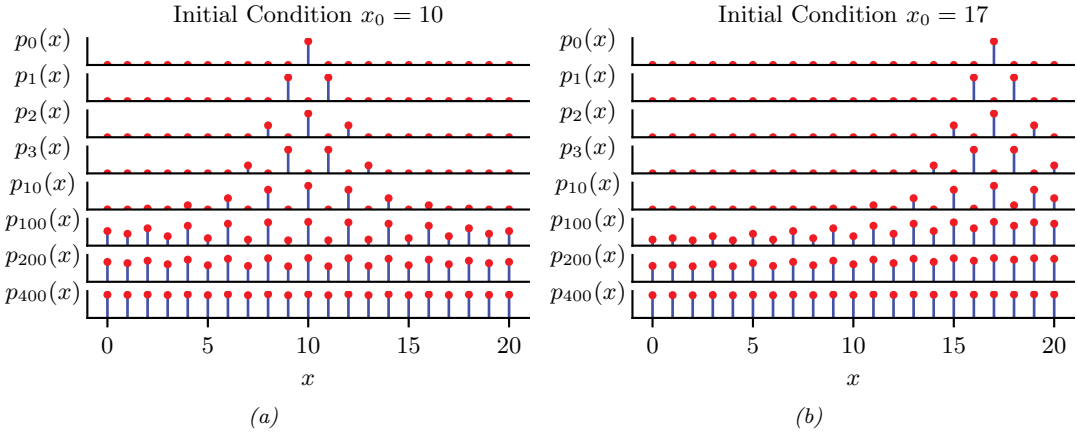


Figure 12.11: Illustration of convergence to the uniform distribution over $\{0, 1, \ldots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Adapted from Figures 29.14 and 29.15 of [Mac03]. Generated by random_walk_integers.ipynb.

12.6.1 Mixing rates of Markov chains

The amount of time it takes for a Markov chain to converge to the stationary distribution, and forget its initial state, is called the **mixing time**. More formally, we say that the mixing time from state x_0 is the minimal time such that, for any constant $\epsilon > 0$, we have that

$$\tau_\epsilon(x_0) \triangleq \min\{t : ||\delta_{x_0}(x)T^t - p^*||_1 \leq \epsilon\} \tag{12.94}$$

where $\delta_{x_0}(x)$ is a distribution with all its mass in state x_0, T is the transition matrix of the chain (which depends on the target p^* and the proposal q), and $\delta_{x_0}(x)T^t$ is the distribution after t steps. The mixing time of the chain is defined as

$$\tau_\epsilon \triangleq \max_{x_0} \tau_\epsilon(x_0) \tag{12.95}$$

This is the maximum amount of time it takes for the chain's distribution to get ϵ close to p^* from any starting state.

The mixing time is determined by the eigengap $\gamma = \lambda_1 - \lambda_2$, which is the difference between the first and second eigenvalues of the transition matrix. For a finie state chain, one cans show $\tau_\epsilon = O(\frac{1}{\gamma} \log \frac{n}{\epsilon})$, where n is the number of states.

We can also study the problem by examining the geometry of the state space. For example, consider the chain in Figure 12.12. We see that the state space consists of two "islands", each of which is connected via a narrow "bottleneck". (If they were completely disconnected, the chain would not be ergodic, and there would no longer be a unique stationary distribution, as discussed in Section 2.6.4.3.) We define the **conductance** ϕ of a chain as the minimum probability, over all subsets S of states, of transitioning from that set to its complement:

$$\phi \triangleq \min_{S:0\leq p^*(S)\leq 0.5} \frac{\sum_{x\in S, x'\in S^c} T(x \to x')}{p^*(S)}, \tag{12.96}$$

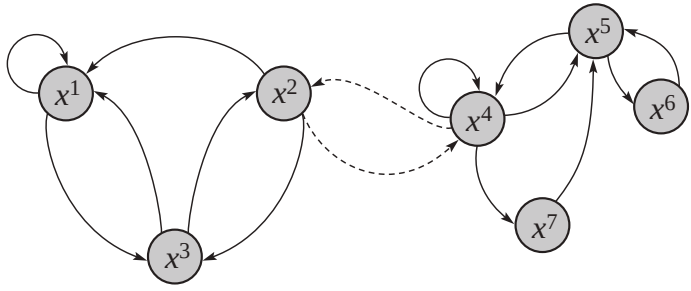


Figure 12.12: A Markov chain with low conductance. The dotted arcs represent transitions with very low probability. From Figure 12.6 of [KF09a]. Used with kind permission of Daphne Koller.

One can show that $\tau_\epsilon \leq O\left(\frac{1}{\phi^2} \log \frac{n}{\epsilon}\right)$. Hence chains with low conductance have high mixing time. For example, distributions with well-separated modes usually have high mixing time. Simple MCMC methods, such as MH and Gibbs, often do not work well in such cases, and more advanced algorithms, such as parallel tempering, are necessary (see e.g., [ED05; Kat+06; BZ20]).

12.6.2 Practical convergence diagnostics

Computing the mixing time of a chain is in general quite difficult, since the transition matrix is usually very hard to compute. Furthermore, diagnosing convergence is computationally intractable in general [BBM10]. Nevertheless, various heuristics have been proposed — see e.g., [Gey92; CC96; BR98; Veh+19]. We discuss some of the current recommended approaches below, following [Veh+19].

12.6.2.1 Trace plots

One of the simplest approaches to assessing if the method has converged is to run multiple chains (typically 3 or 4) from very different **overdispersed** starting points, and to plot the samples of some quantity of interest, such as the value of a certain component of the state vector, or some event such as the value taking on an extreme value. This is called a **trace plot**. If the chain has mixed, it should have "forgotten" where it started from, so the trace plots should converge to the same distribution, and thus overlap with each other.

To illustrate this, we will consider a very simple, but enlightening, example from [McE20, Sec 9.5]. The model is a univariate Gaussian, $y_i \sim \mathcal{N}(\alpha, \sigma)$, with just 2 observations, $y_1 = -1$ and $y_2 = +1$. We first consider a very diffuse prior, $\alpha \sim \mathcal{N}(0, 1000)$ and $\sigma \sim \text{Expon}(0.0001)$, both of which allow for very large values of α and σ. We fit the model using HMC using 3 chains and 500 samples. The result is shown in Figure 12.13. On the right, we show the trace plots for α and σ for 3 different chains. We see that they do not overlap much with each other. In addition, the numerous black vertical lines at the bottom of the plot indicate that HMC had many divergences.

The problem is caused by the overly diffuse priors, which do not get overwhelmed by the likelihood because we only have 2 datapoints. Thus the posterior is also diffuse. We can fix this by using slightly stronger priors, that keep the parameters close to more sensible values. For example, suppose we use $\alpha \sim \mathcal{N}(1, 10)$ and $\sigma \sim \text{Expon}(1)$. Now we get the results in Figure 12.14. On the right we see

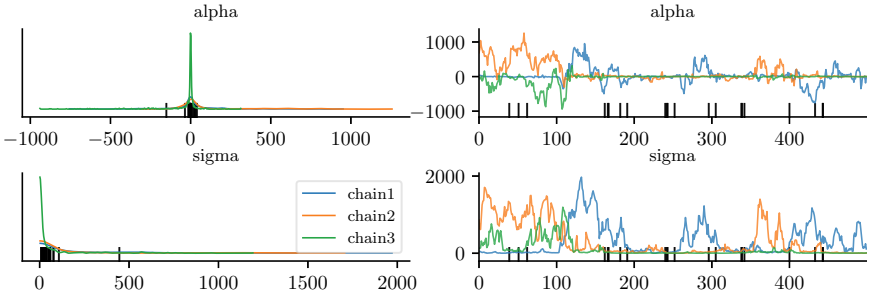


Figure 12.13: Marginals (left) and trace plot (right) for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

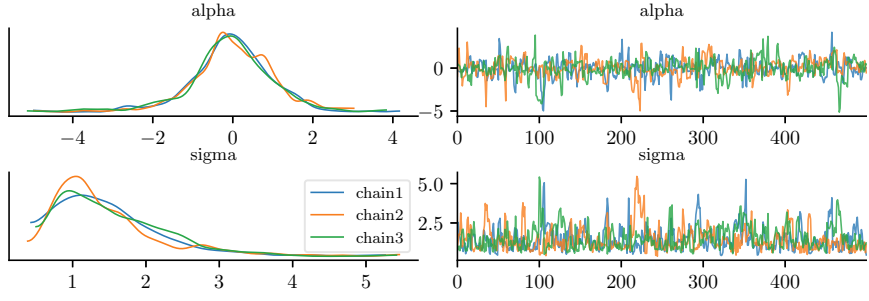


Figure 12.14: Marginals (left) and trace plot (right) for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

that the traceplots overlap. On the left, we see that the marginal distributions from each chain have support over a reasonable interval, and have a peak at the "right" place (the MLE for α is 0, and for σ is 1). And we don't see any divergence warnings (vertical black markers in the plot).

Since trace plots of converging chains correspond to overlapping lines, it can be hard to distinguish success from failure. An alternative plot, known as a **trace rank plot**, was recently proposed in [Veh+19]. (In [McE20], this is called a **trankplot**, a term we borrow.) The idea is to compute the rank of each sample based on all the samples from all the chains, after burnin. We then plot a histogram of the ranks for each chain separately. If the chains have converged, the distribution over ranks should be uniform, since there should be no preference for high or low scoring samples amongst the chains.

The trankplot for the model with the diffuse prior is shown in Figure 12.15. (The x-axis is from 1 to the total number of samples, which in this example is 1500, since we use 3 chains and draw 500 samples from each.) We can see that the different chains are clearly not mixing. The trankplot for the model with the sensible prior is shown in Figure 12.16; this looks much better.

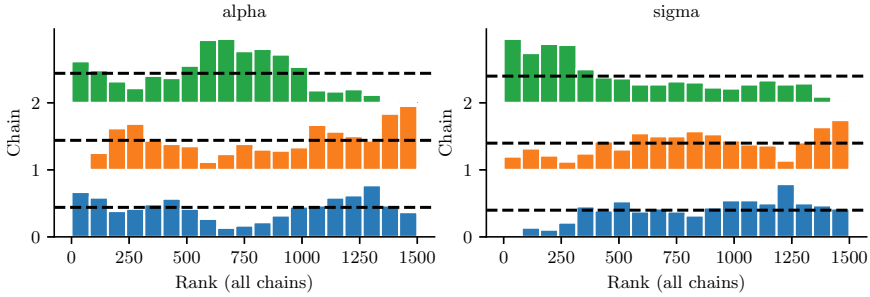


Figure 12.15: Trace rank plot for the univariate Gaussian using the diffuse prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

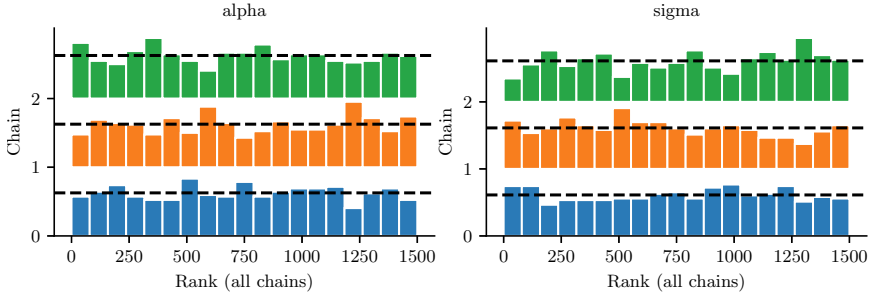


Figure 12.16: Trace rank plot for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

12.6.2.2 Estimated potential scale reduction (EPSR)

In this section, we discuss a way to assess convergence more quantitatively. The basic idea is this: if one or more chains has not mixed well, then the variance of all the chains combined together will be higher than the variance of the individual chains. So we will compare the variance of the quantity of interest computed between and within chains.

More precisely, suppose we have M chains, and we draw N samples from each. Let x_{nm} denote the quantity of interest derived from the n'th sample from the m'th chain. We compute the between- and within-sequence variances as follows:

$$B = \frac{N}{M-1}\sum_{m=1}^{M}(\overline{x}_{\cdot m} - \overline{x}_{\cdot\cdot})^2, \text{ where } \overline{x}_{\cdot m} = \frac{1}{N}\sum_{n=1}^{N} x_{nm}, \;\; \overline{x}_{\cdot\cdot} = \frac{1}{M}\sum_{m=1}^{M}\overline{x}_{\cdot m} \tag{12.97}$$

$$W = \frac{1}{M}\sum_{m=1}^{M} s_m^2, \text{ where } s_m^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_{nm} - \overline{x}_{\cdot m})^2 \tag{12.98}$$

The formula for s_m^2 is the usual unbiased estimate for the variance from a set of N samples; W is just the average of this. The formula for B is similar, but scaled up by N since it is based on the variance of $\overline{x}_{\cdot m}$, which are averaged over N values.

Next we compute the following average variance:

$$\hat{V}^+ \triangleq \frac{N-1}{N}W + \frac{1}{N}B \tag{12.99}$$

Finally, we compute the following quantity, known as the **estimated potential scale reduction** or **R-hat**:

$$\hat{R} \triangleq \sqrt{\frac{\hat{V}^+}{W}} \tag{12.100}$$

In [Veh+19], they recommend checking if $\hat{R} < 1.01$ before declaring convergence.

For example, consider the $\hat{R}$ values for various samplers for our univariate GMM example. In particular, consider the 3 MH samplers in Figure 12.1, and the Gibbs sampler in Figure 12.4. The $\hat{R}$ values are 1.493, 1.039, 1.005, and 1.007. So this diagnostic has correctly identified that the first two samplers are unreliable, which evident from the figure.

In practice, it is recommended to use a slightly different quantity, known as **split-$\hat{R}$**. This can be computed by splitting each chain into the first and second halves, thus doubling the number of chains M (but halving the number of samples N from each), before computing $\hat{R}$. This can detect non-stationarity within a single chain.

12.6.3 Effective sample size

Although MCMC lets us draw samples from a target distribution (assuming it has converged), the samples are not independent, so we may need to draw a lot of them to get a reliable estimate. In this section, we discuss how to compute the **effective sample size** or **ESS** from a set of (possibly correlated) samples.

To start, suppose we draw N *independent* samples from the target distribution, and let $\hat{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ be our empirical estimate of the mean of the quantity of interest. The variance of this estimate is given by

$$\mathbb{V}[\hat{x}] = \frac{1}{N^2}\mathbb{V}\left[\sum_{n=1}^{N} x_n\right] = \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{V}[x_n] = \frac{1}{N}\sigma^2 \tag{12.101}$$

where $\sigma^2 = \mathbb{V}[X]$. If the samples are correlated, the variance of the estimate will be higher, as we show below.

Recall that for N (not necessarily independent) random variables we have

$$\mathbb{V}\left[\sum_{n=1}^{N} x_n\right] = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathrm{Cov}[x_i, x_j] = \sum_{i=1}^{N}\mathbb{V}[x_i] + 2\sum_{1\le i<j\le N}\mathrm{Cov}[x_i, x_j] \tag{12.102}$$

Let $\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ be our estimate based on these correlated samples. The variance of this estimate is given by

$$\mathbb{V}[\overline{x}] = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\mathrm{Cov}[x_i, x_j] \tag{12.103}$$

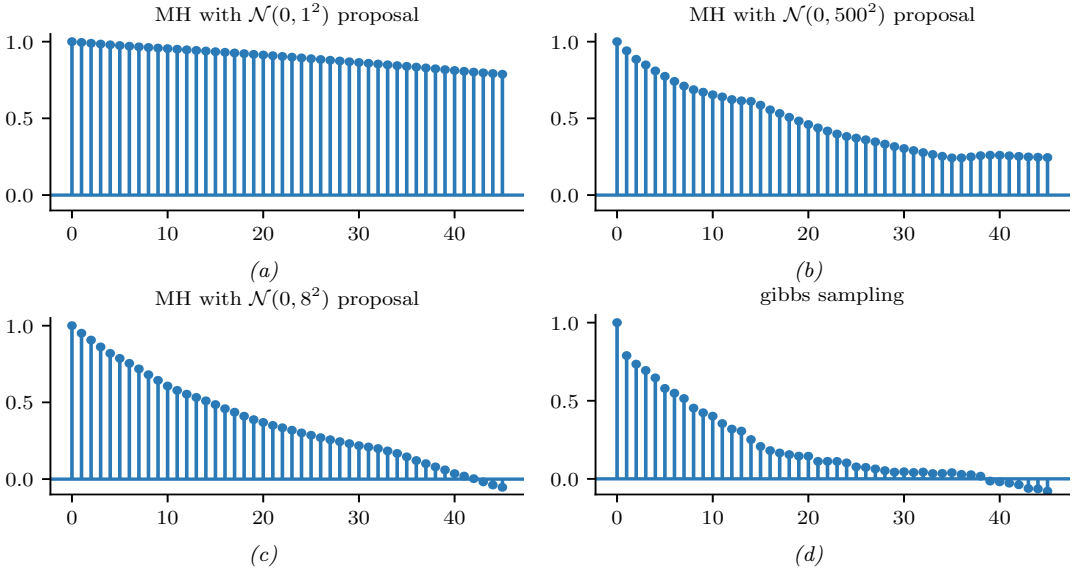


Figure 12.17: Autocorrelation functions for various MCMC samplers for the mixture of two 1d Gaussians. (a-c) These are the MH samplers in Figure 12.1. (d) This is the Gibbs sampler in Figure 12.4. Generated by mcmc_gmm_demo.ipynb.

We now rewrite this in a more convenient form. First recall that the correlation of x_i and x_j is given by

$$\text{corr}\left[x_i, x_j\right] = \frac{\text{Cov}\left[x_i, x_j\right]}{\sqrt{\mathbb{V}\left[x_i\right]\mathbb{V}\left[x_j\right]}} \tag{12.104}$$

Since we assume we are drawing samples from the target distribution, we have $\mathbb{V}\left[x_i\right] = \sigma^2$, where σ^2 is the true variance. Hence

$$\mathbb{V}\left[\overline{x}\right] = \frac{\sigma^2}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\text{corr}\left[x_i, x_j\right] \tag{12.105}$$

For a fixed i, we can think of $\text{corr}\left[x_i, x_j\right]$ as a function of j. This will usually decay as j gets further from i. As $N \to \infty$ we can approximate the sum of correlations by

$$\sum_{j=1}^{N}\text{corr}\left[x_i, x_j\right] \to \sum_{\ell=-\infty}^{\infty}\text{corr}\left[x_i, x_{i+\ell}\right] = 1 + 2\sum_{\ell=1}^{\infty}\text{corr}\left[x_i, x_{i+\ell}\right] \tag{12.106}$$

since $\text{corr}\left[x_i, x_i\right] = 1$ and $\text{corr}\left[x_i, x_{i-\ll}\right] = \text{corr}\left[x_i, x_{i+\ell}\right]$ for lag $\ell > 0$. Since we assume the samples are coming from a stationary distribution, the index i does not matter. Thus we can define the **autocorrelation time** as

$$\rho = 1 + 2\sum_{\ell=1}^{\infty}\rho(\ell) \tag{12.107}$$

where $\rho(\ell)$ is the **autocorrelation function** (ACF), defined as

$$\rho(\ell) \triangleq \text{corr}\left[x_0, x_\ell\right] \tag{12.108}$$

The ACF can be approximated efficiently by convolving the signal $\boldsymbol{x}$ with itself. In Figure 12.17, we plot the ACF for our four samplers for the GMM. We see that the ACF of the Gibbs sampler (bottom right) dies off to 0 much more rapidly than the MH samplers. Intuitively this indicates that each Gibbs sample is "worth" more than each MH sample. We quantify this below.

From Equation (12.105), we can compute the variance of pur estimate in terms of the ACF as follows: $\mathbb{V}\left[\overline{x}\right] = \frac{\sigma^2}{N^2}\sum_{i=1}^{N}\rho = \frac{\sigma^2}{N}\rho$. By contrast, the variance of the estimate from independent samples is $\mathbb{V}\left[\hat{x}\right] = \frac{\sigma^2}{N}$. So we see that the variance is a factor ρ larger when there is correlation. We therefore define the **effective sample size** of our set of samples to be

$$N_{\text{eff}} \triangleq \frac{N}{\rho} = \frac{N}{1 + 2\sum_{\ell=1}^{\infty}\rho(\ell)} \tag{12.109}$$

In practice, we truncate the sum at lag L, which is the last integer at which $\rho(L)$ is positive. Also, if we run M chains, the numerator should be NM, so we get the following estimate:

$$\hat{N}_{\text{eff}} = \frac{NM}{1 + 2\sum_{\ell=1}^{L}\hat{\rho}(\ell)} \tag{12.110}$$

In [Veh+19], they propose various extensions of the above estimator, such as using rank statistics, to make the estimate more robust.

12.6.4 Improving speed of convergence

There are many possible things you could try if the $\hat{R}$ value is too large, and/or the effective sample size is too low. Here is a brief list:

- Try using a non-centered parameterization (see Section 12.6.5).
- Try sampling variables in groups or blocks (see Section 12.3.7).
- Try using Rao-Blackwellization, i.e., analytically integrating out some of the variables (see Section 12.3.8).
- Try adding auxiliary variables (see Section 12.4).
- Try using adaptive proposal distributions (see Section 12.2.3.5).

More details can be found in [Rob+18].

12.6.5 Non-centered parameterizations and Neal's funnel

A common problem that arises when applying sampling to hierarchical Bayesian models is when a set of parameters at one level of the model have a tight depenendence on parameters at the level above. We show some practical examples of this in the hierarchical Gaussian 8-schools example in

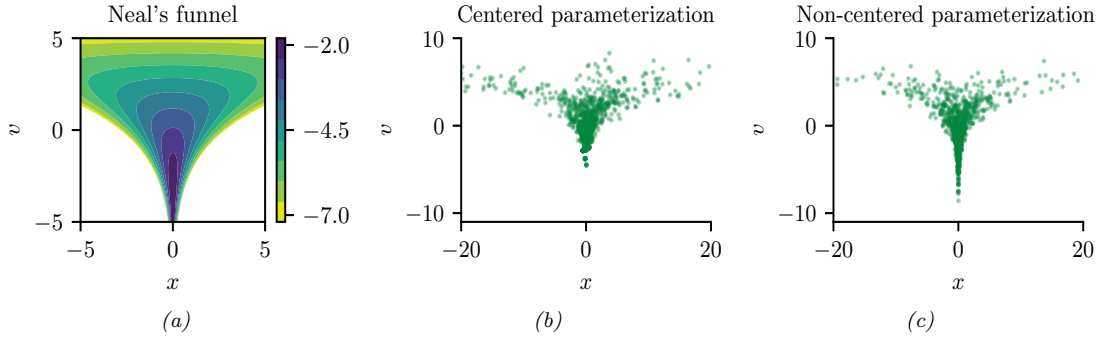


Figure 12.18: Neal's funnel. (a) Joint density. (b) HMC samples from centered representation. (c) HMC samples from non-centered representation. Generated by neals_funnel.ipynb.

Section 3.6.2.2 and the hierarchical radon regression example in Section 15.5.2.2. Here, we focus on the following simple toy model that captures the essence of the problem:

$$\nu \sim \mathcal{N}(0, 3) \tag{12.111}$$

$$x \sim \mathcal{N}(0, \exp(\nu)) \tag{12.112}$$

The corresponding joint density $p(x, \nu)$ is shown in Figure 12.18a. This is known **Neal's funnel**, named after [Nea03]. It is hard for a sampler to "descend" in the narrow "neck" of the distribution, corresponding to areas where the variance ν is small [BG13].

Fortunately, we can represent this model in an equivalent way that makes it easier to sample from, providing we use a **non-centered parameterization** [PR03]. This has the form

$$\nu \sim \mathcal{N}(0, 3) \tag{12.113}$$

$$z \sim \mathcal{N}(0, 1) \tag{12.114}$$

$$x = z \exp(\nu) \tag{12.115}$$

This is easier to sample from, since $p(z, \nu)$ is a product of 2 independent Gaussians, and we can derive x deterministically from these Gaussian samples. The advantage of this reparameterization is shown in Figure 12.18. A method to automatically derive such reparameterizations is discussed in [GMH20].

12.7 Stochastic gradient MCMC

Consider an unnormalized target distribution of the following form:

$$\pi(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}, \mathcal{D}) = p_0(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{\theta}) \tag{12.116}$$

where $\mathcal{D} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. Alternatively we can define the target distribution in terms of an energy function (negative log joint) as follows:

$$p(\boldsymbol{\theta}, \mathcal{D}) \propto \exp(-\mathcal{E}(\boldsymbol{\theta})) \tag{12.117}$$

The energy function can be decomposed over data samples:

$$\mathcal{E}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{E}_n(\boldsymbol{\theta}) \tag{12.118}$$

$$\mathcal{E}_n(\boldsymbol{\theta}) = -\log p(\boldsymbol{x}_n|\boldsymbol{\theta}) - \frac{1}{N}\log p_0(\boldsymbol{\theta}) \tag{12.119}$$

Evaluating the full energy (e.g., to compute an acceptance probability in the Metropolis-Hastings algorithm, or to compute the gradient in HMC) takes $O(N)$ time, which does not scale to large data. In this section, we discuss some solutions to this problem.

12.7.1 Stochastic gradient Langevin dynamics (SGLD)

Recall from Equation (12.83) that the **Langevin diffusion** SDE has the following form

$$d\boldsymbol{\theta}_t = -\nabla\mathcal{E}(\boldsymbol{\theta}_t)dt + \sqrt{2}d\mathbf{W}_t \tag{12.120}$$

where $d\mathbf{W}_t$ is a Wiener noise (also called Brownian noise) process. In discrete time, we can use the following Euler approximation:

$$\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t - \eta_t\nabla\mathcal{E}(\boldsymbol{\theta}_t) + \sqrt{2\eta_t}\mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{12.121}$$

Computing the gradient $\boldsymbol{g}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\boldsymbol{\theta}_t)$ at each step takes $O(N)$ time. We can compute an unbiased minibatch approximation to the gradient term in $O(B)$ time using

$$\hat{\boldsymbol{g}}(\boldsymbol{\theta}_t) = \frac{N}{B}\sum_{n\in\mathcal{B}_t}\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) = -\frac{N}{B}\left(\sum_{n\in\mathcal{B}_t}\nabla\log p(\boldsymbol{x}_n|\boldsymbol{\theta}_t) + \frac{B}{N}\nabla\log p_0(\boldsymbol{\theta}_t)\right) \tag{12.122}$$

where $\mathcal{B}_t$ is the minibatch at step t. This gives rise to the following approximate update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t\hat{\boldsymbol{g}}(\boldsymbol{\theta}_t) + \sqrt{2\eta_t}\mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{12.123}$$

This is called **stochastic gradient Langevin dynamics** or **SGLD** [Wel11]. The resulting update step is identical to SGD, except for the addition of a Gaussian noise term. (See [Neg+21] for some recent analysis of this method; they also suggest setting $\eta_t \propto N^{-2/3}$.)

12.7.2 Preconditionining

As in SGD, we can get better results (especially for models such as neural networks) if we use preconditioning to scale the gradient updates. In [PT13], they use the Fisher information matrix (FIM) as the preconditioner; this method is known as **stochastic gradient Riemannian Langevin dynamics** or **SGRLD**.

Unfortunately, computing the FIM is often hard. In [Li+16], they propose to use the same kind of diagonal approximation as used by RMSprop; this is called **preconditioned SGLD**. An alternative is to use an Adam-like preconditioner, as proposed in [KSL21]. This is called **SGLD-Adam**. For more details, see [CSN21].

12.7.3 Reducing the variance of the gradient estimate

The variance of the noise introduced by minibatching can be quite large, which can hurt the performance of methods such as SGLD [BDM18]. In [Bak+17], they propose to reduce the variance of this estimate by using a **control variate** estimator; this method is therefore called **SGLD-CV**. Specifically they use the following gradient approximation:

$$\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \frac{N}{B}\sum_{n\in\mathcal{S}_t}\Big(\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})\Big) \tag{12.124}$$

Here $\hat{\boldsymbol{\theta}}$ is any fixed value, but it is often taken to be an approximate MAP estimate (e.g., based on one epoch of SGD). The reason Equation (12.124) is valid is because the terms we add and subtract are equal in expectation, and hence we get an unbiased estimate:

$$\mathbb{E}\left[\hat{\nabla}_{cv}\mathcal{E}(\boldsymbol{\theta}_t)\right] = \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \mathbb{E}\left[\frac{N}{B}\sum_{n\in\mathcal{S}_t}\Big(\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})\Big)\right] \tag{12.125}$$

$$= \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) + \nabla\mathcal{E}(\boldsymbol{\theta}_t) - \nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) = \nabla\mathcal{E}(\boldsymbol{\theta}_t) \tag{12.126}$$

Note that the first term, $\nabla\mathcal{E}(\hat{\boldsymbol{\theta}}) = \sum_{n=1}^{N}\nabla\mathcal{E}_n(\hat{\boldsymbol{\theta}})$, requires a single pass over the entire dataset, but only has to be computed once (e.g., while estimating $\hat{\boldsymbol{\theta}}$).

One disadvantage of SGLD-CV is that the reference point $\hat{\boldsymbol{\theta}}$ has to be precomputed, and is then fixed. An alternative is to update the reference point online, by performing periodic full batch estimates. This is called **SVRG-LD** [Dub+16; Cha+18], where SVRG stands for stochastic variance reduced gradient, and LD stands for Langevin dynamics. If we use $\tilde{\boldsymbol{\theta}}_t$ to denote the most recent snapshot (reference point), the corresponding gradient estimate is given by

$$\hat{\nabla}_{svrg}\mathcal{E}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\tilde{\boldsymbol{\theta}}_t) + \frac{N}{B}\sum_{n\in\mathcal{S}_t}\Big(\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}}_t)\Big) \tag{12.127}$$

We recompute the snapshot every τ steps (known as the epoch length). See Algorithm 12.5 for the pseudo-code.

Algorithm 12.5: SVRG Langevin descent

Initialize $\boldsymbol{\theta}_0$
for $t = 1 : T$ **do**
  if $t \mod \tau = 0$ **then**
    $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_t$
    $\tilde{\boldsymbol{g}} = \sum_{n=1}^{N}\mathcal{E}_n(\tilde{\boldsymbol{\theta}})$
  Sample minibatch $\mathcal{B}_t \in \{1,\ldots,N\}$
  $\boldsymbol{g}_t = \tilde{\boldsymbol{g}} + \frac{N}{B}\sum_{n\in\mathcal{B}_t}\Big(\nabla\mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla\mathcal{E}_n(\tilde{\boldsymbol{\theta}})\Big)$
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t\boldsymbol{g}_t + \sqrt{2\eta_t}\mathcal{N}(\mathbf{0},\mathbf{I})$

The disadvantage of SVRG is that it needs to perform a full pass over the data every τ steps. An alternative approach, called **SAGA-LD** [Dub+16; Cha+18] (which stands for stochastic averaged gradient acceleration), avoids this by storing all N gradient vectors, and then doing incremental updates. Unfortunately the memory requirements of this algorithm usually make it impractical.

12.7.4 SG-HMC

We discussed Hamiltonian Monte Carlo (HMC) in Section 12.5, which uses auxiliary momentum variables to improve performance over Langevin MC. In this section, we discuss a way to speed it up by approximating the gradients using minibatches. This is called **SG-HMC** [CFG14; ZG21], where SG stands for "stochastic gradient".

Recall that the leapfrog updates have the following form:

$$\boldsymbol{v}_{t+1/2} = \boldsymbol{v}_t - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_t) \tag{12.128}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta\boldsymbol{v}_{t+1/2} = \boldsymbol{\theta}_t + \eta\boldsymbol{v}_t - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_t) \tag{12.129}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_{t+1/2} - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_{t+1}) = \boldsymbol{v}_t - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_t) - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_{t+1}) \tag{12.130}$$

We can replace the full batch gradient with a stochastic approximation, to get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta\boldsymbol{v}_t - \frac{\eta^2}{2}\boldsymbol{g}(\boldsymbol{\theta}_t, \boldsymbol{\xi}_t) \tag{12.131}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \frac{\eta}{2}\boldsymbol{g}(\boldsymbol{\theta}_t, \boldsymbol{\xi}_t) - \frac{\eta}{2}\boldsymbol{g}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\xi}_{t+1/2}) \tag{12.132}$$

where $\boldsymbol{\xi}_t$ and $\boldsymbol{\xi}_{t+1/2}$ are independent sources of randomness (e.g., batch indices). In [ZG21], they show that this algorithm (even without the MH rejection step) provides a good approximation to the posterior (in the sense of having small Wasserstein-2 distance) for the case where the energy functon is strongly convex. Furthermore, performance can be considerably improved if we use the variance reduction methods discussed in Section 12.7.3.

12.7.5 Underdamped Langevin dynamics

The **underdamped Langevin dynamics** (**ULD**) has the form of the following SDE [CDC15; LMS16; Che+18a; Che+18d]:

$$\begin{aligned} d\boldsymbol{\theta}_t &= \boldsymbol{v}_t dt \\ d\boldsymbol{v}_t &= -\boldsymbol{g}(\boldsymbol{\theta}_t)dt - \gamma\boldsymbol{v}_t dt + \sqrt{2\gamma}d\mathbf{W}_t \end{aligned} \tag{12.133}$$

where $\boldsymbol{g}(\boldsymbol{\theta}_t) = \nabla\mathcal{E}(\boldsymbol{\theta}_t)$ is the gradient or **force** acting on the particle, $\gamma > 0$ is the **friction** parameter, and $d\mathbf{W}_t$ is Wiener noise.

Equation (12.133) is like the Langevin dynamics of Equation (12.83) but with an added momentum term $\boldsymbol{v}_t$. We can solve the dynamics using various integration methods. It can be shown (see e.g., [LMS16]) that these methods are accurate to second order, whereas solving standard (overdamped) Langevin is only accurate to first order, and thus will require more sampling steps to achieve a given accuracy.

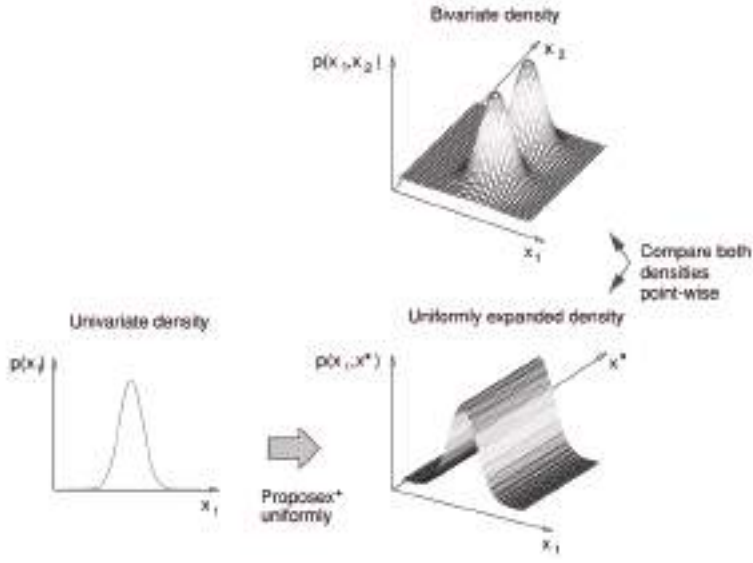


Figure 12.19: To compare a 1d model against a 2d model, we first have to map the 1d model to 2d space so the two have a common measure. Note that we assume the ridge has finite support, so it is integrable. From Figure 17 of [And+03]. Used with kind permission of Nando de Freitas.

12.8 Reversible jump (transdimensional) MCMC

Suppose we have a set of models with different numbers of parameters, e.g., mixture models in which the number of mixture components is unknown. Let the model be denoted by m, and let its unknowns (e.g., parameters) be denoted by $\boldsymbol{x}_m \in \mathcal{X}_m$ (e.g., $\mathcal{X}_m = \mathbb{R}^{n_m}$, where n_m is the dimensionality of model m). Sampling in spaces of differing dimensionality is called **trans-dimensional MCMC**. We could sample the model indicator $m \in \{1, \ldots, M\}$ and sample all the parameters from the product space $\prod_{m=1}^{M} \mathcal{X}_m$, but this is very inefficient, and only works if M is finite. It is more parsimonious to sample in the union space $\mathcal{X} = \cup_{m=1}^{M} \{m\} \times \mathcal{X}_m$, where we only worry about parameters for the currently active model.

The difficulty with this approach arises when we move between models of different dimensionality. The trouble is that when we compute the MH acceptance ratio, we are comparing densities defined on spaces of different dimensionality, which is not well defined. For example, comparing densities on two points of a sphere makes sense, but comparing a density on a sphere to a density on a circle does not, as there is a dimensional mismatch in the two concepts. The solution, proposed by [Gre95] and known as **reversible jump MCMC** or **RJMCMC**, is to augment the low dimensional space with extra random variables so that the two spaces have a common measure. This is illustrated in Figure 12.19.

We give a sketch of the algorithm below. For more details, see e.g., [Gre03; HG12].

12.8.1 Basic idea

To explain the method in more detail, we follow the presentation of [And+03]. To ensure a common measure, we need to define a way to extend each pair of subspaces $\mathcal{X}_m$ and $\mathcal{X}_n$ to $\mathcal{X}_{m,n} = \mathcal{X}_m \times \mathcal{U}_{m,n}$ and $\mathcal{X}_{n,m} = \mathcal{X}_n \times \mathcal{U}_{n,m}$. We also need to define a deterministic, differentiable and invertible mapping

$$(\boldsymbol{x}_m, \boldsymbol{u}_{m,n}) = f_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m}) = (f^x_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m}), f^u_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m})) \tag{12.134}$$

Invertibility means that

$$f_{m\to n}(f_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m})) = (\boldsymbol{x}_n, \boldsymbol{u}_{n,m}) \tag{12.135}$$

Finally, we need to define proposals $q_{n\to m}(\boldsymbol{u}_{n,m}|n, \boldsymbol{x}_n)$ and $q_{m\to n}(\boldsymbol{u}_{m,n}|m, \boldsymbol{x}_m)$.

Suppose we are in state $(n, \boldsymbol{x}_n)$. We move to $(m, \boldsymbol{x}_m)$ by generating $\boldsymbol{u}_{n,m} \sim q_{n\to m}(\cdot|n, \boldsymbol{x}_n)$, and then computing $(\boldsymbol{x}_m, \boldsymbol{u}_{m,n}) = f_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m})$. We then accept the move with probability

$$A_{n\to m} = \min\left\{1, \frac{p(m, \boldsymbol{x}^*_m)}{p(n, \boldsymbol{x}_n)} \times \frac{q(n|m)}{q(m|n)} \times \frac{q_{m\to n}(\boldsymbol{u}_{m,n}|m, \boldsymbol{x}^*_m)}{q_{n\to m}(\boldsymbol{u}_{n,m}|n, \boldsymbol{x}_n)} \times |\det \mathbf{J}_{f_{m\to n}}|\right\} \tag{12.136}$$

where $\boldsymbol{x}^*_m = f^x_{n\to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m})$, $\mathbf{J}_{f_{m\to n}}$ is the Jacobian of the transformation

$$J_{f_{m\to n}} = \frac{\partial f_{n\to m}(\boldsymbol{x}_m, \boldsymbol{u}_{m,n})}{\partial(\boldsymbol{x}_m, \boldsymbol{u}_{m,n})} \tag{12.137}$$

and $|\det \mathbf{J}|$ is the absolute value of the determinant of the Jacobian.

12.8.2 Example

Let us consider an example from [AFD01]. They consider an RBF network for nonlinear regression of the form

$$f(\boldsymbol{x}) = \sum_{j=1}^{k} a_j \mathcal{K}(||\boldsymbol{x} - \boldsymbol{\mu}_j||) + \boldsymbol{\beta}^{\mathsf{T}}\boldsymbol{x} + \beta_0 + \epsilon \tag{12.138}$$

where $\mathcal{K}()$ is some kernel function (e.g., a Gaussian), k is the number of such basis functions, and ϵ is a Gaussian noise term. If $k = 0$, the model corresponds to linear regression.

They fit this model to the data in Figure 12.20(a). The predictions on the test set are shown in Figure 12.20(b). Estimates of $p(k|\mathcal{D})$, the (distribution over the) number of basis functions, are shown in Figure 12.20(c) as a function of the iteration number; the posterior at the final iteration is shown in Figure 12.20(d). There is clearly the most posterior support for $k = 2$, which makes sense given the two "bumps" in the data.

To generate these results, they consider several kinds of proposal. One of them is to split a current basis function μ into two new ones using

$$\mu_1 = \mu - u_{n,n+1}\alpha, \;\; \mu_2 = \mu + u_{n,n+1}\alpha \tag{12.139}$$

where α is a parameter of the proposal, and $u_{n,m}$ is sampled from some distribution (e.g., uniform). To ensure reversibility, they define a corresponding merge move

$$\mu = \frac{\mu_1 + \mu_2}{2} \tag{12.140}$$

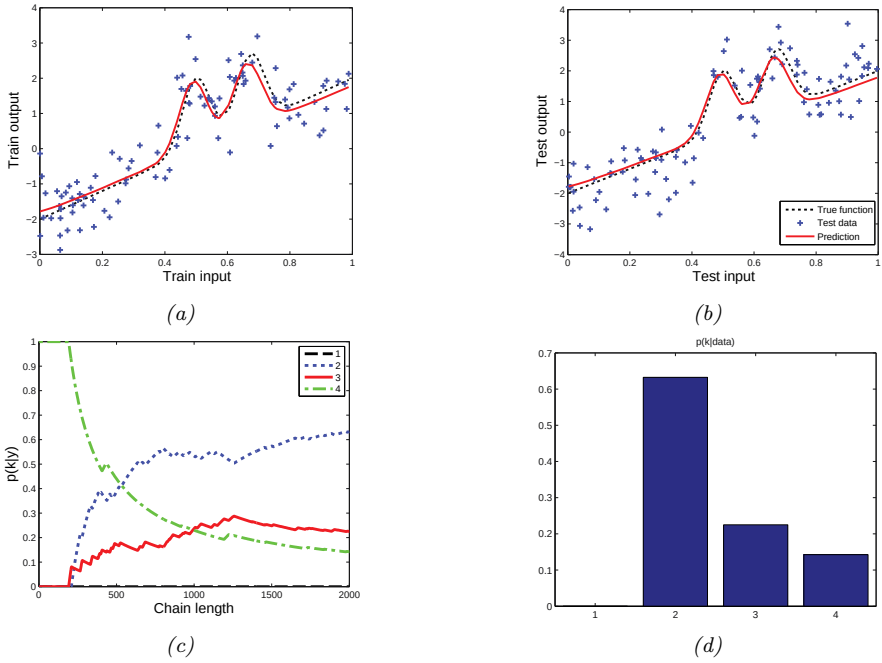


Figure 12.20: Fitting an RBF network to some 1d data using RJMCMC. (a) Prediction on train set. (b) Prediction on test set. (c) Plot of $p(k|\mathcal{D})$ vs iteration. (d) Final posterior $p(k|\mathcal{D})$. Adapted from Figure 4 of [AFD01]. Generated by rjmcmc_rbf, written by Nando de Freitas.

where μ_1 is chosen at random, and μ_2 is its nearest neighbor. To ensure these moves are reversible, we require $||\mu_1 - \mu_2|| < 2\beta$.

The acceptance ratio for the split move is given by

$$A_{split} = \min\left\{1, \frac{p(k+1, \mu_{k+1})}{p(k, \mu_{k+1})} \times \frac{1/(k+1)}{1/k} \times \frac{1}{p(u_{n,m})} \times |\det \mathbf{J}_{split}|\right\} \tag{12.141}$$

where $1/k$ is the probability of choosing one of the k bases uniformly at random. The Jacobian is

$$\mathbf{J}_{split} = \frac{\partial(\mu_1, \mu_2)}{\partial(\mu, u_{n,m})} = \det \begin{pmatrix} 1 & 1 \\ -\beta & \beta \end{pmatrix} \tag{12.142}$$

so $|\det \mathbf{J}_{split}| = 2\beta$. The acceptance ratio for the merge move is given by

$$A_{merge} = \min\left\{1, \frac{p(k-1, \mu_{k-1})}{p(k, \mu_k)} \times \frac{1/(k-1)}{1/k} \times |\det \mathbf{J}_{merge}|\right\} \tag{12.143}$$

where $|\det \mathbf{J}_{merge}| = 1/(2\beta)$.

The overall pseudo-code for the algorithm, assuming the current model has index k, is given in Algorithm 12.6. Here b_k is the probability of a birth move, d_k is the probability of a death move, s_k

Algorithm 12.6: Generic reversible jump MCMC (single step)

Sample $u \sim U(0,1)$
If $u \leq b_k$
then birth move
else if $u \leq (b_k + d_k)$ then death move
else if $u \leq (b_k + d_k + s_k)$ then split move
else if $u \leq (b_k + d_k + s_k + m_k)$ then merge move
else update parameters

is the probability of a split move, and m_k is the probability of a merge move. If we don't make a dimension-changing move, we just update the parameters of the current model using random walk MH.

12.8.3 Discussion

RJMCMC algorithms can be quite tricky to implement. If, however, the continuous parameters can be integrated out (resulting in a method called collapsed RJMCMC), much of the difficulty goes away, since we are just left with a discrete state space, where there is no need to worry about change of measure. For example, if we fix the centers $\boldsymbol{\mu}_j$ in Equation (12.138) (e.g., using samples from the data, or using K-means clustering), we are left with a linear model, where we can integrate out the parameters. All that is left to do is sample which of these fixed basis functions to include in the model, which is a discrete variable selection problem. See e.g., [Den+02] for details.

In Chapter 31, we discuss Bayesian nonparametric models, which allow for an infinite number of different models. Surprisingly, such models are often easier to deal with computationally (as well as more realistic, statistically) than working with a finite set of different models.

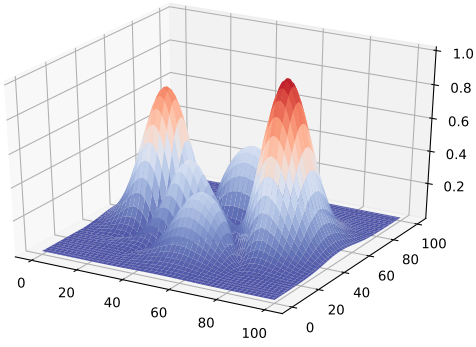
12.9 Annealing methods

Many distributions are multimodal and hence hard to sample from. However, by analogy to the way metals are heated up and then cooled down in order to make the molecules align, we can imagine using a computational temperature parameter to "smooth out" a distribution, gradually cooling it to recover the original "bumpy" distribution. We first explain this idea in more detail in the context of an algorithm for MAP estimation. We then discuss extensions to the sampling case.

12.9.1 Simulated annealing

In this section, we discuss the **simulated annealing** algorithm [KJV83; LA87], which is a variant of the Metropolis-Hastings algorithm which is designed to find the global optimum of blackbox function. (Other approaches to blackbox optimization are discussed in Section 6.7.)

Annealing is a physical process of heating a solid until thermal stresses are released, then cooling it very slowly until the crystals are perfectly arranged, acheiving a minimum energy state. Depending on how fast or slow the temperature is cooled, the results will have better or worse quality. We can apply this approach to probability distributions, to control the number of modes (low energy states)



(a) (b)

Figure 12.21: (a) A peaky distribution. (b) Corresponding energy function. Generated by simulated_annealing_2d_demo.ipynb.

that they have, by defining

$$p_T(\boldsymbol{x}) = \exp(-\mathcal{E}(\boldsymbol{x})/T) \tag{12.144}$$

where T is the temperature, which is reduced over time. As an example, consider the **peaks function**:

$$p(x,y) \propto |3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}| \tag{12.145}$$

This is plotted in Figure 12.21a. The corresponding energy is in Figure 12.21b. We plot annealed versions of this distribution in Figure 12.22. At high temperatures, $T \gg 1$, the surface is approximately flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools, the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is possible to "track" the largest peak, and thus find the global optimum (minimum energy state). This is an example of a **continuation method**.

In more detail, at each step, we sample a new state according to some proposal distribution $\boldsymbol{x}' \sim q(\cdot|\boldsymbol{x}_t)$. For real-valued parameters, this is often simply a random walk proposal centered on the current iterate, $\boldsymbol{x}' = \boldsymbol{x}_t + \boldsymbol{\epsilon}_{t+1}$, where $\boldsymbol{\epsilon}_{t+1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. (The matrix $\boldsymbol{\Sigma}$ is often diagonal, and may be updated over time using the method in [Cor+87].) Having proposed a new state, we compute the acceptance probability

$$\alpha_{t+1} = \exp\left(-(\mathcal{E}(\boldsymbol{x}') - \mathcal{E}(\boldsymbol{x}_t))/T_t\right) \tag{12.146}$$

where T_t is the temperature of the system. We then accept the new state (i.e., set $\boldsymbol{x}_{t+1} = \boldsymbol{x}'$) with probability $\min(1, \alpha_{t+1})$, otherwise we stay in the current state (i.e., set $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t$). This means that if the new state has lower energy (is more probable), we will definitely accept it, but if it has higher energy (is less probable), we might still accept, depending on the current temperature. Thus the algorithm allows "downhill" moves in probability space (uphill in energy space), but less frequently as the temperature drops.

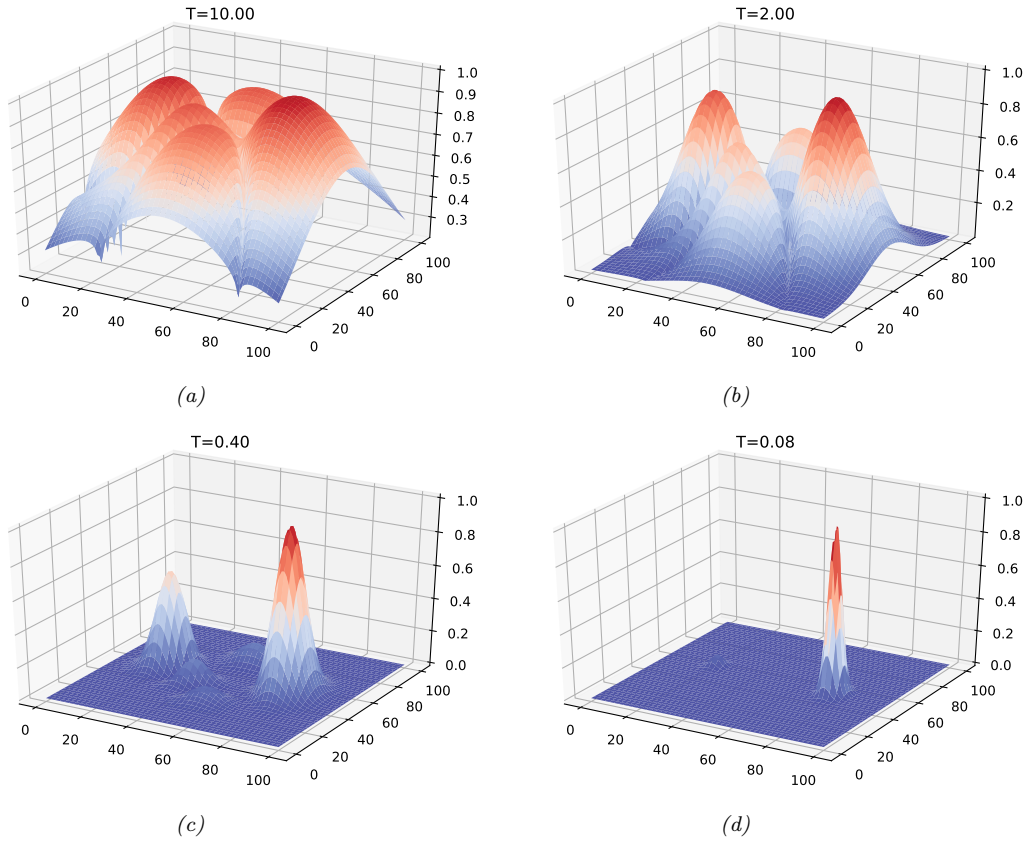


Figure 12.22: Annealed version of the distribution in Figure 12.21a at different temperatures. Generated by simulated_ annealing_ 2d_ demo.ipynb.

The rate at which the temperature changes over time is called the **cooling schedule**. It has been shown [Haj88] that if one cools according to a logarithmic schedule, $T_t \propto 1/\log(t+1)$, then the method is guaranteed to find the global optimum under certain assumptions. However, this schedule is often too slow. In practice it is common to use an **exponential cooling schedule** of the form $T_{t+1} = \gamma T_t$, where $\gamma \in (0, 1]$ is the cooling rate. Cooling too quickly means one can get stuck in a local maximum, but cooling too slowly just wastes time. The best cooling schedule is difficult to determine; this is one of the main drawbacks of simulated annealing.

In Figure 12.23a, we show a cooling schedule using $\gamma = 0.9$. If we combine this with a Gaussian random walk proposal with $\sigma = 10$ to the peaky distribution in Figure 12.21a, we get the results shown in Figure 12.23 and Figure 12.23b. We see that the algorithm concentrates its samples near the global optimum (the peak on the middle right).

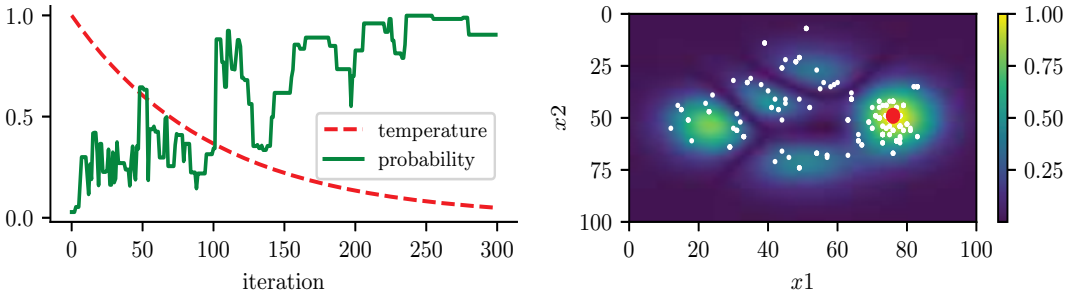


Figure 12.23: Simulated annealing applied to the distribution in Figure 12.21a. (a) Temperature vs iteration and probability of each visited point vs iteration. (b) Visited samples, superimposed on the target distribution. The big red dot is the highest probability point found. Generated by simulated_annealing_2d_demo.ipynb.

12.9.2 Parallel tempering

Another way to combine MCMC and annealing is to run multiple chains in parallel at different temperatures, and allow one chain to sample from another chain at a neighboring temperature. In this way, the high temperature chain can make long distance moves through the state space, and have this influence lower temperature chains. This is known as **parallel tempering**. See e.g., [ED05; Kat+06] for details.