where

$$\boldsymbol{m} = \int_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}) p(\boldsymbol{x}) \tag{8.211a}$$

$$\mathbf{S} = \int_{\boldsymbol{x}} (\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{m})(\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{m})^{\mathsf{T}} p(\boldsymbol{x}) \tag{8.211b}$$

$$\mathbf{C} = \int_{\boldsymbol{x}} (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{m})^{\mathsf{T}} p(\boldsymbol{x}) \tag{8.211c}$$

We can compute these expectations with a variety of methods, as we discuss in Section 8.8.3. See Algorithm 14 for the pseudocode.

Once we have estimated $\boldsymbol{\theta} = (\mathbf{A}, \boldsymbol{b}, \boldsymbol{\Omega})$, the joint distribution follows from Equation (2.81):

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \mathbf{A}\boldsymbol{\mu} + \boldsymbol{b} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \boldsymbol{\Sigma}\mathbf{A}^{\mathsf{T}} \\ \mathbf{A}\boldsymbol{\Sigma} & \boldsymbol{\Omega} + \boldsymbol{\Omega} + \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathsf{T}} \end{pmatrix} \right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{m} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{C} \\ \mathbf{C}^{\mathsf{T}} & \boldsymbol{\Omega} + \mathbf{S} \end{pmatrix} \right) \tag{8.212}$$

We see that this matches the moments of $p(\boldsymbol{x}, \boldsymbol{y})$, and hence minimizes the KL divergence.

Algorithm 14: Statistical linear regression.

```
def SLR(μ, Σ, m, S, C) :
  Â = C^T Σ^{-1}
  b̂ = m − Âμ
  Ω̂ = S − ÂΣÂ^T
  Return (Â, b̂, Ω̂)
```

8.9.2 Prior linearization filter

We now discuss how to use SLR to perform online filtering. When computing the expectations, we use the prior predictive disribution. Hence this is called the **prior linearization filter** or **PrLF** [GFSS17]. We will use other distributions for the expectations in later sections.

In the prediction step, we approximate $p(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t | \boldsymbol{y}_{1:t-1})$ and then we compute the marginal distribution $p(\boldsymbol{z}_t | \boldsymbol{y}_{1:t-1})$. For notational simplicity, we use the UKF in the predict step:

$$(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \mathbf{UnscentedPredict}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}(\cdot, \boldsymbol{u}_t), \mathbf{Q}_t) \tag{8.213}$$

In the update step, we approximate $p(\boldsymbol{z}_t, \boldsymbol{y}_t | \boldsymbol{y}_{1:t-1})$ by applying the SLR transform to the observation model, and then we perform Gaussian updating:

$$(\boldsymbol{m}, \mathbf{S}, \mathbf{C}) = \mathbf{UnscentedPredict}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{0}) \tag{8.214}$$

$$(\mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t) = \mathbf{SLR}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \boldsymbol{m}, \mathbf{S}, \mathbf{C}) \tag{8.215}$$

$$(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t}, \ell_t) = \mathbf{LinGaussUpdate}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, \mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t + \mathbf{R}_t, \boldsymbol{y}_t) \tag{8.216}$$

where $\tilde{\mathbf{R}}_t$ is the extra covariance introduced by approximating the observation model.

A special case of the prior linearization filter arises if we set $\tilde{\mathbf{R}}_t = \mathbf{0}$, which ignores extra uncertainty introduced by the linear approximation. This simpler case is known as the **statistically linearized filter** [Gel74] (see also [Sar13, Sec 5.4]).

8.9.3 Iterated posterior linearization filter

In [GF+15] they propose a method called the **posterior linearization filter**, which uses SLR, but where the expectations for the measurement update step are taken with respect to the posterior instead of the prior. This ensures that the observation model is linearized near the location where the posterior has most of its mass.

Of course, this approach cannot be implemented as stated, because it requires knowledge of the posterior to approxmate the posterior. However, we can approximate it by an iterative method, as shown in Algorithm 15, where J is the number of iterations. This is called the **iterated posterior linearization filter** or **IPLF**. (If we set $J = 1$, we recover the prior linearization filter.)

Algorithm 15: Iterated posterior linearization filter.

def **IPLF**$(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{h}, \mathbf{R}, \boldsymbol{y}_{1:T}, J)$:
  foreach $t = 1 : T$ **do**
    Predict step:
    $(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \textbf{UnscentedPredict}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}, \mathbf{Q}_t)$
    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**
      $(\boldsymbol{m}, \mathbf{S}, \mathbf{C}) = \textbf{UnscentedPredict}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{0})$
      $(\mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t) = \textbf{SLR}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{m}, \mathbf{S}, \mathbf{C})$
      $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \ell_t) = \textbf{LinGaussUpdate}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t + \mathbf{R}_t, \boldsymbol{y}_t)$
  Return $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})_{t=1}^T$, $\sum_{t=1}^T \ell_t$

The iterated EKF (Section 8.5.3) is similar to the IPLF, except it uses a first order Taylor series approximation of the measurement function to compute the moments (rather than a sigma point approximation), and it ignores the uncertainty due to linearization. This is much faster than IPLF. Unfortunately both IEKF and IPLF can diverge. A robust IEKF method, that uses line search to perform damped (partial) updates of $\boldsymbol{\mu}_{t|t}^j$, is presented in [SHA15]. In [Rai+18b], a similar robust method is derived for IPLF.

8.9.4 Iterated posterior linearization smoother

In [GFSS17] they propose a method called the **iterated posterior linearization smoother** or **IPLS**, which extends the IPLF method of Section 8.9.3 to the offline setting. The basic idea is to compute expectations with respect to the previous smoothed posterior, and to iterate this process until convergence. See Algorithm 16 for the pseudocode.

The **iterated EKS** is similar to the IPLS, except it uses a first order Taylor series approximation of the measurement and dynamics functions at the posterior mean from the previous iteration, instead of using SLR. In [Bel94], it is shown to be equivalent to a Gauss-Newton method for computing the MAP estimate of the smoothing posterior.

Unfortunately both IEKS and IPLS can diverge. A robust IEKS method, that uses line search and Levenberg-Marquadt to update the parameters, is presented in [SS20a]. In [Lin+21c], a similar

Algorithm 16: Iterated posterior linearization smoother

def **IPLS**$(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{h}, \mathbf{R}, \boldsymbol{y}_{1:T}, J)$:
  $(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T})_{t=1}^T = \text{UnscentedSmoother}(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{h}, \mathbf{R}, \boldsymbol{y}_{1:T})$
  foreach $j = 1 : J$ **do**
    foreach $t = 1 : T$ **do**
      $(\boldsymbol{m}, \mathbf{S}, \mathbf{C}) = \textbf{UnscentedPredict}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}, \boldsymbol{f}(\cdot, \boldsymbol{u}_t), \mathbf{0})$
      $(\mathbf{F}_t, \boldsymbol{b}_t, \tilde{\mathbf{Q}}_t) = \textbf{SLR}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}, \boldsymbol{m}, \mathbf{S}, \mathbf{C})$
      $(\boldsymbol{m}, \mathbf{S}, \mathbf{C}) = \textbf{UnscentedPredict}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}, \boldsymbol{h}(\cdot, \boldsymbol{u}_t), \mathbf{0})$
      $(\mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t) = \textbf{SLR}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}, \boldsymbol{m}, \mathbf{S}, \mathbf{C})$
    $(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T})_{t=1}^T = \text{KalmanSmoother}(\{\mathbf{F}_t, \boldsymbol{b}_t, \mathbf{Q} + \tilde{\mathbf{Q}}_t, \mathbf{H}_t, \boldsymbol{d}_t, \mathbf{R} + \tilde{\mathbf{R}}_t\}, \boldsymbol{y}_{1:T})$
  Return $(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T})_{t=1}^T$

robust method is derived for IPLS.

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 general graphical models with Gaussian potentials but nonlinear dependencies.

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" distributions, which is the current posterior minus the incoming message that was sent at the last iteration, similar to Section 8.3.4.4. The advantage of this is that the outgoing message does not double count the evidence. The disadvantage is that this may be numerically unstable.

8.9.5 Beyond additive Gaussian noise

In this section, we extend SLR to handle the case of non-Gaussian / non-additive noise, following [TGFS18].

8.9.5.1 Generalized statistical linear regression

We assume $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X)$. We allow $p(\boldsymbol{y}|\boldsymbol{x})$ to be any kind of distribution, provided it has finite variance. We only require that we can approximate the first and second **conditional moments**:

$$\boldsymbol{m}_Y(\boldsymbol{x}) = \mathbb{E}\left[\boldsymbol{y}|\boldsymbol{x}\right], \mathbf{V}_Y(\boldsymbol{x}) = \text{Cov}\left[\boldsymbol{y}|\boldsymbol{x}\right] \tag{8.217}$$

For example, if $p(y|x) = \text{Poisson}(y|ce^x)$, we have

$$m_Y(x) = V_Y(x) = ce^x \tag{8.218}$$

As usual we will approximate the conditional distribution by a linear Gaussian model of the form $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\mathbf{A}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{\Omega})$. We can then easily compute the joint $p(\boldsymbol{x}, \boldsymbol{y})$, the marginal $p(\boldsymbol{y})$, and the posterior $p(\boldsymbol{x}|\boldsymbol{y})$.

The optimal parameters, in the sense of minimizing the mean squared norm of the residual, $||\boldsymbol{y} - \mathbf{A}\boldsymbol{x} - \boldsymbol{b}||$, and the corresponding error matrix, are given by the SLR equations in Algorithm 14, where the moments are computed using

$$\boldsymbol{m} = \mathbb{E}[\mathbf{Y}] = \mathbb{E}[\mathbb{E}[\mathbf{Y}|\mathbf{X}]] = \int_{\boldsymbol{x}} p(\boldsymbol{x})\boldsymbol{m}_Y(\boldsymbol{x}) \tag{8.219}$$

$$\mathbf{S} = \mathbb{V}[\mathbf{Y}] = \mathbb{E}[\mathbb{V}[\mathbf{Y}|\boldsymbol{x}]] + \mathbb{V}[\mathbb{E}[\mathbf{Y}|\mathbf{X}]] \tag{8.220}$$

$$= \int_{\boldsymbol{x}} p(\boldsymbol{x})\mathbf{V}_Y(\boldsymbol{x}) + \int_{\boldsymbol{x}} p(\boldsymbol{x})(\boldsymbol{m}_Y(\boldsymbol{x}) - \boldsymbol{m})(\boldsymbol{m}_Y(\boldsymbol{x}) - \boldsymbol{m})^\mathsf{T} \tag{8.221}$$

$$\mathbf{C} = \mathrm{Cov}[\mathbf{X}, \mathbf{Y}] = \mathrm{Cov}[\mathbf{X}, \mathbb{E}[\mathbf{Y}|\mathbf{X}]] = \int_{\boldsymbol{x}} p(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{\mu}_X)(\boldsymbol{m}_Y(\boldsymbol{x}) - \boldsymbol{m}) \tag{8.222}$$

where expectations are taken wrt $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X)$. This is called **generalized statistical linear regression**.

In general, we cannot compute these expectations exactly, so we need to use numerical integration (e.g., sigma point methods). Alternatively, we can make a Taylor series approximation to the conditional moments, and then we can compute the unconditional moments analytically. In more detail, we first compute

$$\boldsymbol{m}_Y(\boldsymbol{x}) \approx \underbrace{\boldsymbol{m}_Y(\boldsymbol{\mu})}_{\boldsymbol{\mu}_Y} + \underbrace{\nabla \boldsymbol{m}_Y(\boldsymbol{\mu})}_{\mathbf{J}}(\boldsymbol{x} - \boldsymbol{\mu}) \tag{8.223}$$

$$\mathbf{V}_Y(\boldsymbol{x}) \approx \mathbf{V}_Y(\boldsymbol{\mu}) \tag{8.224}$$

We can then compute the unconditional moments using Algorithm 17.

Algorithm 17: Linearized Gaussian approximation to a conditional distribution.

```
def CondMomentsPredict(μ_X, Σ_X, m_Y(), V_Y()):
  J = Jac(m_Y)(μ_X)
  m = m_Y(μ_X)
  S = V_Y(μ_X) + J Σ_X J^T
  C = Σ_X J^T
  Return (m, S, C)
```

8.9.5.2 Generalized IPLF

Once we have computed the moments, we can use SLR to compute a linear Gaussian approximation to $p(\boldsymbol{y}|\boldsymbol{x})$ and hence we can compute $p(\boldsymbol{x}, \boldsymbol{y})$, from which we can extract $p(\boldsymbol{y})$ for the predict step and $p(\boldsymbol{x}|\boldsymbol{y})$ for the update step.

As in the IPLF method of Section 8.9.3, we can get improved accuracy by iterating the computation of the moments, using the most recent posterior estimate for the expectation. We call this the **generalized IPLF** or **GIPLF**. We give the pseudocode in Algorithm 18, where we assume (for notational simplicity) that the dynamics model has additive Gaussian noise, and where we use Taylor series linearization rather than sigma point methods. If the observation model is linear Gaussian, the resulting algorithm is equivalent to the IEKF in Section 8.5.3.

Algorithm 18: Generalized IPLF using Taylor series linearization.

def **GIPLF**$(\boldsymbol{f}, \mathbf{Q}, \boldsymbol{m}_Y(), \mathbf{V}_Y(), \boldsymbol{y}_{1:T}, J)$:
  foreach $t = 1 : T$ **do**
    Predict step:
    $(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}, -) = \textbf{LinearizedPredict}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1}, \boldsymbol{f}, \mathbf{Q}_t)$
    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**
      $(\boldsymbol{m}, \mathbf{S}, \mathbf{C}) = \textbf{CondMomentsPredict}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{m}_Y, \mathbf{V}_Y)$
      $(\mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t) = \textbf{SLR}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{m}, \mathbf{S}, \mathbf{C})$
      $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \ell_t) = \textbf{LinGaussUpdate}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \mathbf{H}_t, \boldsymbol{d}_t, \tilde{\mathbf{R}}_t, \boldsymbol{y}_t)$
  Return $(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})_{t=1}^T$

8.9.5.3 Extensions

The generalized IPLF algorithm can be extended in various ways. In [Hos+20b] they use it as a proposal distribution inside of a particle filtering algorithm (Section 13.2). In [TGFS18], they extend it to the smoothing (offline) setting. In [GFTS19], they apply it to fitting a Gaussian process with a Bernoulli likelihood.

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.

8.10 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**

$$p_{t|t-1}(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) q_{t-1}(\boldsymbol{z}_{t-1}) d\boldsymbol{z}_{t-1} \tag{8.225}$$

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

$$p_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \frac{1}{Z_t} p(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{t|t-1}(\boldsymbol{z}_t) \tag{8.226}$$

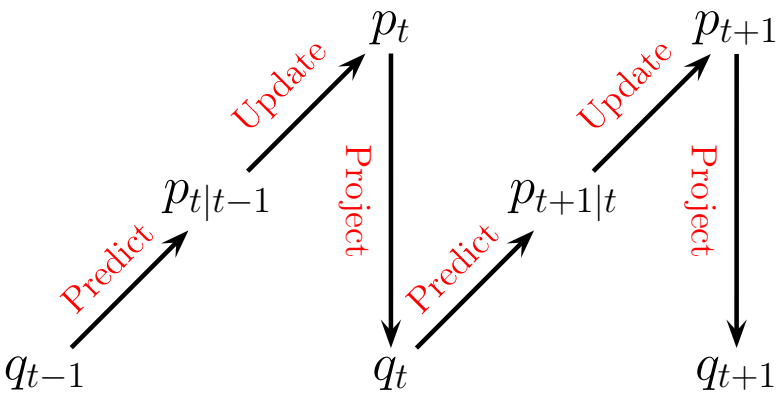


Figure 8.13: Illustration of the predict-update-project cycle of assumed density filtering. $q_t \in \mathcal{Q}$ is a tractable distribution, whereas we may have $p_{t|t-1} \notin \mathcal{Q}$ and $p_t \notin \mathcal{Q}$.

where

$$Z_t = \int p(\boldsymbol{y}_t|\boldsymbol{z}_t) p_{t|t-1}(\boldsymbol{z}_t) d\boldsymbol{z}_t \tag{8.227}$$

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

$$q_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathbb{KL}}\left(p_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) \parallel q(\boldsymbol{z}_t)\right) \tag{8.228}$$

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 8.13.

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 5.1.3.4). 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 5.1.3.3), rather than trying to capture overall moments.

8.10.1 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 8.8. GF corresponds to solving the following optimization problem

$$q_{t|t-1}(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t) = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathbb{KL}}\left(p(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t|\boldsymbol{y}_{1:t-1}) \parallel q(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t|\boldsymbol{y}_{1:t-1})\right) \tag{8.229}$$

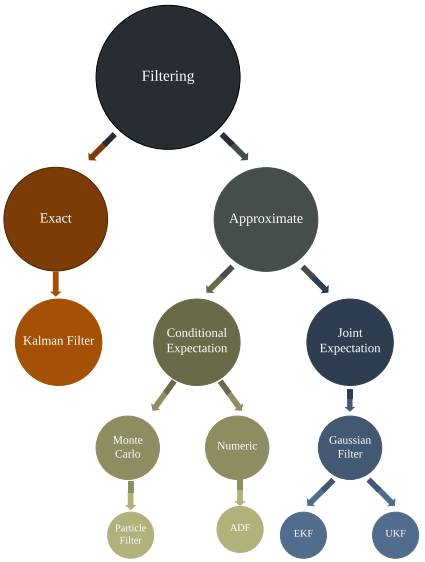


Figure 8.14: A taxonomy of filtering algorithms. Adapted from Figure 2 of [Wüt+16].

which can be solved by moment matching (see Section 8.8). We then conditioning 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 $q_{t|t-1}(\boldsymbol{z}_t, \tilde{\boldsymbol{y}}_t|\boldsymbol{y}_{1:t-1})$, from which we derive $p_t(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$ by conditioning.

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 8.14 for a summary of how these different methods relate.

8.10.2 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:

$$b_{t-1}^i \triangleq p(\boldsymbol{z}_{t-1}, m_{t-1} = i|\boldsymbol{y}_{1:t-1}) = \pi_{t-1|t-1}^i \mathcal{N}(\boldsymbol{z}_{t-1}|\boldsymbol{\mu}_{t-1|t-1}^i, \boldsymbol{\Sigma}_{t-1|t-1}^i) \tag{8.230}$$

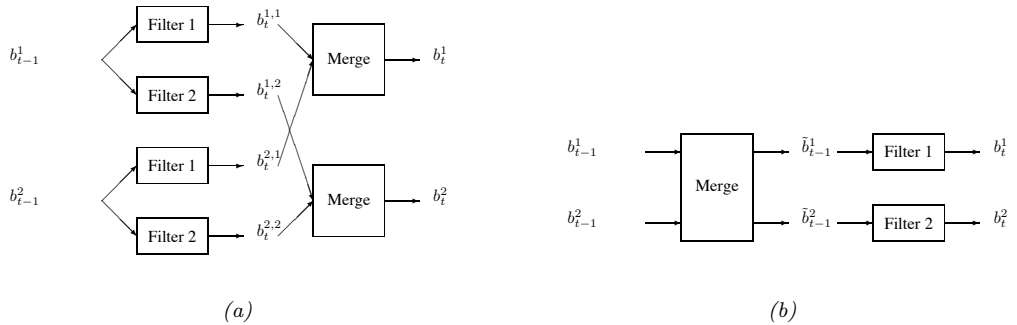


Figure 8.15: ADF for a switching linear dynamical system with 2 discrete states. (a) GPB2 method. (b) IMM method.

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

$$b_t^{ij} \triangleq p(\boldsymbol{z}_t, m_{t-1} = i, m_t = j | \boldsymbol{y}_{1:t}) = \pi_{t|t}^{ij} \mathcal{N}(\boldsymbol{z}_t | \boldsymbol{\mu}_{t|t}^{ij}, \boldsymbol{\Sigma}_{t|t}^{ij}) \tag{8.231}$$

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

$$b_t^{j} \triangleq p(\boldsymbol{z}_t, m_t = j | \boldsymbol{y}_{1:t}) = \pi_{t|t}^{j} \mathcal{N}(\boldsymbol{z}_t | \boldsymbol{\mu}_{t|t}^{j}, \boldsymbol{\Sigma}_{t|t}^{j}) \tag{8.232}$$

See Figure 8.15a 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,

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{z}] = \sum_k \pi^k \boldsymbol{\mu}^k \tag{8.233}$$

$$\boldsymbol{\Sigma} = \mathrm{Cov}[\boldsymbol{z}] = \sum_k \pi^k \left(\boldsymbol{\Sigma}^k + (\boldsymbol{\mu}^k - \boldsymbol{\mu})(\boldsymbol{\mu}^k - \boldsymbol{\mu})^\mathsf{T} \right) \tag{8.234}$$

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):

$$\pi^j = \sum_i \pi^{ij} \tag{8.235}$$

$$\pi^{j|i} = \frac{\pi^{ij}}{\sum_{j'} \pi^{ij'}} \tag{8.236}$$

$$\boldsymbol{\mu}^j = \sum_i \pi^{j|i} \boldsymbol{\mu}^{ij} \tag{8.237}$$

$$\boldsymbol{\Sigma}^j = \sum_i \pi^{j|i} \left(\boldsymbol{\Sigma}^{ij} + (\boldsymbol{\mu}^{ij} - \boldsymbol{\mu}^j)(\boldsymbol{\mu}^{ij} - \boldsymbol{\mu}_j)^\mathsf{T} \right) \tag{8.238}$$

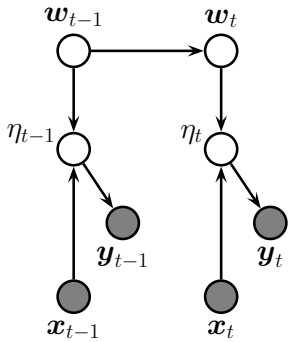


Figure 8.16: A dynamic logistic regression model. $\boldsymbol{w}_t$ are the regression weights at time t, and $\eta_t = \boldsymbol{w}_t^\mathsf{T}\boldsymbol{x}_t$. Compare to Figure 29.24a.

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 8.15b for a sketch.

8.10.3 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:

$$p(y_t|\boldsymbol{x}_t, \boldsymbol{w}_t) = \text{Ber}(y_t|\sigma(\boldsymbol{x}_t^\mathsf{T}\boldsymbol{w}_t)) \tag{8.239}$$

$$p(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}) = \mathcal{N}(\boldsymbol{w}_t|\boldsymbol{w}_{t-1}, \mathbf{Q}) \tag{8.240}$$

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 8.16 for an illustration of the model.

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

$$p(\boldsymbol{w}_{t-1}|\mathcal{D}_{1:t-1}) \approx p_{t-1}(\boldsymbol{w}_{t-1}) = \prod_j \mathcal{N}(w_{t-1}^j|\mu_{t-1|t-1}^j, \tau_{t-1|t-1}^j) \tag{8.241}$$

where $\mu_{t-1|t-1}^j$ and $\tau_{t-1|t-1}^j$ 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 $p_{t|t-1}(\boldsymbol{w}_t) = \prod_j \mathcal{N}(w_t^j | \mu_{t|t-1}^j, \tau_{t|t-1}^j)$, then we can compute the 1d prior predictive distribution for η_t as follows:

$$p(\eta_t | \mathcal{D}_{1:t-1}, \boldsymbol{x}_t) \approx p_{t|t-1}(\eta_t) = \mathcal{N}(\eta_t | m_{t|t-1}, v_{t|t-1}) \tag{8.242}$$

$$m_{t|t-1} = \sum_j x_{t,j} \mu_{t|t-1}^j \tag{8.243}$$

$$v_{t|t-1} = \sum_j x_{t,j}^2 \tau_{t|t-1}^j \tag{8.244}$$

The posterior for the 1d η_t is given by

$$p(\eta_t | \mathcal{D}_{1:t}) \approx p_t(\eta_t) = \mathcal{N}(\eta_t | m_t, v_t) \tag{8.245}$$

$$m_t = \int \eta_t \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \tag{8.246}$$

$$v_t = \int \eta_t^2 \frac{1}{Z_t} p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t - m_t^2 \tag{8.247}$$

$$Z_t = \int p(y_t | \eta_t) p_{t|t-1}(\eta_t) d\eta_t \tag{8.248}$$

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].

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:

$$m_t = m_{t|t-1} + \delta_m, \;\; v_t = v_{t|t-1} + \delta_v \tag{8.249}$$

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:

$$p_t(w_t^i) = \mathcal{N}(w_t^i | \mu_{t|t}^i, \tau_{t|t}^i) \tag{8.250}$$

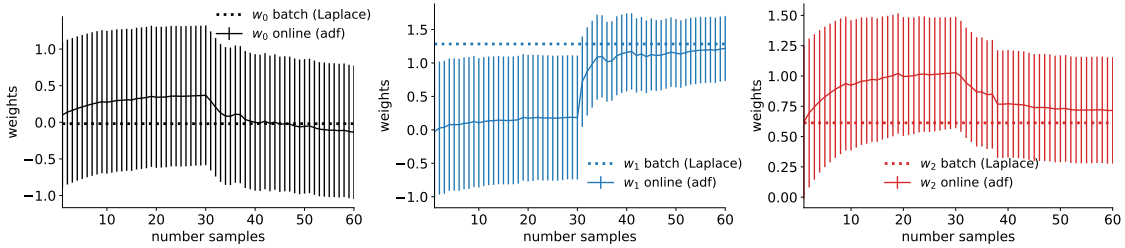
$$\mu_{t|t}^i = \mu_{t|t-1}^i + a_i \delta_m \tag{8.251}$$

$$\tau_{t|t}^i = \tau_{t|t-1}^i + a_i^2 \delta_v \tag{8.252}$$

$$a_i \triangleq \frac{x_t^i \tau_{t|t-1}^i}{\sum_j (x_t^j)^2 + \tau_{t|t-1}^j} \tag{8.253}$$

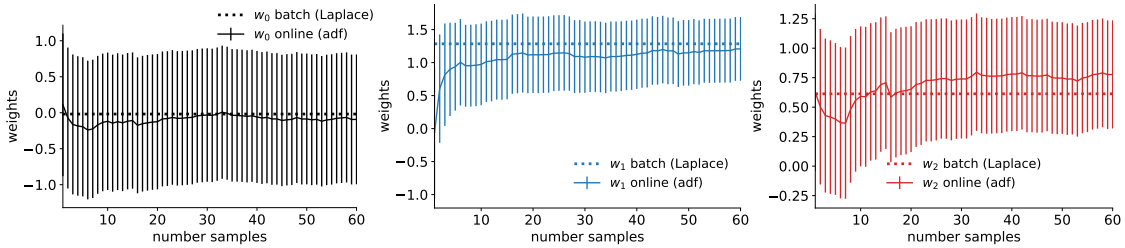
Thus we see that the parameters which correspond to inputs i with larger magnitude (big $|x_t^i|$) or larger uncertainty (big $\tau_{t|t-1}^i$) get updated most, due to a large a_i factor, which makes intuitive sense.

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.



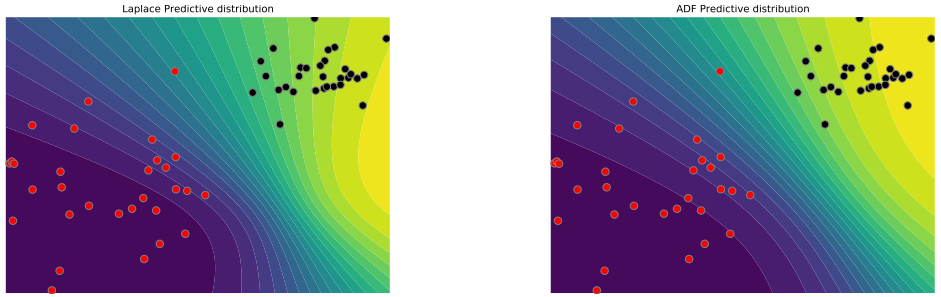
(a) (b) (c)

Figure 8.17: Bayesian inference applied to a 2d binary logistic regression problem, $p(y = 1|\boldsymbol{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$. We show the training data and the posterior predictive produced by different methods. (a) Offline MCMC approximation. (b) Offline Laplace approximation. (c) Online ADF approximation at the final step of inference. Generated by adf_logistic_regression_demo.ipynb.



(a) (b) (c)

Figure 8.18: Same as Figure 8.17, except the order in which the data is visited is different. Generated by adf_logistic_regression_demo.ipynb.



(a) (b)

Figure 8.19: 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.

In Figure 8.17 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 8.18, 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 8.19, 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.

The overall approach is very similar to the generalized posterior linearization filter of Section 8.9.5.2, 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 8.10.1.

8.10.4 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 8.10.3 to the case of nonlinear models.

8.11 Other inference methods for SSMs

There are a variety of other inference algorithms that can be applied to SSMs which we discuss elsewhere in this book. We give a very brief summary below, mostly focused on the case of offline inference (smoothing).

8.11.1 Expectation propagation

In Section 10.7 we discuss the expectation propagation (EP) algorithm, which can be viewed as an iterative version of ADF (Section 8.10). 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 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 8.8), as explained in [Kam+22].

8.11.2 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 5.1.3.3, 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].

8.11.3 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.

8.11.4 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 neglible 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 Inference for graphical models

9.1 Introduction

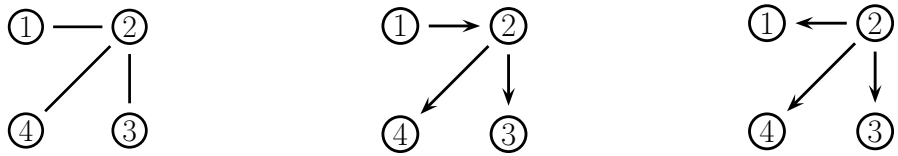
In this chapter we consider posterior inference (i.e., computing marginals, modes, samples, etc) for probability distributions that can be represented by a PGM with some kind of sparse graph structure (i.e., it is not a fully connected graph). The algorithms we discuss will leverage the conditional independence 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.

The methods we discuss generalize the algorithms from Chapter 8, which only work with chain structured graphical models, to work with PGMs with arbitrary graph structure. This requires that we specify a **message passing schedule**. For chain structured models, a natural approach is to send messages forwards in time, and then backwards in time, as we discussed in Chapter 8. We can generalize this approach to work with trees, as we discuss in Section 9.2. For general graphs, there may be cycles or loops, as we discuss in Section 9.3. 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 9.4 and Section 9.5. We can also pose the inference problem as an optimization problem, as we discuss in Section 9.6.

9.2 Belief propagation on trees

The forwards-backwards algorithm for HMMs (Section 8.2.4) and the Kalman smoother algorithm for LDS (Section 8.3.3) can both be interpreted as message passing algorithms applied to chain structured graphical models. In this section, we generalize these algorithms to work with trees.



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

Figure 9.1: An undirected tree and two equivalent directed trees.

9.2.1 Directed vs undirected trees

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

$$p^*(\boldsymbol{z}) \triangleq p(\boldsymbol{z}|\boldsymbol{y}) \propto \prod_{s\in\mathcal{V}} \psi_s(z_s|\boldsymbol{y}_s) \prod_{(s,t)\in\mathcal{E}} \psi_{s,t}(z_s, z_t) \tag{9.1}$$

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 9.1a. We can always convert this into a directed tree by picking an arbitrary node as the root, and then "picking 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 9.1b. This corresponds to the following directed graphical model:

$$p^*(\boldsymbol{z}) \propto p^*(z_1)p^*(z_2|z_1)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.2}$$

However, if we pick node 2 as the root, we get Figure 9.1c. This corresponds to the following directed graphical model:

$$p^*(\boldsymbol{z}) \propto p^*(z_2)p^*(z_1|z_2)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.3}$$

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

$$p^*(\boldsymbol{z}) \propto \prod_{s\in\mathcal{V}} p^*(z_s) \prod_{(s,t)\in\mathcal{E}} \frac{p^*(z_s, z_t)}{p^*(z_s)p^*(z_t)} \tag{9.4}$$

For example, for Figure 9.1a we have

$$p^*(z_1, z_2, z_3, z_4) \propto p^*(z_1)p^*(z_2)p^*(z_3)p^*(z_4)\frac{p^*(z_1, z_2)p^*(z_2, z_3)p^*(z_2, z_4)}{p^*(z_1)p^*(z_2)p^*(z_2)p^*(z_3)p^*(z_2)p^*(z_4)} \tag{9.5}$$

```
// Collect to root
for each node s in post-order
    bel_s(z_s) ∝ ψ_s(z_s) ∏_{t∈ch_s} m_{t→s}(z_s)
    t = parent(s)
    m_{s→t}(z_t) = Σ_{z_s} ψ_st(z_s, z_t) bel_s(z_s)

// Distribute from root
for each node t in pre-order
    s = parent(t)
    m_{s→t}(z_t) = bel_s(z_s) / m_{t→s}(z_s)
    bel_t(z_t) ∝ bel_t(z_t) m_{s→t}(z_t)
```

Figure 9.2: Belief propagation on a pairwise, rooted tree.

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

$$p^*(z_1, z_2, z_3, z_4) \propto p^*(z_1, z_2)\frac{p^*(z_2, z_3)}{p^*(z_2)}\frac{p^*(z_2, z_4)}{p^*(z_2)} \tag{9.6}$$

$$= p^*(z_1)p^*(z_2|z_1)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.7}$$

$$= p^*(z_2)p^*(z_1|z_2)p^*(z_3|z_2)p^*(z_4|z_2) \tag{9.8}$$

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.

9.2.2 Sum-product algorithm

In this section, we assume that our model is an undirected tree, as in Equation (9.1). 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 8.2.2. 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:

$$\text{bel}_s(z_s) \propto \psi_s(z_s) \prod_{t\in \text{ch}_s} m_{t\to s}(z_s) \tag{9.9}$$

To compute the outgoing message that s should send to its parent t, we pass the local belief through

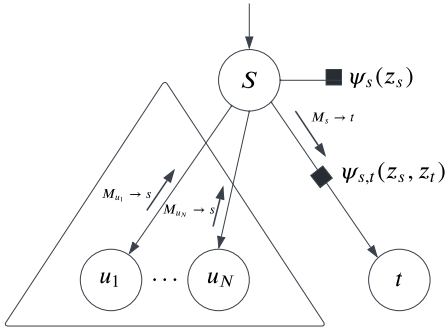


Figure 9.3: Illustration of how the top-down message from s to t is computed during BP on a tree. The u_i nodes are the other children of s, besides t. Square nodes represent clique potentials.

the pairwise potential linking s and t, and then marginalize out s to get

$$m_{s\to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t)\text{bel}_s(z_s) \tag{9.10}$$

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:

$$m_{s\to t}(z_t) = \sum_{z_t} \left(\psi_s(z_s)\psi_{st}(z_s, z_t) \prod_{u \in \text{ch}_s \setminus t} m_{u\to s}(z_s) \right) \tag{9.11}$$

See Figure 9.3. 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 (9.9), so we don't need to recompute that term. Thus we get

$$m_{s\to t}(z_t) = \sum_{z_s} \psi_{st}(z_s, z_t) \frac{\text{bel}_s(z_s)}{m_{t\to s}(z_s)} \tag{9.12}$$

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:

$$\text{bel}_t(z_t) \propto \text{bel}_t(z_t) m_{s\to t}(z_t) \tag{9.13}$$

(Note that Equation (9.9) 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 (8.24), 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 9.2 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.

9.2.3 Max-product algorithm

In Section 9.2.2 we described the sum-product algorithm, that computes the posterior marginals:

$$\text{bel}_i(k) = \gamma_i(k) = p(z_i = k|\boldsymbol{y}) = \sum_{\boldsymbol{z}_{-i}} p(z_i = k, \boldsymbol{z}_{-i}|\boldsymbol{y}) \tag{9.14}$$

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:

$$\zeta_i(k) = \max_{\boldsymbol{z}_{-i}} p(z_i = k, \boldsymbol{z}_{-i}|\boldsymbol{y}) \tag{9.15}$$

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.

9.2.3.1 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 8.2.7), 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 get reduced at each round.

9.2.3.2 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:[1] $\gamma_1 = [0.3469, 0.6531]$, $\gamma_2 = [0.6531, 0.3469]$, $\gamma_3 = [0.6531, 0.3469]$. The exact max marginals are all the same, 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 34.1). 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 8.4, 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.

9.2.3.3 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

$$\boldsymbol{z}_Q^* = \arg\max_{\boldsymbol{z}_Q} \sum_{\boldsymbol{z}_R} p(\boldsymbol{z}_Q, \boldsymbol{z}_R|\boldsymbol{z}_V) \tag{9.16}$$

(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).

1. See error_correcting_code_demo.ipynb for the code.

9.3 Loopy belief propagation

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

> When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the network — [Pea88, p.195]

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].)

9.3.1 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 (9.1). However, unlike Section 9.2.2, 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

$$\text{bel}_s(z_s) \propto \psi_s(z_s) \prod_{t \in \text{nbr}_s} m_{t \to s}(z_s) \tag{9.17}$$

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

$$m_{s \to t}(z_t) = \sum_{z_s} \left(\psi_s(z_s) \psi_{st}(z_s, z_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \to s}(z_s) \right) \tag{9.18}$$

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.

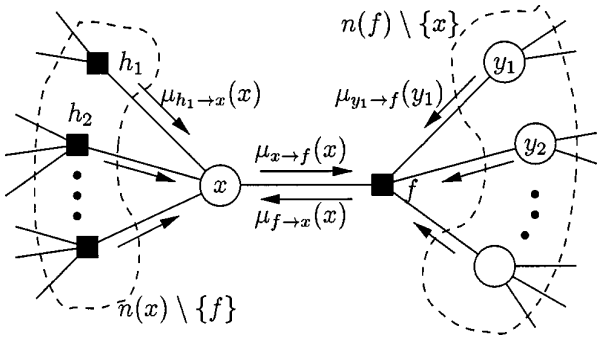


Figure 9.4: Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent variables. The y_i nodes correspond to the neighbors x'_i of f other than x. From Figure 6 of [KFL01]. Used with kind permission of Brendan Frey.

9.3.2 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].[2] For a version that works for Forney factor graphs, see [Loe+07].

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

$$m_{x \to f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \to x}(x) \tag{9.19}$$

and factors to variables

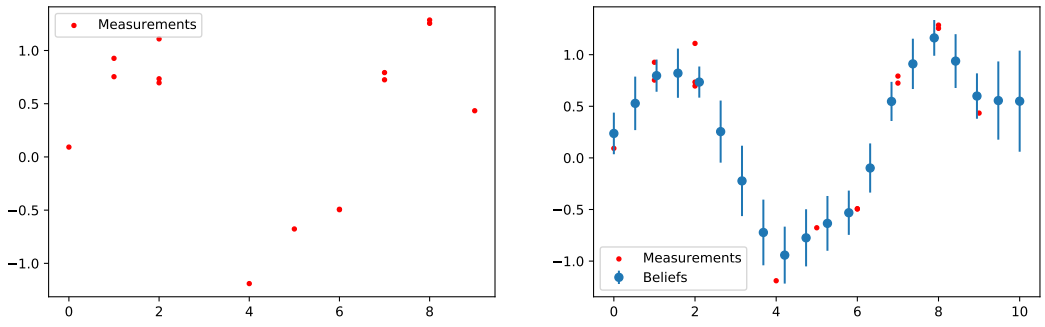
$$m_{f \to x}(x) = \sum_{\boldsymbol{x}'} f(x, \boldsymbol{x}') \prod_{x' \in \text{nbr}(f) \setminus \{x\}} m_{x' \to f}(x') \tag{9.20}$$

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 9.4. At convergence, we can compute the final beliefs as a product of incoming messages:

$$\text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \to x}(x) \tag{9.21}$$

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.

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



(a) (b)

Figure 9.5: Interpolating noisy data using Gaussian belief propagation applied to a 1d MRF. Generated by gauss-bp-1d-line.ipynb.

9.3.3 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 2.2.7 to compute the messages and beliefs. Note that computing the posterior mean in a linear-Gaussian system is equivalent to solving a linear system, so these methods are also useful for linear algebra. See e.g., [PL03; Bic09; Du+18] for details.

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:

$$\psi_i(z_{i-1}, z_i) = \frac{1}{\sigma^2}(\hat{y}_i - y_i)^2 \tag{9.22}$$

$$\hat{y}_i = (1 - \gamma_i)z_{i-1} + \gamma_i z_i \tag{9.23}$$

$$\gamma_i = \frac{x_i - g_i}{g_i - g_{i-1}} \tag{9.24}$$

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}:

$$\phi_i(z_{i-1}, z_i) = \frac{1}{\tau^2}\delta_i^2 \tag{9.25}$$

$$\delta_i = z_i - z_{i-1} \tag{9.26}$$

The overall model is

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g}, \sigma^2, \tau^2) \propto \prod_{i=1}^{G} \psi_i(z_{i-1}, z_i)\phi_i(z_{i-1}, z_i) \tag{9.27}$$

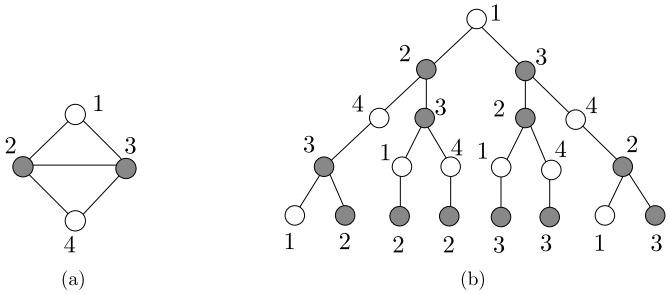


Figure 9.6: (a) A simple loopy graph. (b) The computation tree, rooted at node 1, after 4 rounds of message passing. Nodes 2 and 3 occur more often in the tree because they have higher degree than nodes 1 and 2. From Figure 8.2 of [WJ08]. Used with kind permission of Martin Wainwright.

Suppose the true underlying function is a sine wave. We show some sample data in Figure 9.5(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 9.5(b), are exact. We see that the method has inferred the underlying sine shape just based on a smoothness prior.

To perform message passing in models with non-Gaussian potentials, we can extend the techniques from Section 8.5.2 from chains to general graphs. For example, we can use local linearization, similar to extended Kalman filter (see [PHR18]); or we can use **sigma point BP** [MHH14], similar to unscented Kalman filter.

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).

9.3.4 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.

9.3.4.1 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 9.6 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.

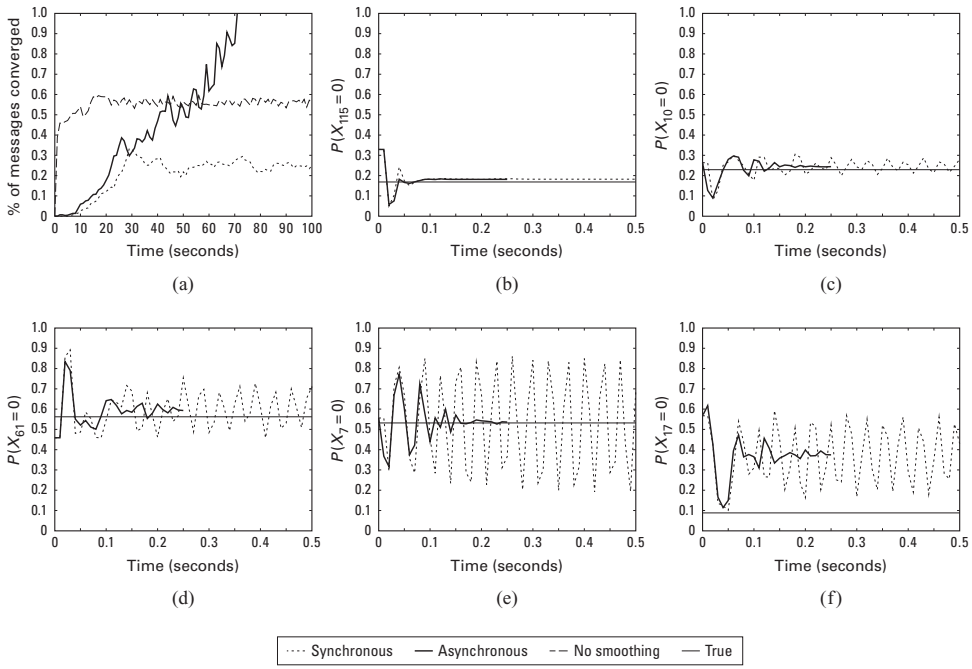


Figure 9.7: Illustration of the behavior of loopy belief propagation on an 11×11 *Ising grid with random potentials,* $w_{ij} \sim \text{Unif}(-C, C)$*, where* $C = 11$*. For larger* C*, inference becomes harder. (a) Percentage of messages that have converged vs time for 3 different update schedules: 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 asynchronous. From Figure 11.C.1 of [KF09a]. Used with kind permission of Daphne Koller.*

9.3.4.2 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, we use an update of the form

$$m^k_{t\to s}(x_s) = \lambda m_{t\to s}(x_s) + (1-\lambda) m^{k-1}_{t\to s}(x_s) \tag{9.28}$$

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 9.7, where we see that damped updating results in convergence much more often than undamped updating (see [ZLG20] for some analysis of the benefits of damping).

It is possible to devise methods, known as **double loop algorithms**, which are guaranteed to converge to a local minimum of the same objective that LBP is minimizing [Yui01; WT01].

Unfortunately, these methods are rather slow and complicated, and the accuracy of the resulting marginals is usually not much greater than with standard LBP. (Indeed, oscillating marginals is sometimes a sign that the LBP approximation itself is a poor one.) Consequently, these techniques are not very widely used.

9.3.4.3 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

$$\boldsymbol{m}_{1:E}^{k+1} = (f_1(\boldsymbol{m}^k), \ldots, f_E(\boldsymbol{m}^k)) \tag{9.29}$$

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 [Ber97] that the Gauss-Seidel method, which performs **asynchronous updates** in a fixed round-robin fashion, converges faster when solving linear systems of equations. We can apply the same idea to LBP, using updates of the form

$$\boldsymbol{m}_i^{k+1} = f_i\left(\{\boldsymbol{m}_j^{k+1} : j < i\}, \{\boldsymbol{m}_j^{k} : j > i\}\right) \tag{9.30}$$

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 9.7, 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

$$r(s,t,k) = ||\log m_{s\to t} - \log m_{s\to t}^k||_\infty = \max_j |\log \frac{m_{s\to t}(j)}{m_{s\to t}^k(j)}| \tag{9.31}$$

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, 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 is similar.

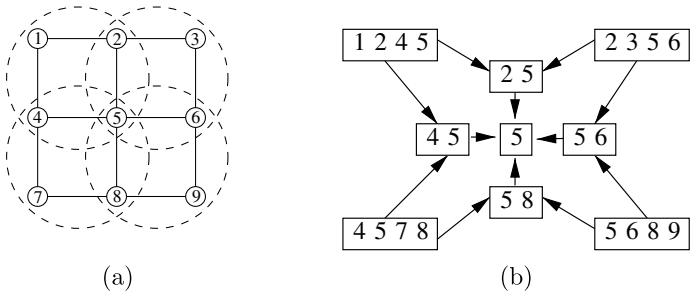


Figure 9.8: (a) Clusters superimposed on a 3×3 lattice graph. (b) Corresponding hyper-graph. Nodes represent clusters, and edges represent set containment. From Figure 4.5 of [WJ08]. Used with kind permission of Martin Wainwright.

9.3.5 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].

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.

9.3.6 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 9.5) 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 9.8 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.

9.3.7 Convex BP

In Supplementary Section 10.4.3 we analyse 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.

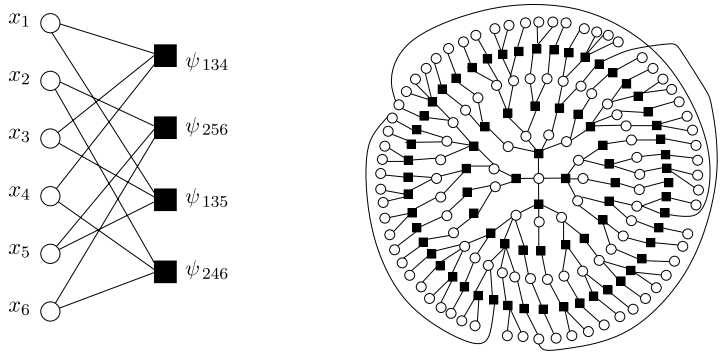


Figure 9.9: (a) A simple factor graph representation of a (2,3) low-density parity check code. Each message bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity factor has the form $\psi_{stu}(x_s, x_t, x_u) = \mathbb{I}(x_s \otimes x_t \otimes x_u = 1)$, 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.

9.3.8 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.

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 9.9(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

$$\psi_{stu}(x_s, x_t, x_u) \triangleq \begin{cases} 1 & \text{if } x_s \otimes x_t \otimes x_u = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.32}$$

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

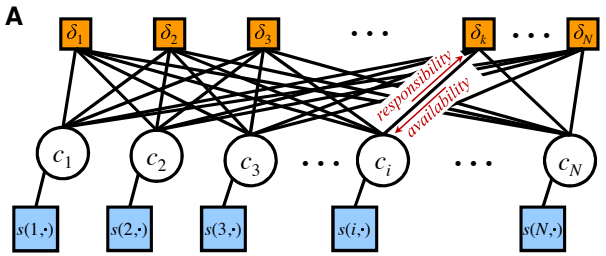


Figure 9.10: Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of [FD07a]. Used with kind permission of Brendan Frey.

a similar way.)

Figure 9.9(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 methods such as these, people have been able to approach the lower bound in Shannon's channel coding theorem, meaning they have produced codes with very little redundancy for a given amount of noise in the channel. See e.g., [MMC98; Mac03] for more details. Such codes are widely used, e.g., in modern cellphones.

9.3.9 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 data point must choose another data point as its exemplar or centroid; some data points will choose themselves as centroids, and this will automatically determine the number of clusters. More precisely, let $c_i \in \{1, \ldots, N\}$ represent the centroid for datapoint i. The goal is to maximize the following function

$$J(\boldsymbol{c}) = \sum_{i=1}^{N} S(i, c_i) + \sum_{k=1}^{N} \delta_k(\boldsymbol{c}) \tag{9.33}$$

where $S(i, c_i)$ is the similarity between data point i and its centroid c_i. The second term is a penalty term that is $-\infty$ if some data point i has chosen k as its exemplar (i.e., $c_i = k$), but k has not chosen itself as an exemplar (i.e., we do not have $c_k = k$). More formally,

$$\delta_k(\boldsymbol{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 & \text{otherwise} \end{cases} \tag{9.34}$$

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 9.10, or we can use N^2 binary nodes (see [GF09] for the details). We will assume the former representation.

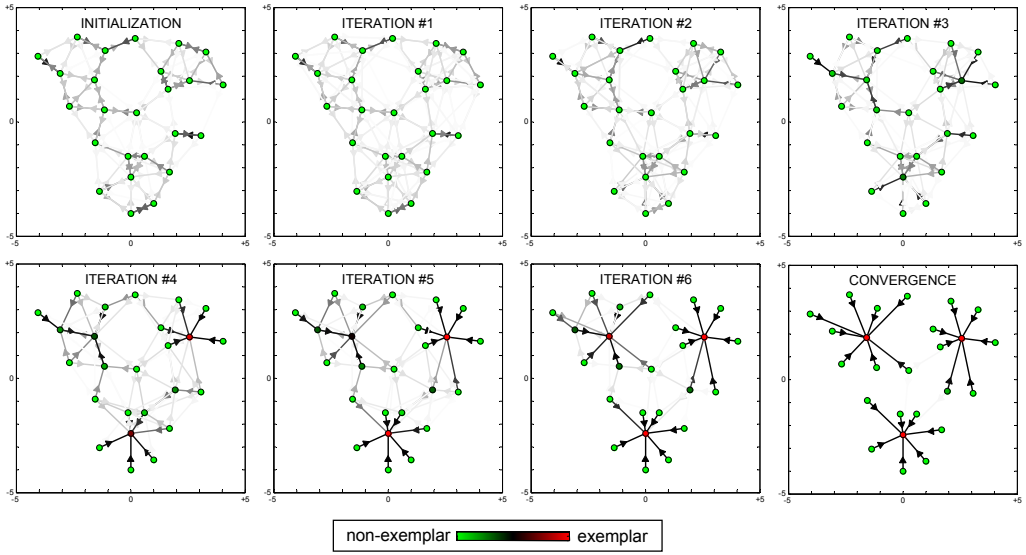


Figure 9.11: Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \to k$ arrow reflects how much point i wants to belong to exemplar k. From Figure 1 of [FD07a]. Used with kind permission of Brendan Frey.

We can find a strong local maximum of the objective by using max-product loopy belief propagation (Section 9.3). Referring to the model in Figure 9.10, 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, 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 data points k has looked at.

As usual with loopy BP, the method might oscillate, and convergence is not guaranteed. However, by using damping, the method is very reliable in practice. If the graph is densely connected, message passing takes $O(N^2)$ time, but with sparse similarity matrices, it only takes $O(E)$ time, where E is the number of edges or non-zero entries in S.

The number of clusters can be controlled by scaling the diagonal terms $S(i,i)$, which reflect how much each data point wants to be an exemplar. Figure 9.11 gives a simple example of some 2d data, where the negative Euclidean distance was used to measured similarity. The $S(i,i)$ values were set to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are reported in [FD07a], who show that the method significantly outperforms K-medoids.

9.3.10 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 10.3.6 for more discussion).

9.4 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 2.2.7.

9.4.1 Derivation of the algorithm

We will explain the algorithm by applying it to an example. Specifically, we consider the student network from Section 4.2.2.2. Suppose we want to compute $p(J = 1)$, the marginal probability that a person will get a job. Since we have 8 binary variables, we could simply enumerate over all possible assignments to all the variables (except for J), adding up the probability of each joint instantiation:

$$p(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C p(C, D, I, G, S, L, J, H) \tag{9.35}$$

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].

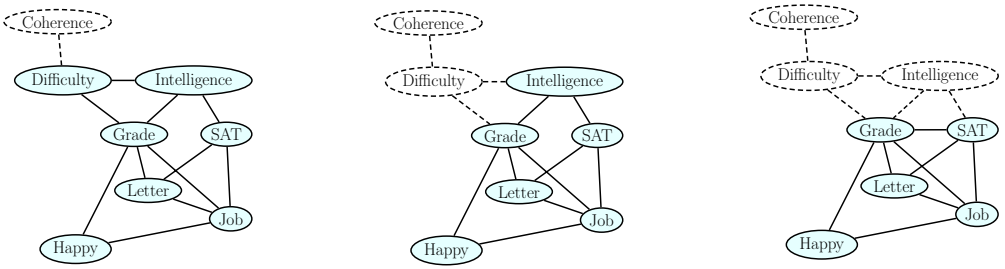


Figure 9.12: Example of the elimination process, in the order C, D, I, H, G, S, L. When we eliminate I (figure c), we add a fill-in edge between G and S, since they are not connected. Adapted from Figure 9.10 of [KF09a].

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

$$\tau_1'(C,D) = \psi_C(C)\psi_D(D,C) \tag{9.36}$$

Then we marginalize out C to get the new factor

$$\tau_1(D) = \sum_C \tau_1'(C,D) \tag{9.37}$$

Next we multiply together all the terms in the scope of the $\sum_D$ operator and then marginalize out to create

$$\tau_2'(G,I,D) = \psi_G(G,I,D)\tau_1(D) \tag{9.38}$$

$$\tau_2(G,I) = \sum_D \tau_2'(G,I,D) \tag{9.39}$$

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,

$$p(J = j|I = 1, H = 0) = \frac{p(J = j, I = 1, H = 0)}{\sum_{j'} p(J = j', I = 1, H = 0)} \tag{9.40}$$

9.4.2 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 9.12 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

$$\tau'_3(G,S,I) = \psi_S(S,I)\psi_I(I)\tau_2(G,I) \tag{9.41}$$

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:

$$\{C,D\}, \{D,I,G\}, \{G,L,S,J\}, \{G,J,H\}, \{G,I,S\} \tag{9.42}$$

It is clear that the time complexity of VE is

$$\sum_{c \in \mathcal{C}(G_\prec)} K^{|c|} \tag{9.43}$$

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 9.4.3 for a discussion of some approximate methods for finding good elimination orders.

9.4.3 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.[3] 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 $O(N^{3/2})$.

9.4.4 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.

3. See the description of the symamd command at https://bit.ly/31N6E2b. ("sym" stands for symbolic, "amd" stands approximate minimum degree.)

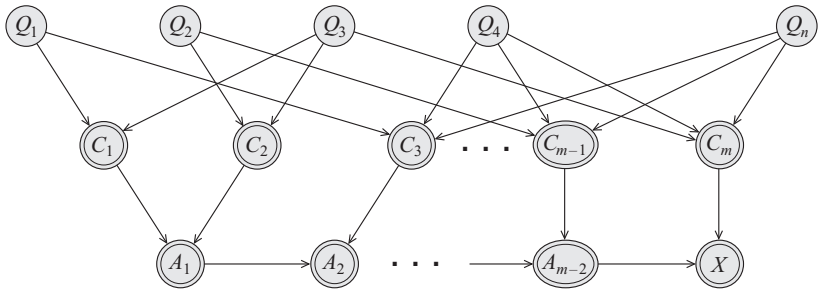


Figure 9.13: Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final 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 9.13. 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 3.2). Inference in mixed discrete-continuous models can also be hard [LP01].

As a consequence of these hardness results, we often have to resort to approximate inference methods, such as variational inference (Chapter 10) and Monte Carlo inference (Chapter 11).

9.4.5 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 8.2.4. But now suppose we want to compute $p(z_{T-1}|\boldsymbol{y})$. We have to run VE again, at a cost

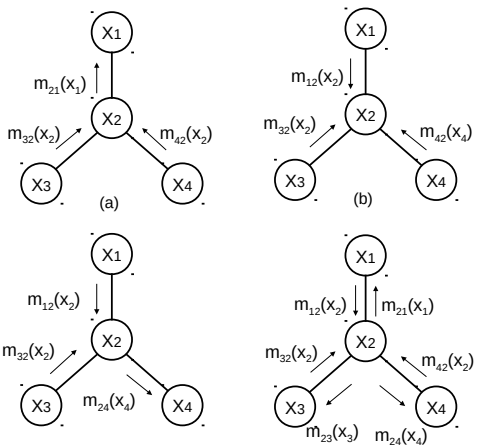


Figure 9.14: Sending multiple messages along a tree. (a) z_1 is root. (b) z_2 is root. (c) z_4 is root. (d) All of the messages needed to compute all singleton marginals. Adapted from Figure 4.3 of [Jor07].

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 8.2.4. 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 9.14. 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 9.2.

The question is: how do we get these benefits of message passing on a tree when the graph is not a tree? We give the answer in Section 9.5.

9.5 The junction tree algorithm (JTA)

The **junction tree algorithm** or **JTA** is a generalization of variable elimination that lets us efficiently compute all the posterior marginals without repeating redundant work, by using dynamic programming, thus avoiding the problems mentioned in Section 9.4.5. 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 9.4.2, 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 9.2.2. See Supplementary Section 9.2.2 for details.

9.6 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.

9.6.1 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 2.3) follows:

$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_c \psi_c(\boldsymbol{x}_c) = \exp(\sum_c \boldsymbol{\eta}_c^\mathsf{T}\mathcal{T}(\boldsymbol{x}_c) - \log A(\boldsymbol{\eta})) = \exp(\boldsymbol{\eta}^\mathsf{T}\mathcal{T}(\boldsymbol{x}) - \log A(\boldsymbol{\eta})) \tag{9.44}$$

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

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{T}(\boldsymbol{x})] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s\neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{st;jk}\}_{s\neq t}) \tag{9.45}$$

The key result, from Equation (2.219), 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]).

9.6.1.1 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

$$Z(\boldsymbol{\psi}) = \sum_{x_1,x_2,x_3} \psi_1(x_1)\psi_2(x_2)\psi_3(x_3)\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3) \tag{9.46}$$

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.[4] 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].

9.6.2 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].

9.6.2.1 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

$$p(\boldsymbol{x}) \propto \mathcal{N}(\mathbf{G}\boldsymbol{x}|\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p) \propto \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_x\boldsymbol{x} + \boldsymbol{h}_x^\mathsf{T}\boldsymbol{x}) \tag{9.47}$$

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

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\mathbf{H}\boldsymbol{x} + \boldsymbol{c}, \boldsymbol{\Sigma}_n) \propto \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_{y|x}\boldsymbol{x} + \boldsymbol{h}_{y|x}^\mathsf{T}\boldsymbol{x} - \frac{1}{2}\boldsymbol{y}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}\boldsymbol{y}) \tag{9.48}$$

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 (c.f., one step of the information filter in Section 8.3.4)

$$p(\boldsymbol{x}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{9.49}$$

$$\boldsymbol{\Sigma}^{-1} = \mathbf{K} = \mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\mathbf{G} + \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}\mathbf{H} \tag{9.50}$$

$$\boldsymbol{\mu} = \mathbf{K}(\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\mu}_p + \mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}(\boldsymbol{y} - \boldsymbol{c})) \tag{9.51}$$

4. 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":

$$p(\boldsymbol{x}|\boldsymbol{y}) \propto \prod_{l=1}^{L} \exp(-\frac{1}{2}\boldsymbol{x}^\mathsf{T}\mathbf{K}_l\boldsymbol{x} + \boldsymbol{h}_l^\mathsf{T}\boldsymbol{x}) \propto \prod_{l=1}^{L} \mathcal{N}(\boldsymbol{f}_l^\mathsf{T}\boldsymbol{x}; \mu_l, \Sigma_l) \tag{9.52}$$

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:

$$\tilde{\boldsymbol{x}} = \mathbf{K}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}\tilde{\boldsymbol{\mu}}_p + \mathbf{K}^{-1}\mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}(\tilde{\boldsymbol{y}} - \boldsymbol{c}) \tag{9.53}$$

$$= \underbrace{\mathbf{K}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\Sigma}_p^{-1}}_{\mathbf{A}}(\boldsymbol{\mu}_p + \boldsymbol{\epsilon}_\mu) + \underbrace{\mathbf{K}^{-1}\mathbf{H}^\mathsf{T}\boldsymbol{\Sigma}_n^{-1}}_{\mathbf{B}}(\boldsymbol{y} + \boldsymbol{\epsilon}_y - \boldsymbol{c}) \tag{9.54}$$

$$= \boldsymbol{\mu} + \mathbf{A}\boldsymbol{\epsilon}_\mu + \mathbf{B}\boldsymbol{\epsilon}_y \tag{9.55}$$

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 9.3), which can often compute the exact posterior mean (see e.g., [WF01a; JMW06; Bic09; Du+18]).

9.6.2.2 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

$$S(\boldsymbol{x}; \boldsymbol{c}) = \log p(\boldsymbol{x}|\boldsymbol{c}) + \text{const} = \sum_i \log \phi_i(x_i) + \sum_{ij} \log \psi_{ij}(x_{i,j}) \tag{9.56}$$

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; Zha+19b; Bro18].

10.1.1 Variational free energy

Consider a model with unknown variables $\boldsymbol{z}$, known variables $\boldsymbol{x}$, and fixed parameters $\boldsymbol{\theta}$. (If the parameters are unknown, they can be added to $\boldsymbol{z}$.) Since computing the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is assumed intractable, we will use an approximation $q(\boldsymbol{z})$, which we choose to 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 $\boldsymbol{\psi}$ to represent the **variational parameters**. We compute the best variational parameters (for given $\boldsymbol{x}$) as follows:

$$\boldsymbol{\psi}^* = \operatorname*{argmin}_{\boldsymbol{\psi}} D_{\mathbb{KL}}\left(q(\boldsymbol{z}|\boldsymbol{\psi}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \tag{10.2}$$

$$= \operatorname*{argmin}_{\boldsymbol{\psi}} \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log q(\boldsymbol{z}|\boldsymbol{\psi}) - \log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right)\right] \tag{10.3}$$

$$= \operatorname*{argmin}_{\boldsymbol{\psi}} \underbrace{\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log q(\boldsymbol{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right]}_{\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta},\boldsymbol{x})} + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.4}$$

The final term $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}$ 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 using the following shorthand:[1]

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) = D_{\mathbb{KL}}\left(q(\boldsymbol{z}|\boldsymbol{\psi}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right) \triangleq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log q(\boldsymbol{z}|\boldsymbol{\psi}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] \tag{10.5}$$

1. Technically speaking $D_{\mathbb{KL}}\left(q(\boldsymbol{z}|\boldsymbol{\psi}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right)$ is not a KL divergence, since these are distributions over different spaces: the first is a conditional distribution over $\boldsymbol{z}$, the second is a joint distribution over $\boldsymbol{z}$ and $\boldsymbol{x}$. However, hopefully this shorthand is clear.

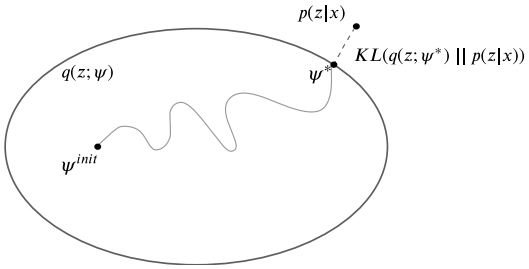


Figure 10.1: Illustration of variational inference. The large oval represents the set of variational distributions $\mathcal{Q} = \{q(\boldsymbol{z};\boldsymbol{\psi}) : \boldsymbol{\psi} \in \mathbb{R}^K\}$, where K is the number of variational parameters. The true distribution is the point $p(\boldsymbol{z}|\boldsymbol{x})$, 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.

If we define $\mathcal{E}(\boldsymbol{z}) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$ as the energy, then we can write

$$\mathcal{L}(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\mathcal{E}(\boldsymbol{z})\right] - \mathbb{H}(q) \tag{10.6}$$

where $\mathbb{H}(q)$ is the entropy. In physics, this is known as the **variational free energy**. We can interpret this as the expected energy minus the entropy:

$$\text{VFE} = \text{expected energy} - \text{entropy} \tag{10.7}$$

This is an upper bound on the **free energy**, $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which follows from the fact that

$$D_{\mathbb{KL}}\left(q \parallel p\right) = \text{VFE}(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \tag{10.8}$$

Our goal is to minimize the VFE. See Figure 10.1 for an illustration.

10.1.2 Evidence lower bound (ELBO)

The negative of the VFE is known as the **evidence lower bound** or **ELBO** function [BKM16]:

$$Ł(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) \triangleq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{\psi})\right] \tag{10.9}$$

where $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$ is the unnormalized log joint. The name "ELBO" arises because

$$Ł(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.10}$$

where $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the called the "evidence". The inequality follows from Equation (10.8). Therefore maximizing the ELBO wrt $\boldsymbol{\psi}$ will decrease 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 the latter denotes a loss we want to minimize.)

We can rewrite the ELBO as follows:

$$Ł(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}(q(\boldsymbol{z}|\boldsymbol{\psi})) \tag{10.11}$$

We can interpret this

$$\text{ELBO} = \text{expected log joint} + \text{entropy of posterior} \tag{10.12}$$

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 in the following equivalent way:

$$Ł(\boldsymbol{\psi}|\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{\psi})\right] \tag{10.13}$$

$$= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{\psi})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\mathbb{KL}}\left(q(\boldsymbol{z}|\boldsymbol{\psi}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) \tag{10.14}$$

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.

(See also Section 21.2.2, where we discuss the ELBO in more detail, in the context of variational autoencoders.)

10.2 Mean field VI

A common approximation in variational inference is to assume that all the latent variables are independent, i.e.,

$$q(\boldsymbol{z}|\boldsymbol{\psi}) = \prod_{j=1}^{J} q_j(z_j) \tag{10.16}$$

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

$$Ł(\boldsymbol{\psi}) = \int q(\boldsymbol{z}|\boldsymbol{\psi}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} + \sum_{j=1}^{J} \mathbb{H}(q_j) \tag{10.17}$$

since the entropy of a product distribution is the sum of entropies of each component in the product. We can either directly optimize this (see e.g., [Baq+16]), or use a coordinate-wise optimization scheme, as we discuss in Section 10.2.1.

10.2.1 Coordinate ascent variational inference (CAVI)

We npw discuss a coordinate ascent method for optimizing the mean field objective, which we call **coordinate ascent variational inference** or **CAVI**.

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

$$Ł(q_1, q_2, q_3) = \sum_{z_1}\sum_{z_2}\sum_{z_3} q_1(z_1) q_2(z_2) q_3(z_3) \log \tilde{p}(z_1, z_2, z_3) + \sum_{j=1}^{3} \mathbb{H}(q_j) \tag{10.18}$$

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:

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\sum_{z_1} \sum_{z_2} q_1(z_1) q_2(z_2) \log \tilde{p}(z_1, z_2, z_3) \right] + \mathbb{H}(q_3) + \text{const} \tag{10.19}$$

$$= \sum_{z_3} q_3(z_3) \left[g_3(z_3) - \log q_3(z_3) \right] + \text{const} \tag{10.20}$$

where

$$g_3(z_3) \triangleq \sum_{z_1} \sum_{z_2} q_1(z_1) q_2(z_2) \log \tilde{p}(z_1, z_2, z_3) = \mathbb{E}_{\boldsymbol{z}_{-3}} \left[\log \tilde{p}(z_1, z_2, z_3) \right] \tag{10.21}$$

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

$$\tilde{f}_3(z_3) = \exp(g_3(z_3)) \tag{10.22}$$

which we can normalize to get

$$f_3(z_3) = \frac{\tilde{f}_3(z_3)}{\sum_{z_3'} \tilde{f}_3(z_3')} \propto \exp(g_3(z_3)) \tag{10.23}$$

Since $g_3(z_3) \propto \log f_3(z_3)$ we get

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\log f_3(z_3) - \log q_3(z_3) \right] + \text{const} = -D_{\mathbb{KL}} \left(q_3 \parallel f_3 \right) + \text{const} \tag{10.24}$$

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

$$Ł_3(q_3) = \sum_{z_3} q_3(z_3) \left[\sum_{z_2} q_2(z_2) \log \tilde{p}(z_2, z_3) \right] + \mathbb{H}(q_3) + \text{const} \tag{10.25}$$

$$= \sum_{z_3} q_3(z_3) \left[\log f_3(z_3) - \log q_3(z_3) \right] + \text{const} \tag{10.26}$$

where

$$f_3(z_3) \propto \exp \left[\sum_{z_2} q_2(z_2) \log \tilde{p}(z_2, z_3) \right] = \exp \left[\mathbb{E}_{\boldsymbol{z}_{\text{mb}_3}} \left[\log \tilde{p}(z_2, z_3) \right] \right] \tag{10.27}$$

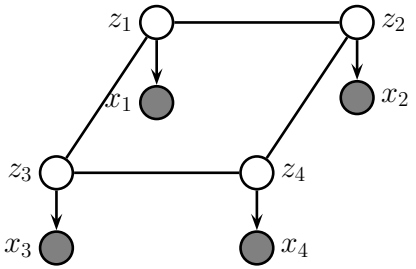


Figure 10.2: A grid-structured MRF with hidden nodes z_i and local evidence nodes x_i. The prior $p(\boldsymbol{z})$ is an undirected Ising model, and the likelihood $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(x_i|z_i)$ is a directed fully factored model.

where $\text{mb}_3 = (z_2)$ 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

$$q_j(\boldsymbol{z}_j) \propto \exp\left[\mathbb{E}_{\boldsymbol{z}_{\text{mb}_j}}\left[\log \tilde{p}(\boldsymbol{z}_j, \boldsymbol{z}_{\text{mb}_j})\right]\right] \tag{10.28}$$

(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 19). Convergence is guaranteed since the bound is convex wrt each of the factors q_i [Bis06, p. 466].

Algorithm 19: 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 10.3). We give some examples below that will make this clearer.

10.2.2 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 10.2 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

$$p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(x_i|z_i) = \exp(\sum_i L_i(z_i)) \tag{10.29}$$

Our goal is to approximate the posterior $p(\boldsymbol{z}|\boldsymbol{x})$. We will use an Ising model for the prior:

$$p(\boldsymbol{z}) = \frac{1}{Z_0} \exp(-\mathcal{E}_0(\boldsymbol{z})) \tag{10.30}$$

$$\mathcal{E}_0(\boldsymbol{z}) = -\sum_{i \sim j} W_{ij} z_i z_j \tag{10.31}$$

where we sum over each $i - j$ edge. Therefore the posterior has the form

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp(-\mathcal{E}(\boldsymbol{z})) \tag{10.32}$$

$$\mathcal{E}(\boldsymbol{z}) = \mathcal{E}_0(\boldsymbol{z}) - \sum_i L_i(z_i) \tag{10.33}$$

We will now make the following fully factored approximation:

$$q(\boldsymbol{z}) = \prod_i q_i(z_i) = \prod_i \text{Ber}(z_i|\mu_i) \tag{10.34}$$

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:

$$\log \tilde{p}(\boldsymbol{z}) = z_i \sum_{j \in \text{nbr}_i} W_{ij} z_j + L_i(z_i) + \text{const} \tag{10.35}$$

This only depends on the states of the neighboring nodes. Hence

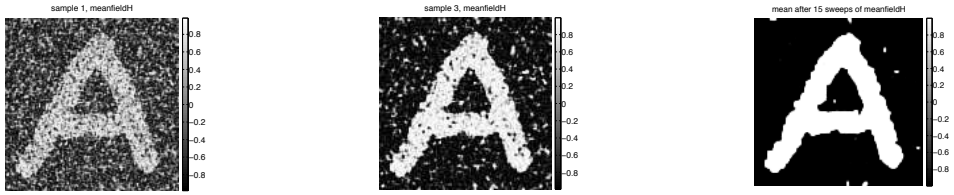
$$q_i(z_i) \propto \exp(\mathbb{E}_{q_{-i}(\boldsymbol{z})}[\log \tilde{p}(\boldsymbol{z})]) = \exp\left(z_i \sum_{j \in \text{nbr}_i} W_{ij} \mu_j + L_i(z_i)\right) \tag{10.36}$$

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

$$q_i(z_i = 1) = \frac{e^{m_i + L_i^+}}{e^{m_i + L_i^+} + e^{-m_i + L_i^-}} = \frac{1}{1 + e^{-2m_i + L_i^- - L_i^+}} = \sigma(2a_i) \tag{10.37}$$

$$a_i \triangleq m_i + 0.5(L_i^+ - L_i^-) \tag{10.38}$$



(a) (b) (c)

Figure 10.3: 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.

Similarly, we have $q_i(z_i = -1) = \sigma(-2a_i)$. From this we can compute the new mean for site i:

$$\mu_i = \mathbb{E}_{q_i}[z_i] = q_i(z_i = +1) \cdot (+1) + q_i(z_i = -1) \cdot (-1) \tag{10.39}$$

$$= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} = \frac{e^{a_i}}{e^{a_i} + e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i} + e^{a_i}} = \tanh(a_i) \tag{10.40}$$

We can turn the above equations into a fixed point algorithm by writing

$$\mu_i^t = \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{10.41}$$

Following [MWJ99], we can use **damped updates** of the following form to improve convergence:

$$\mu_i^t = (1 - \lambda)\mu_i^{t-1} + \lambda \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{10.42}$$

for $0 < \lambda < 1$. We can update all the nodes in parallel, or update them asychronously.

Figure 10.3 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" artefacts.)

10.2.3 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].

In this section, we assume there are no latent variables except for the shared global parameters, so the model has the form

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathcal{D}_n|\boldsymbol{\theta}) \tag{10.43}$$

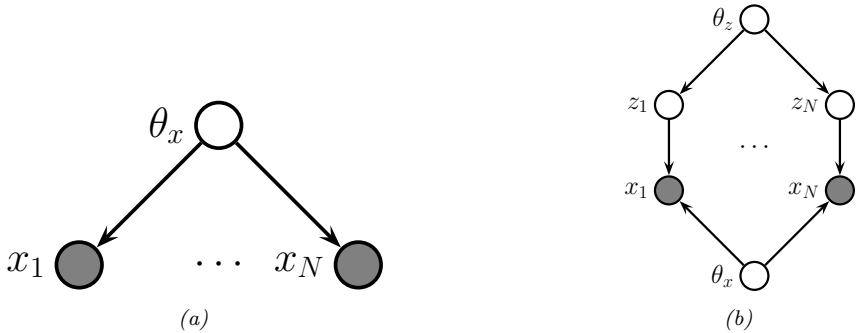


Figure 10.4: Graphical models with (a) Global hidden variable $\boldsymbol{\theta}_x$ and observed variables $\boldsymbol{x}_{1:N}$. (b) Local hidden variables $\boldsymbol{z}_{1:N}$, global hidden variables $\boldsymbol{\theta}_x, \boldsymbol{\theta}_z$, and observed variables $\boldsymbol{x}_{1:N}$.

These conditional independencies are illustrated in Figure 10.4a.

We will fit the variational posterior by maximizing the ELBO

$$Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})}\left[\log p(\boldsymbol{\theta}, \mathcal{D})\right] + \mathbb{H}(q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})) \tag{10.44}$$

We will assume the variational posterior factorizes over the parameters:

$$q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) = \prod_j q(\boldsymbol{\theta}_j|\boldsymbol{\psi}_{\theta_j}) \tag{10.45}$$

We can then update each $\boldsymbol{\psi}_{\theta_j}$ using CAVI (Section 10.2.1).

10.2.4 Example: VB for a univariate Gaussian

Consider inferring the parameters of a 1d Gaussian. The likelihood is given by $p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \lambda^{-1})$, where μ is the mean and λ is the precision. Suppose we use a conjugate prior of the form

$$p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, (\kappa_0\lambda)^{-1})\mathrm{Ga}(\lambda|a_0, b_0) \tag{10.46}$$

It is possible to derive the posterior $p(\mu, \lambda|\mathcal{D})$ for this model exactly, as shown in Section 3.2.3.3. However, here we use the VB method with the following factored approximate posterior:

$$q(\mu, \lambda) = q(\mu|\boldsymbol{\psi}_\mu)q(\lambda|\boldsymbol{\psi}_\lambda) \tag{10.47}$$

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].

10.2.4.1 Target distribution

The unnormalized log posterior has the form

$$\log \tilde{p}(\mu, \lambda) = \log p(\mu, \lambda, \mathcal{D}) = \log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda) \tag{10.48}$$

$$= \frac{N_{\mathcal{D}}}{2} \log \lambda - \frac{\lambda}{2} \sum_{n=1}^{N_{\mathcal{D}}} (x_n - \mu)^2 - \frac{\kappa_0 \lambda}{2} (\mu - \mu_0)^2$$

$$+ \frac{1}{2} \log(\kappa_0 \lambda) + (a_0 - 1) \log \lambda - b_0 \lambda + \text{const} \tag{10.49}$$

10.2.4.2 Updating $q(\mu|\boldsymbol{\psi}_\mu)$

The optimal form for $q(\mu|\boldsymbol{\psi}_\mu)$ is obtained by averaging over λ:

$$\log q(\mu|\boldsymbol{\psi}_\mu) = \mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)} \left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda)\right] + \text{const} \tag{10.50}$$

$$= -\frac{\mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)} [\lambda]}{2} \left\{ \kappa_0 (\mu - \mu_0)^2 + \sum_{n=1}^{N_{\mathcal{D}}} (x_n - \mu)^2 \right\} + \text{const} \tag{10.51}$$

By completing the square one can show that $q(\mu|\boldsymbol{\psi}_\mu) = \mathcal{N}(\mu|\mu_N, \kappa_N^{-1})$, where

$$\mu_N = \frac{\kappa_0 \mu_0 + N_{\mathcal{D}} \overline{x}}{\kappa_0 + N_{\mathcal{D}}}, \ \kappa_N = (\kappa_0 + N_{\mathcal{D}}) \mathbb{E}_{q(\lambda|\boldsymbol{\psi}_\lambda)} [\lambda] \tag{10.52}$$

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.

10.2.4.3 Updating $q(\boldsymbol{\lambda}|\boldsymbol{\psi}_{\boldsymbol{\lambda}})$

The optimal form for $q(\lambda|\boldsymbol{\psi}_\lambda)$ is given by

$$\log q(\lambda|\boldsymbol{\psi}_\lambda) = \mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)} \left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda)\right] + \text{const} \tag{10.53}$$

$$= (a_0 - 1) \log \lambda - b_0 \lambda + \frac{1}{2} \log \lambda + \frac{N_{\mathcal{D}}}{2} \log \lambda$$

$$- \frac{\lambda}{2} \mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)} \left[\kappa_0 (\mu - \mu_0)^2 + \sum_{n=1}^{N_{\mathcal{D}}} (x_n - \mu)^2 \right] + \text{const} \tag{10.54}$$

We recognize this as the log of a Gamma distribution, hence $q(\lambda|\boldsymbol{\psi}_\lambda) = \text{Ga}(\lambda|a_N, b_N)$, where

$$a_N = a_0 + \frac{N+1}{2} \tag{10.55}$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}_{q(\mu|\boldsymbol{\psi}_\mu)} \left[\kappa_0 (\mu - \mu_0)^2 + \sum_{n=1}^{N_{\mathcal{D}}} (x_n - \mu)^2 \right] \tag{10.56}$$

10.2.4.4 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

$$\mathbb{E}_{q(\mu)}[\mu] = \mu_N \tag{10.57}$$

$$\mathbb{E}_{q(\mu)}[\mu^2] = \frac{1}{\kappa_N} + \mu_N^2 \tag{10.58}$$

Since $q(\lambda) = \text{Ga}(\lambda|a_N, b_N)$, we have

$$\mathbb{E}_{q(\lambda)}[\lambda] = \frac{a_N}{b_N} \tag{10.59}$$

We can now give explicit forms for the update equations. For $q(\mu)$ we have

$$\mu_N = \frac{\kappa_0\mu_0 + N_{\mathcal{D}}\overline{x}}{\kappa_0 + N_{\mathcal{D}}} \tag{10.60}$$

$$\kappa_N = (\kappa_0 + N_{\mathcal{D}})\frac{a_N}{b_N} \tag{10.61}$$

and for $q(\lambda)$ we have

$$a_N = a_0 + \frac{N+1}{2} \tag{10.62}$$

$$b_N = b_0 + \frac{1}{2}\kappa_0(\mathbb{E}[\mu^2] + \mu_0^2 - 2\mathbb{E}[\mu]\mu_0) + \frac{1}{2}\sum_{n=1}^{N_{\mathcal{D}}}\left(x_n^2 + \mathbb{E}[\mu^2] - 2\mathbb{E}[\mu]x_n\right) \tag{10.63}$$

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.)

10.2.4.5 Illustration

Figure 10.5 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 5.1.3.3.

10.2.4.6 Lower bound

In VB, we maximize a lower bound on the log marginal likelihood:

$$Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) \leq \log p(\mathcal{D}) = \log \iint p(\mathcal{D}|\mu,\lambda)p(\mu,\lambda)d\mu d\lambda \tag{10.64}$$

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

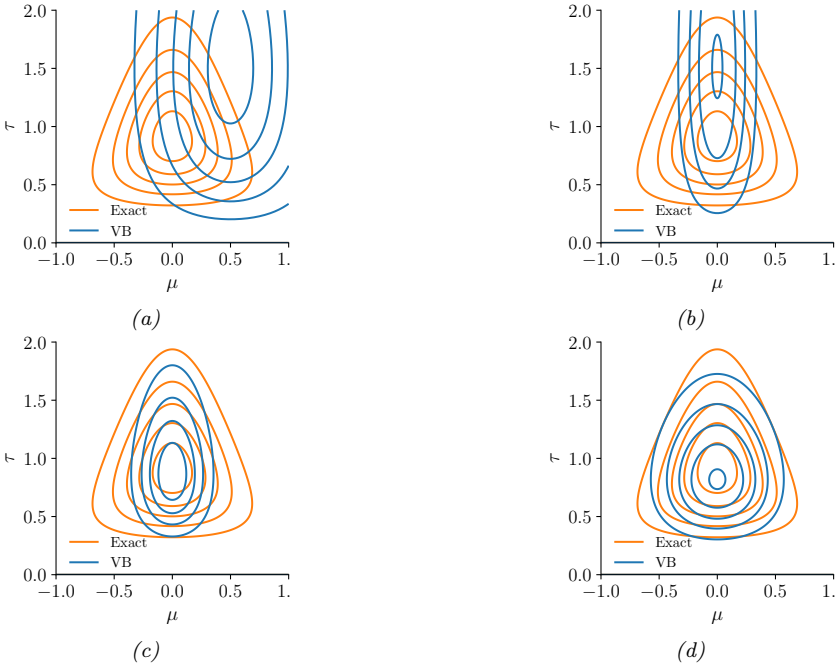


Figure 10.5: Factored variational approximation (red) to the Gaussian-Gamma distribution (green). (a) Initial guess. (b) After updating $q(\mu|\boldsymbol{\psi}_\mu)$. (c) After updating $q(\lambda|\boldsymbol{\psi}_\lambda)$. (d) At convergence (after 5 iterations). Adapted from Fig. 10.4 of [Bis06]. 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. One can show that the lower bound has the following form:

$$Ł = \text{const} + \frac{1}{2}\ln\frac{1}{\kappa_N} + \ln\Gamma(a_N) - a_N \ln b_N \tag{10.65}$$

10.2.5 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 10.4b 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

$$p(\boldsymbol{h}, \mathcal{D}) = p(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}, \mathcal{D}) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(\boldsymbol{z}_n|\boldsymbol{\theta}) p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\theta}) \tag{10.66}$$

We will make the following mean field assumption:

$$q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}}) = q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) \prod_{n=1}^{N} q(\boldsymbol{z}_n|\boldsymbol{\psi}_n) \tag{10.67}$$

Henceforth we will usually omit the variational parameters $(\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}})$ for brevity.

We will use VI to maximize the ELBO:

$$Ł(\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N}, \boldsymbol{\psi}_{\boldsymbol{\theta}})} \left[\log p(\boldsymbol{z}_{1:N}, \boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta}, \boldsymbol{z}_{1:N})\right] \tag{10.68}$$

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 (10.68) simplifies to the the "LVM ELBO":

$$Ł(\boldsymbol{\psi}_{1:N}, \boldsymbol{\theta}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N})} \left[\log p(\boldsymbol{\theta}, \mathcal{D}, \boldsymbol{z}_{1:N}) - \log q(\boldsymbol{z}_{1:N}|\boldsymbol{\psi}_{1:N})\right] \tag{10.69}$$

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 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 plugin a point estimate of the model parameters, rather than averaging over the parameters. For more details on VEM, see Section 6.6.6.1.

10.2.6 Example: VBEM for a GMM

Consider a standard Gaussian mixture model (GMM):

$$p(\boldsymbol{z}, \boldsymbol{x}|\boldsymbol{\theta}) = \prod_n \prod_k \pi_k^{z_{nk}} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}} \tag{10.70}$$

where $z_{nk} = 1$ if data point 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

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\breve{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k|\breve{\boldsymbol{m}}, (\breve{\kappa}\,\boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k|\breve{\mathbf{L}}, \breve{\nu}) \tag{10.71}$$

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 10.3.5.3, where we discuss a different variational approximation for this model.)

10.2.6.1 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:

$$q_n(z_n) = \text{Cat}(\boldsymbol{z}_n|\boldsymbol{r}_n) \tag{10.72}$$

$$q(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\, \widehat{\boldsymbol{\alpha}}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k|\, \widehat{\boldsymbol{m}}_k, (\widehat{\kappa}_k\, \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k|\, \widehat{\mathbf{L}}_k, \widehat{\nu}_k) \tag{10.73}$$

where $\boldsymbol{r}_n$ are the posterior responsibilities, and the parameters with hats on them are the hyperparameters from the prior updated with data.

10.2.6.2 Derivation of $q(\boldsymbol{\theta})$ (variational M step)

Using the mean field recipe in Algorithm 19, we write down the log joint, and take expectations over all variables except $\boldsymbol{\theta}$, so we average out the $\boldsymbol{z}_n$ wrt $q(\boldsymbol{z}_n) = \text{Cat}(z_n|\boldsymbol{r}_n)$:

$$\log q(\boldsymbol{\theta}) = \underbrace{\log p(\boldsymbol{\pi}) + \sum_n \mathbb{E}_{q(z_n)}\left[\log p(\boldsymbol{z}_n|\boldsymbol{\pi})\right]}_{L_{\boldsymbol{\pi}}} + \sum_k \left[\underbrace{\log p(\mu_k, \boldsymbol{\Lambda}_k) \sum_n \mathbb{E}_{q(z_n)}\left[z_{nk}\right] \log \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})}_{L_{\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k}} \right] + \text{const} \tag{10.74}$$

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

$$q(\boldsymbol{\theta}) = q(\boldsymbol{\pi}) \prod_k q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \tag{10.75}$$

For the $\boldsymbol{\pi}$ term, we have

$$\log q(\boldsymbol{\pi}) = (\alpha_0 - 1) \sum_k \log \pi_k + \sum_k \sum_n r_{nk} \log \pi_k + \text{const} \tag{10.76}$$

Exponentiating, we recognize this as a Dirichlet distribution:

$$q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi}|\, \widehat{\boldsymbol{\alpha}}) \tag{10.77}$$

$$\widehat{\alpha}_k = \alpha_0 + N_k \tag{10.78}$$

$$N_k = \sum_n r_{nk} \tag{10.79}$$

For the $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$ terms, we have

$$q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}(\boldsymbol{\mu}_k | \, \widehat{\boldsymbol{m}}_k, (\widehat{\kappa}_k \, \boldsymbol{\Lambda}_k)^{-1}) \mathrm{Wi}(\boldsymbol{\Lambda}_k | \, \widehat{\mathbf{L}}_k, \widehat{\nu}_k) \tag{10.80}$$

$$\widehat{\kappa}_k = \breve{\kappa} + N_k \tag{10.81}$$

$$\widehat{\boldsymbol{m}}_k = (\breve{\kappa} \, \breve{\boldsymbol{m}} + N_k \overline{\boldsymbol{x}}_k) / \, \widehat{\kappa}_k \tag{10.82}$$

$$\widehat{\mathbf{L}}_k^{-1} = \breve{\mathbf{L}}^{-1} + N_k \mathbf{S}_k + \frac{\breve{\kappa} \, N_k}{\breve{\kappa} + N_k} (\overline{\boldsymbol{x}}_k - \breve{\boldsymbol{m}})(\overline{\boldsymbol{x}}_k - \breve{\boldsymbol{m}})^\mathsf{T} \tag{10.83}$$

$$\widehat{\nu}_k = \breve{\nu} + N_k \tag{10.84}$$

$$\overline{\boldsymbol{x}}_k = \frac{1}{N_k} \sum_n r_{nk} \boldsymbol{x}_n \tag{10.85}$$

$$\mathbf{S}_k = \frac{1}{N_k} \sum_n r_{nk} (\boldsymbol{x}_n - \overline{\boldsymbol{x}}_k)(\boldsymbol{x}_n - \overline{\boldsymbol{x}}_k)^\mathsf{T} \tag{10.86}$$

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 $\hat{\boldsymbol{\theta}}$.

10.2.6.3 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

$$\log q(\boldsymbol{z}) = \sum_n \sum_k z_{nk} \left(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k] + \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] - \frac{D}{2} \log(2\pi) \right.$$
$$\left. - \frac{1}{2} \mathbb{E}_{q(\boldsymbol{\theta})} \left[(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\mathsf{T} \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right] \right) + \text{const} \tag{10.87}$$

Using the fact that $q(\boldsymbol{\pi}) = \mathrm{Dir}(\boldsymbol{\pi} | \, \widehat{\boldsymbol{\alpha}})$, one can show that

$$\exp(\mathbb{E}_{q(\boldsymbol{\pi})} [\log \pi_k]) = \frac{\exp(\psi(\widehat{\alpha}_k))}{\exp(\psi(\sum_{k'} \widehat{\alpha}_{k'}))} \triangleq \tilde{\pi}_k \tag{10.88}$$

where ψ is the **digamma function**

$$\psi(x) = \frac{d}{dx} \log \Gamma(x) \tag{10.89}$$

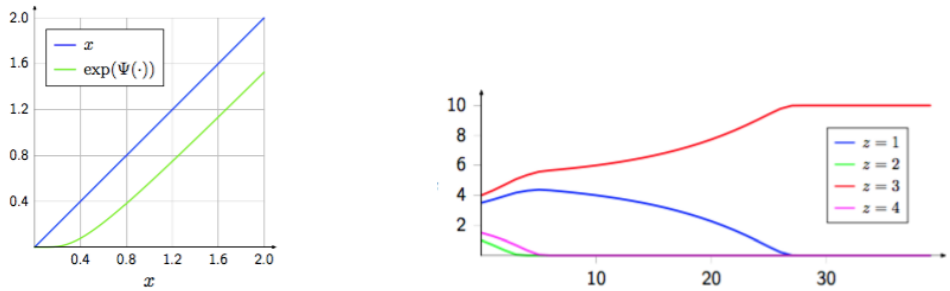
This takes care of the first term.

For the second term, one can show

$$\mathbb{E}_{q(\boldsymbol{\Lambda}_k)} [\log |\boldsymbol{\Lambda}_k|] = \sum_{j=1}^{D} \psi \left(\frac{\widehat{\nu}_k + 1 - j}{2} \right) + D \log 2 + \log | \, \widehat{\mathbf{L}}_k \, | \tag{10.90}$$

Finally, for the expected value of the quadratic form, one can show

$$\mathbb{E}_{q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)} \left[(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\mathsf{T} \boldsymbol{\Lambda}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right] = D \, \widehat{\kappa}_k^{-1} + \widehat{\nu}_k \, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^\mathsf{T} \, \widehat{\mathbf{L}}_k \, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k) \triangleq \tilde{\Lambda}_k \tag{10.91}$$



(a) *(b)*

Figure 10.6: (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

$$r_{nk} \propto \tilde{\pi}_k \tilde{\Lambda}_k^{\frac{1}{2}} \exp\left(-\frac{D}{2\,\widehat{\kappa}_k} - \frac{\widehat{\nu}_k}{2}(\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)^\intercal\, \widehat{\boldsymbol{\Lambda}}_k\, (\boldsymbol{x}_n - \widehat{\boldsymbol{m}}_k)\right) \tag{10.92}$$

Compare this to the expression used in regular EM:

$$r_{nk}^{EM} \propto \hat{\pi}_k |\hat{\boldsymbol{\Lambda}}_k|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_k)^\intercal \hat{\boldsymbol{\Lambda}}_k (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_k)\right) \tag{10.93}$$

where $\hat{\pi}_k$ is the MAP estimate for π_k. The significance of this difference is discussed in Section 10.2.6.4.

10.2.6.4 Automatic sparsity inducing effects of VBEM

In regular EM, the E step has the form given in Equation (10.93), whereas in VBEM, the E step has the form given in Equation (10.92). 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 (10.88) we have

$$r_{nk}^{VB} = \tilde{\pi}_k = \frac{\exp(\psi(\widehat{\alpha}_k))}{\exp(\psi(\sum_{k'} \widehat{\alpha}_{k'}))} \tag{10.94}$$

And from the usual EM MAP estimation equations for GMM mixing weights (see e.g., [Mur22, Sec 8.7.3.4]) we have

$$r_{nk}^{EM} = \hat{\pi}_k = \frac{\widehat{\alpha}_k - 1}{\sum_{k'}(\widehat{\alpha}_{k'} - 1)} \tag{10.95}$$

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 2.8 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 data points 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:

$$\tilde{\pi}_k = \frac{\exp(\psi(N_k))}{\exp(\psi(\sum_{k'} N_{k'}))} \tag{10.96}$$

$$\hat{\pi}_k = \frac{N_k - 1}{\sum_{k'}(N_{k'} - 1)} \tag{10.97}$$

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 10.6(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 10.6(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 Section 31.3, when we discuss Dirichlet process mixture models.

The reason for this effect is shown in Figure 10.6(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 15.2.5). 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 10.7. In Figure 10.8, 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.

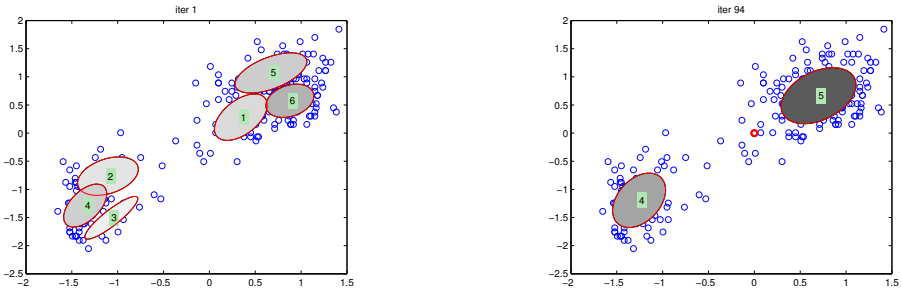
Thus we see that VBEM for GMMs with a sparse Dirichlet prior provides an efficient way to choose the number of clusters. Similar techniques can be used to choose the number of states in an HMM and other latent variable models. However, this **variational pruning effect** (also called **posterior collapse**), 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.

10.2.6.5 Lower bound on the marginal likelihood

The VBEM algorithm is maximizing the following lower bound

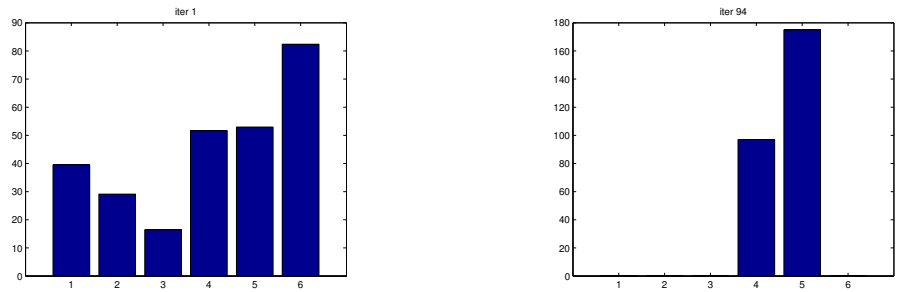
$$\mathcal{L} = \sum_{\boldsymbol{z}} \int d\boldsymbol{\theta}\, q(\boldsymbol{z}, \boldsymbol{\theta}) \log \frac{p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\theta})}{q(\boldsymbol{z}, \boldsymbol{\theta})} \leq \log p(\boldsymbol{x}) \tag{10.98}$$

This quantity increases monotonically with each iteration, as shown in Figure 10.9.



(a) (b)

Figure 10.7: 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 10.8: We visualize the posterior values of $\boldsymbol{\alpha}_k$ for the model in Figure 10.7 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.

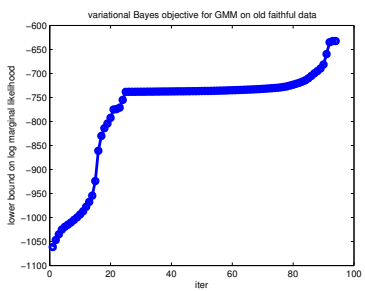


Figure 10.9: Lower bound vs iterations for the VB algorithm in Figure 10.7. The steep parts of the curve correspond to places where the algorithm figures out that it can increase the bound by "killing off" unnecessary mixture components, as described in Section 10.2.6.6. The plateaus correspond to slowly moving the clusters around. Generated by gmm_vb_em.ipynb.

10.2.6.6 Model selection using VBEM

Section 10.2.6.4 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

$$p(K|\mathcal{D}) = \frac{p(\mathcal{D}|K)}{\sum_{K'} p(\mathcal{D}|K')} \approx \frac{e^{\mathcal{L}(K)}}{\sum_{K'} e^{\mathcal{L}(K')}} \tag{10.99}$$

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 $\log p(\mathcal{D}|K) \approx \mathcal{L}(K) + \log(K!)$.

10.2.7 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.,

$$p(\boldsymbol{z}_j|\boldsymbol{z}_{-j}, \boldsymbol{x}) = h(\boldsymbol{z}_j) \exp[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j) - A_j(\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x}))] \tag{10.100}$$

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 (10.100) holds, the mean field update node j becomes

$$q_j(\boldsymbol{z}_j) \propto \exp\left[\mathbb{E}\left[\log p(\boldsymbol{z}_j|\boldsymbol{z}_{-j}, \boldsymbol{x})\right]\right] \tag{10.101}$$

$$= \exp\left[\log h(\boldsymbol{z}_j) + \mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right]^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j) - \mathbb{E}\left[A_j(\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x}))\right]\right] \tag{10.102}$$

$$\propto h(\boldsymbol{z}_j) \exp\left[\mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right]^\mathsf{T} \mathcal{T}(\boldsymbol{z}_j)\right] \tag{10.103}$$

Thus we update the local natural parameters using the expected values of the other nodes. These become the new variational parameters:

$$\boldsymbol{\psi}_j = \mathbb{E}\left[\boldsymbol{\eta}_j(\boldsymbol{z}_{-j}, \boldsymbol{x})\right] \tag{10.104}$$

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 9.2): 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.

10.2.8 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).

10.3 Fixed-form VI

Recall that the goal of variational inference is to maximize the following lower bound wrt the variational parameters $\boldsymbol{\psi}$:

$$Ł(\boldsymbol{\psi}|\mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{z})p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] = \mathbb{E}_{q_{\boldsymbol{\psi}}}\left[\ell_{\boldsymbol{\psi}}(\boldsymbol{z})\right] \tag{10.105}$$

where

$$\ell_{\boldsymbol{\psi}}(\boldsymbol{z}) = \log p(\boldsymbol{z}, \mathcal{D}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z}) \tag{10.106}$$

Here $\boldsymbol{z}$ are the unknown latent variables (or parameters), and $\boldsymbol{\psi}$ are the variational parameters. In the mean field method of Section 10.2, we assumed that $q(\boldsymbol{z})$ factorized across (groups of) variables, $q(\boldsymbol{z}|\boldsymbol{\psi}) = \prod_{j=1}^{J} q_j(z_j)$. We do not specify the form of q_j, so this approach is called "free-form" VI; however, the form of the optimal q_j can be derived analytically. We then optimized each q_j using coordinate ascent.

In this section, we take a different approach: we pick any convenient form we like for $q(\boldsymbol{z})$, such as a multivariate Gaussian, and then we directly maximize the ELBO using gradient ascent. This is called **fixed-form VI**.

10.3.1 Stochastic variational inference

In many models, the likelihood factorizes into a product of terms, in which case we have

$$\ell_{\boldsymbol{\psi}}(\boldsymbol{z}) = \left[\sum_{n=1}^{N} \log p(\boldsymbol{x}_n|\boldsymbol{z})\right] + \log p(\boldsymbol{z}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z}) \tag{10.107}$$

If N is large, we can compute an unbiased minibatch approximation to this expression as follows:

$$\hat{\ell}_{\boldsymbol{\psi}}(\boldsymbol{z}) = \left[\frac{N}{B}\sum_{b=1}^{B} \log p(\boldsymbol{x}_b|\boldsymbol{z})\right] + \log p(\boldsymbol{z}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z}) \tag{10.108}$$

We can use this to create an unbiased Monte Carlo approximation to the ELBO:

$$Ł(\boldsymbol{\psi}|\mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\psi}}}\left[\hat{\ell}_{\boldsymbol{\psi}}(\boldsymbol{z})\right] \tag{10.109}$$

This is called **stochastic variational inference** or **SVI** [Hof+13], and allows VI to scale to large datasets.

10.3.2 Black-box variational inference

In this section, we assume that we can evaluate $\ell_{\boldsymbol{\psi}}(\boldsymbol{z})$ 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 **black box variational inference** or **BBVI** [RGB14; ASD20].

To estimate the gradient of the ELBO, we will use the **score function estimator**, also called the **REINFORCE** estimator (Section 6.5.3). To derive this, we use the fact that $\nabla \log q = \frac{\nabla q}{q}$ to conclude that $\nabla q = q \nabla \log q$. (This is called the **log derivative trick**.) We also exploit the fact that $\int q(\boldsymbol{z}) d\boldsymbol{z} = 1$. With this, the gradient of the ELBO can be derived as follows:

$$\nabla_{\boldsymbol{\psi}} Ł(\boldsymbol{\psi}) = \nabla_{\boldsymbol{\psi}} \int q_{\boldsymbol{\psi}}(\boldsymbol{z}) \log \frac{p(\boldsymbol{z}, \mathcal{D})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})} d\boldsymbol{z} \tag{10.110}$$

$$= \int [\nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\log \frac{p(\boldsymbol{z}, \mathcal{D})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} + \int [q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\nabla_{\boldsymbol{\psi}} \log \frac{p(\boldsymbol{z}) p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} \tag{10.111}$$

$$= \int [\nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\log \frac{p(\boldsymbol{z}) p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} - \int q_{\boldsymbol{\psi}}(\boldsymbol{z}) \nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}) d\boldsymbol{z} \tag{10.112}$$

$$= \int [\nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\log \frac{p(\boldsymbol{z}) p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} - \int \nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z}) d\boldsymbol{z} \tag{10.113}$$

$$= \int [\nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\log \frac{p(\boldsymbol{z}) p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} - \nabla_{\boldsymbol{\psi}} \int q_{\boldsymbol{\psi}}(\boldsymbol{z}) d\boldsymbol{z} \tag{10.114}$$

$$= \int [\nabla_{\boldsymbol{\psi}} q_{\boldsymbol{\psi}}(\boldsymbol{z})] \left[\log \frac{p(\boldsymbol{z}) p(\mathcal{D}|\boldsymbol{z})}{q_{\boldsymbol{\psi}}(\boldsymbol{z})}\right] d\boldsymbol{z} \tag{10.115}$$

$$= \int q_{\boldsymbol{\psi}}(\boldsymbol{z}) \left(\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z})\ \ell_{\boldsymbol{\psi}}(\boldsymbol{z})\right) d\boldsymbol{z} \tag{10.116}$$

$$= \mathbb{E}_{q_{\boldsymbol{\psi}}(\boldsymbol{z})} \left[\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z})\ \ell_{\boldsymbol{\psi}}(\boldsymbol{z})\right] \tag{10.117}$$

We can compute a stochastic approximation to this gradient by sampling $\boldsymbol{z}_s \sim q_{\boldsymbol{\psi}}(\boldsymbol{z})$ and then computing

$$\widehat{\nabla_{\boldsymbol{\psi}} Ł(\boldsymbol{\psi}_t)} = \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}_s)\ \ell_{\boldsymbol{\psi}}(\boldsymbol{z}_s)|_{\boldsymbol{\psi}=\boldsymbol{\psi}_t} \tag{10.118}$$

We can pass this to any kind of gradient optimizer, such as SGD or Adam. If we further approximate $\ell_{\boldsymbol{\psi}}(\boldsymbol{z}_s)$ using the SVI minibatch approxoimation, we get a "**doubly stochastic**" approximation [TLG14].

In practice, the variance of this estimator is quite large, so it is important to use methods such as **control variates** or **CV** (Section 6.5.3.1). To see how this works, consider the naive gradient estimator in Equation (10.118), which for the i'th component we can write as

$$\widehat{\nabla_{\boldsymbol{\psi}_i} \text{Ł}(\boldsymbol{\psi}_t)}^{\text{naive}} = \frac{1}{S}\sum_{s=1}^{S} \tilde{g}_i(\boldsymbol{z}_s) \tag{10.119}$$

$$\tilde{g}_i(\boldsymbol{z}_s) = g_i(\boldsymbol{z}_s) \times \ell_{\boldsymbol{\psi}}(\boldsymbol{z}_s) \tag{10.120}$$

$$g_i(\boldsymbol{z}_s) = \nabla_{\boldsymbol{\psi}_i} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}_s) \tag{10.121}$$

The control variate version of this can be obtained by replacing $\tilde{g}_i(\boldsymbol{z}_s)$ with

$$\tilde{g}_i^{cv}(\boldsymbol{z}) = \tilde{g}_i(\boldsymbol{z}) + c_i(\mathbb{E}\left[b_i(\boldsymbol{z})\right] - b_i(\boldsymbol{z})) \tag{10.122}$$

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, $b_i(\boldsymbol{z}) = \nabla_{\boldsymbol{\psi}_i} \log q_{\boldsymbol{\psi}_i}(\boldsymbol{z})$, 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 (2.245). Hence

$$\tilde{g}_i^{cv}(\boldsymbol{z}) = \tilde{g}_i(\boldsymbol{z}) - c_i g_i(\boldsymbol{z}) = g_i(\boldsymbol{z})(\ell_{\boldsymbol{\psi}}(\boldsymbol{z}) - c_i) \tag{10.123}$$

so the CV estimator is given by

$$\widehat{\nabla_{\boldsymbol{\psi}_i} \text{Ł}(\boldsymbol{\psi}_t)}^{\text{cv}} = \frac{1}{S}\sum_{s=1}^{S} g_i(\boldsymbol{z}_s) \times (\ell_{\boldsymbol{\psi}}(\boldsymbol{z}_s) - c_i) \tag{10.124}$$

One can show that the optimal c_i that minimizes the variance of the CV estimator is

$$c_i = \frac{\text{Cov}\left[g_i(\boldsymbol{z})\ell_{\boldsymbol{\psi}}(\boldsymbol{z}), g_i(\boldsymbol{z})\right]}{\mathbb{V}\left[g_i(\boldsymbol{z})\right]} \tag{10.125}$$

which can be estimated by sampling $\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\boldsymbol{z})$. Thus the overall algorithm is as shown in Algorithm 20.

We can stop the algorithm when the lower bound stops increasing. We can compute a stochastic approximation to the lower bound using

$$\hat{\text{Ł}}(\boldsymbol{\psi}) = \frac{1}{S}\sum_{s=1}^{S} \ell_{\boldsymbol{\psi}}(\boldsymbol{z}_s) \tag{10.126}$$

where $\boldsymbol{z}_s \sim q_{\boldsymbol{\psi}}(\boldsymbol{z})$. To smooth out the noise, we can use a running average over the last w observations to get

$$\overline{\text{Ł}}(\boldsymbol{\psi}_t) = \frac{1}{w}\sum_{k=1}^{w} \hat{\text{Ł}}(\boldsymbol{\psi}_{t-k+1}) \tag{10.127}$$

If the moving average does not improve after P consecutive iterations, we declare convergence, where P is the **patience** parameter. Typical values are $P = 20$ and $w = 20$ [TND21].

Algorithm 20: Blackbox VI with control variates

Initialize $\boldsymbol{\psi}_0$
$\boldsymbol{z}_s \sim q_{\boldsymbol{\psi}_0}(\boldsymbol{z}),\ s = 1:S$
$h_{\boldsymbol{\psi}_0}(\boldsymbol{z}_s) = \log p(\boldsymbol{z}_s, \mathcal{D}) - \log q_{\boldsymbol{\psi}_0}(\boldsymbol{z}_s),\ s = 1:S$
$\boldsymbol{g}_0 = \frac{1}{S}\sum_{s=1}^{S}\left[\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}_0}(\boldsymbol{z}_s)\right] h_{\boldsymbol{\psi}_0}(\boldsymbol{z}_s)$
Compute $\boldsymbol{c}_1$ using Equation (10.125) applied to $\boldsymbol{z}_s$
for $t = 1:T$ **do**
  $\boldsymbol{z}_s \sim q_{\boldsymbol{\psi}_t}(\boldsymbol{z}),\ s = 1:S$
  $h_{\boldsymbol{\psi}_t}(\boldsymbol{z}_s) = \log p(\boldsymbol{z}_s, \mathcal{D}) - \log q_{\boldsymbol{\psi}_t}(\boldsymbol{z}_s),\ s = 1:S$
  $\boldsymbol{g}_t = \frac{1}{S}\sum_{s=1}^{S}\left[\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}_t}(\boldsymbol{z}_s)\right] \odot (h_{\boldsymbol{\psi}_t}(\boldsymbol{z}_s) - \boldsymbol{c}_t)$
  Compute $\boldsymbol{c}_{t+1}$ using Equation (10.125) applied to $\boldsymbol{z}_s$
  $\boldsymbol{\psi}_t = \text{gradient-update}(\boldsymbol{\psi}_{t-1}, \boldsymbol{g}_t)$

10.3.3 Reparameterization VI

In this section, we exploit the **reparameterization trick** from Section 6.5.4 to get a lower variance estimator for the gradient. This assumes that $\ell_{\boldsymbol{\psi}}(\boldsymbol{z})$ is a differentiable function of $\boldsymbol{z}$. It also assumes that we can sample $\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\boldsymbol{z})$ by first sampling a noise term $\boldsymbol{\epsilon} \sim q_0(\boldsymbol{\epsilon})$, and then transforming it to compute the latent random variables $\boldsymbol{z} = r(\boldsymbol{\psi}, \boldsymbol{\epsilon})$. In this case, the ELBO becomes

$$Ł(\boldsymbol{\psi}) = \mathbb{E}_{q_0(\boldsymbol{\epsilon})}\left[\ell_{\boldsymbol{\psi}}(r(\boldsymbol{\psi}, \boldsymbol{\epsilon}))\right] = \mathbb{E}_{q_0(\boldsymbol{\epsilon})}\left[\log p(r(\boldsymbol{\psi}, \boldsymbol{\epsilon}), \mathcal{D}) - \log q_{\boldsymbol{\psi}}(r(\boldsymbol{\psi}, \boldsymbol{\epsilon}))\right] \tag{10.128}$$

Since the sampling distribution $q_0(\boldsymbol{\epsilon})$ is independent of the variational parameters $\boldsymbol{\psi}$, we can push the gradient operator inside the expectation, and thus we can estimate the gradient using standard automatic differentiation methods, as shown in Algorithm 21. This is called **reparameterized VI** or **RVI**, and has provably lower variance than BBVI in certain cases [Xu+19].

Algorithm 21: Estimate of ELBO gradient

```
def elbo(ψ):
    ε ~ q_0(ε)
    z = r(ψ, ε)
    return log p(z, D) − q_ψ(z|D)
def elbo-grad(ψ):
    return grad(elbo(ψ))
```

10.3.3.1 "Sticking the landing" estimator

Applying the results from Section 6.5.4.2, we can derive the gradient estimate of the reparameterized ELBO, for a single Monte Carlo sample, as follows:

$$\nabla_{\boldsymbol{\psi}} \text{Ł}(\boldsymbol{\psi}, \boldsymbol{\epsilon}) = \nabla_{\boldsymbol{\psi}} \left[\log p(\boldsymbol{z}, \mathcal{D}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z}|\mathcal{D})\right] \tag{10.129}$$

$$= \underbrace{\nabla_{\boldsymbol{z}} \left[\log p(\boldsymbol{z}|\mathcal{D}) - \log q_{\boldsymbol{\psi}}(\boldsymbol{z}|\mathcal{D})\right] \mathbf{J}}_{\text{path derivative}} - \underbrace{\nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}|\mathcal{D})}_{\text{score function}} \tag{10.130}$$

where $\boldsymbol{z} = r(\boldsymbol{\psi}, \boldsymbol{\epsilon})$ and $\mathbf{J} = \nabla_{\boldsymbol{\psi}} r(\boldsymbol{\psi}, \boldsymbol{\epsilon})$ is the Jacobian matrix of the noise transformation.

The first term is the indirect effect of $\boldsymbol{\psi}$ on the objective via the generated samples $\boldsymbol{z}$. The second term is the direct effect of $\boldsymbol{\psi}$ on the objective. The second term is zero in expectation since it is the score function (see Equation (2.245)), but it may be non-zero for a finite number of samples, even if $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\mathcal{D}) = p(\boldsymbol{z}|\mathcal{D})$ is the true posterior. In a paper called "**sticking the landing**", [RWD17] propose to drop the second term to create a lower variance estimator.[2] In practice, this means we compute the gradient using Algorithm 22 instead of Algorithm 21.[3]

Algorithm 22: "Sticking the landing" estimator of ELBO gradient

```
def elbo-pd(ψ):
    ε ~ q_0(ε)
    z = r(ψ, ε)
    ψ' = stop-gradient(ψ)
    return log p(z, D) − q_ψ'(z|D)
def elbo-grad-pd(ψ):
    return grad(elbo-pd(ψ))
```

Note that the STL estimator is not always better than the "standard" estimator. In [GD20], they propose to use a weighted combination of estimators, where the weights are optimized so as to reduce variance for a fixed amount of compute.

10.3.3.2 Example: reparameterized SVI for GMMs

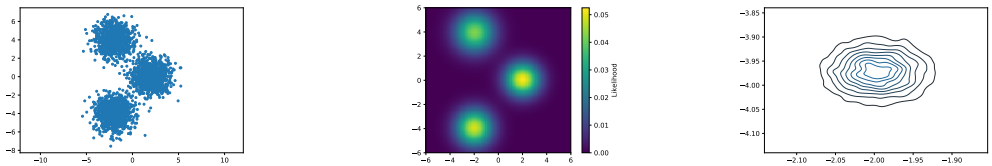
In this section, we use reparameterized SVI to fit a Gaussian mixture model. We will marginalize out the discrete latent variables, so just need to approximate $p(\boldsymbol{\theta}|\mathcal{D})$. We choose a factored variational posterior that is conjugate to the likelihood, but is also reparameterizable, so we can fit the posterior with SGD instead of having to use coordinate ascent (Section 10.2.1).

For simplicity, we assume diagonal covariance matrices. Thus the likelihood for one data point, $\boldsymbol{x} \in \mathbb{R}^D$, is

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\lambda}_k)^{-1}) \tag{10.131}$$

2. The expression "to stick a landing" means to land firmly on one's feet after performing a gymnastics move. In the current context, the analogy is this: if the variational posterior is optimal, so $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\mathcal{D}) = p(\boldsymbol{z}|\mathcal{D})$, then we want our objective to be 0, and not to "wobble" with Monte Carlo noise.

3. The difference is that the path derivative version ignores the score function. This can be achieved by using $\log q_{\boldsymbol{\psi}'}(\boldsymbol{z}|\mathcal{D})$, where $\boldsymbol{\psi}'$ is a "disconnected" copy of $\boldsymbol{\psi}$ that does not affect the gradient.



(a) (b) (c)

Figure 10.10: 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 $q(\boldsymbol{\mu}_1|\boldsymbol{\psi}_{\boldsymbol{\theta}})$. Generated by svi_gmm_demo_2d.ipynb.

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:

$$p(\boldsymbol{\theta}) = \left[\prod_{k=1}^{K}\prod_{d=1}^{D} \mathcal{N}(\mu_{kd}|0,1)\text{Ga}(\lambda_{kd}|5,5)\right] \text{Dir}(\boldsymbol{\pi}|\mathbf{1}) \tag{10.132}$$

We assume the following mean field posterior:

$$q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) = \left[\prod_{k=1}^{K}\prod_{d=1}^{D} \mathcal{N}(\mu_{kd}|m_{kd}, s_{kd})\text{Ga}(\lambda_{kd}|\alpha_{kd}, \beta_{kd})\right] \text{Dir}(\boldsymbol{\pi}|\boldsymbol{c}) \tag{10.133}$$

where $\boldsymbol{\psi}_{\boldsymbol{\theta}} = (\boldsymbol{m}_{1:K,1:D}, \boldsymbol{s}_{1:K,1:D}, \boldsymbol{\alpha}_{1:K,1:D}, \boldsymbol{\beta}_{1:K,1:D}, \boldsymbol{c})$ are the variational parameters for $\boldsymbol{\theta}$.

We can compute the ELBO using

$$Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})}\left[\sum_{n=1}^{N} \log p(\boldsymbol{x}_n|\boldsymbol{\theta})\right] - D_{\mathbb{KL}}\left(q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) \parallel p(\boldsymbol{\theta})\right) \tag{10.134}$$

We can approximate the first term using minibatching. Since $q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})$ is reparameterizable (see Section 10.3.3), we can sample from it and push gradients inside. If we use a single posterior sample per minibatch, $\boldsymbol{\theta}^s \sim q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})$, we get

$$\nabla_{\boldsymbol{\psi}_{\boldsymbol{\theta}}} Ł(\boldsymbol{\psi}_{\boldsymbol{\theta}}|\mathcal{D}) \approx \frac{N}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\psi}_{\boldsymbol{\theta}}} \log p(\boldsymbol{x}_b|\boldsymbol{\theta}^s) - \nabla_{\boldsymbol{\psi}_{\boldsymbol{\theta}}} D_{\mathbb{KL}}\left(q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}}) \parallel p(\boldsymbol{\theta})\right) \tag{10.135}$$

We can now optimize this with SGD.

Figure 10.10 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 10.10a 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 10.10b shows the predictions of the fitted model. More precisely, it shows $p(\boldsymbol{x}|\overline{\boldsymbol{\theta}})$, where $\overline{\boldsymbol{\theta}} = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\psi}_{\boldsymbol{\theta}})}[\boldsymbol{\theta}]$. Figure 10.10c shows a kernel density estimate fit to 10,000 samples from $q(\boldsymbol{\mu}_1|\boldsymbol{\psi}_{\boldsymbol{\theta}})$. 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 $\boldsymbol{\mu}_1^*$.

10.3.4 Gaussian VI

The most widely used RVI approximation is when $q_{\psi}(\boldsymbol{z})$ is a Gaussian, where $\boldsymbol{\psi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Following [TND21], we call this **Gaussian VI**. We give a briefy summary below. For a theoretical analysis of the computational / statistical tradeoff with using this form of posterior approximation, see [Bha+22].

10.3.4.1 Full-rank Gaussian VI

In this section, we represent the covariance using its Cholesky decomposition, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$, where $\mathbf{L} = \text{tril}(\boldsymbol{l})$ is a lower triangular matrix, as in [TLG14; TN18]. The variational parameters are $\boldsymbol{\psi} = (\boldsymbol{\mu}, \boldsymbol{l})$. The noise transformation has the form

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \iff \boldsymbol{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon} \tag{10.136}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, so $r(\boldsymbol{\psi}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$.

For any given sample $\boldsymbol{z}$, we can use automatic differentiation to compute the derivative of the sampled ELBO:

$$\nabla_{\boldsymbol{\psi}} \ell_{\boldsymbol{\psi}}(\boldsymbol{z}(\boldsymbol{\psi})) = \nabla_{\boldsymbol{\psi}} \log p(\boldsymbol{z}(\boldsymbol{\psi}), \mathcal{D}) - \nabla_{\boldsymbol{\psi}} \log q_{\boldsymbol{\psi}}(\boldsymbol{z}(\boldsymbol{\psi})) \tag{10.137}$$

10.3.4.2 Low-rank Gaussian VI

In high dimensions, an efficient alternative to using a Cholesky decomposition is the factor decomposition

$$\boldsymbol{\Sigma} = \mathbf{B}\mathbf{B}^{\mathsf{T}} + \mathbf{C}^2 \tag{10.138}$$

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** for Variational Approximation with Factor Covariance.

In the special case where $f = 1$, the covariance matrix becomes

$$\boldsymbol{\Sigma} = \boldsymbol{b}\boldsymbol{b}^{\mathsf{T}} + \text{diag}(\boldsymbol{c}^2) \tag{10.139}$$

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 particular, let $\boldsymbol{g} = (\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3)$ be the regular gradient of the ELBO wrt $\boldsymbol{\mu}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. Let $\boldsymbol{v}_1 = \boldsymbol{c}^2 - 2\boldsymbol{b}^2 \odot \boldsymbol{c}^{-4}$, $\boldsymbol{v}_2 = \boldsymbol{b}^2 \odot \boldsymbol{c}^{-3}$, $\kappa_1 = \sum_{i=1}^d b_i^2/c_i^2$, and $\kappa_2 = \frac{1}{2}(1 + \sum_{i=1}^d v_{2i}^2/v_{1i})^{-1}$. Then the natural gradient is given by

$$\boldsymbol{g}^{\text{nat}} = \begin{pmatrix} (\boldsymbol{g}_1^{\mathsf{T}}\boldsymbol{b})\boldsymbol{b} + \boldsymbol{c}^2 \odot \boldsymbol{g}_1 \\ \frac{1+\kappa_1}{2\kappa_1}((\boldsymbol{g}_2^{\mathsf{T}}\boldsymbol{b})\boldsymbol{b} + \boldsymbol{c}^2 \odot \boldsymbol{g}_2) \\ \frac{1}{2}\boldsymbol{v}_1^{-1} \odot \boldsymbol{g}_3 + \kappa_2[(\boldsymbol{v}_1^{-1} \odot \boldsymbol{v}_2)^{\mathsf{T}}\boldsymbol{g}_3](\boldsymbol{v}_1^{-1} \odot \boldsymbol{v}_2) \end{pmatrix} \tag{10.140}$$

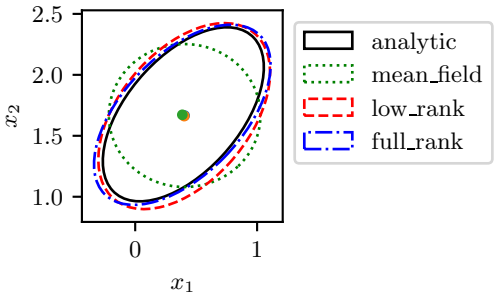


Figure 10.11: Gaussian variational approximation to a Gaussian posterior for the mean of a 2d Gaussian. Generated by gaussian_2d_vi.ipynb.

10.3.4.3 Example: GVI for a linear Gaussian system

As a sanity check, we try to approximate the posterior mean of a multivariate Gaussian with fixed covariance. We use a Gaussian prior, so the joint distribution has the form

$$p(\boldsymbol{z}|\mathcal{D}) \propto p(\boldsymbol{z}) \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{y}_n|\boldsymbol{z}, \boldsymbol{\Sigma}) \tag{10.141}$$

where the measurement noise $\boldsymbol{\Sigma}$ is a fixed, non-diagonal matrix. If we use a conjugate prior, $\mathcal{N}(\boldsymbol{z}|\ \breve{\boldsymbol{m}}, \breve{\mathbf{V}})$, then the exact posterior, $p(\boldsymbol{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{z}|\ \hat{\boldsymbol{m}}, \hat{\mathbf{V}})$, can be computed analytically, as discussed in Section 3.3.1.

We also compute a Gaussian variational approximation, $q(\boldsymbol{z}|\boldsymbol{\psi}_{\boldsymbol{\mu}}, \boldsymbol{\psi}_{\boldsymbol{\Sigma}})$, where $\boldsymbol{\psi}_{\boldsymbol{\Sigma}}$ is either full-rank, diagonal, or rank-1 plus diagonal. The results are shown in Figure 10.11. We see that the full-rank approximation matches the true posterior, as expected. However, the diagonal approximation is overconfident, which is a well-known flaw of variational inference.

10.3.5 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.

10.3.5.1 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 to some unconstrained space, with inverse $T^{-1} : \mathbb{R}^D \to \Theta$ that maps to the constrained domain of the prior. Let $\boldsymbol{z} = T(\boldsymbol{\theta})$ be the unconstrained latent variables. We will use a block-factored Gaussian variational approximation to the posterior for $\boldsymbol{z}$, i.e.,:

$$q(\boldsymbol{z}|\boldsymbol{\psi}) = \prod_{b=1}^{B} \mathcal{N}(\boldsymbol{z}_b|\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b) \tag{10.142}$$

where $\boldsymbol{\psi} = (\boldsymbol{\mu}_{1:B}, \boldsymbol{\Sigma}_{1:B})$.

By the change of variable formula Equation (2.286), we have

$$p(\boldsymbol{z}) = p(T^{-1}(\boldsymbol{z}))|\det(\mathbf{J}_{T^{-1}}(\boldsymbol{z}))| \tag{10.143}$$

where $\mathbf{J}_{T^{-1}}$ is the Jacobian of the inverse mapping $\boldsymbol{z} \to \boldsymbol{\theta}$. Hence the ELBO becomes

$$Ł(\boldsymbol{\psi}) = E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{\psi})}\left[p(\mathcal{D}|T^{-1}(\boldsymbol{z})) + \log p(T^{-1}(\boldsymbol{z})) + \log|\det(\mathbf{J}_{T^{-1}}(\boldsymbol{z}))|\right] + \mathbb{H}(\boldsymbol{\psi}) \tag{10.144}$$

We can use a Monte Carlo approximation of the expectation over $\boldsymbol{z}$, together with the reparameterization trick, which (in the fully diagonal case) replaces $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{\psi})$ with $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{\sigma} = \sigma_{1:D}$. We can compute the final entropy term in closed form. In the case of a fully diagonal approximation, it follows from Equation (5.96) that

$$\mathbb{H}(\boldsymbol{\psi}) = \sum_{i=1}^{D} 2\log(\sigma_i) + \text{const} \tag{10.145}$$

Since the objective is stochastic, 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.)

10.3.5.2 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 7.3 and show that the Gaussian fit is a good approximation, despite the skewed nature of the posterior.

10.3.5.3 Example: ADVI for GMMs

In this section, we use ADVI to approximate the posterior of the parametrs of a mixture of Gaussians. The difference from the VBEM algorithm of Section 10.2.6 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

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\left[\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{y}_n|\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\Sigma}_k))\right] \tag{10.146}$$

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]

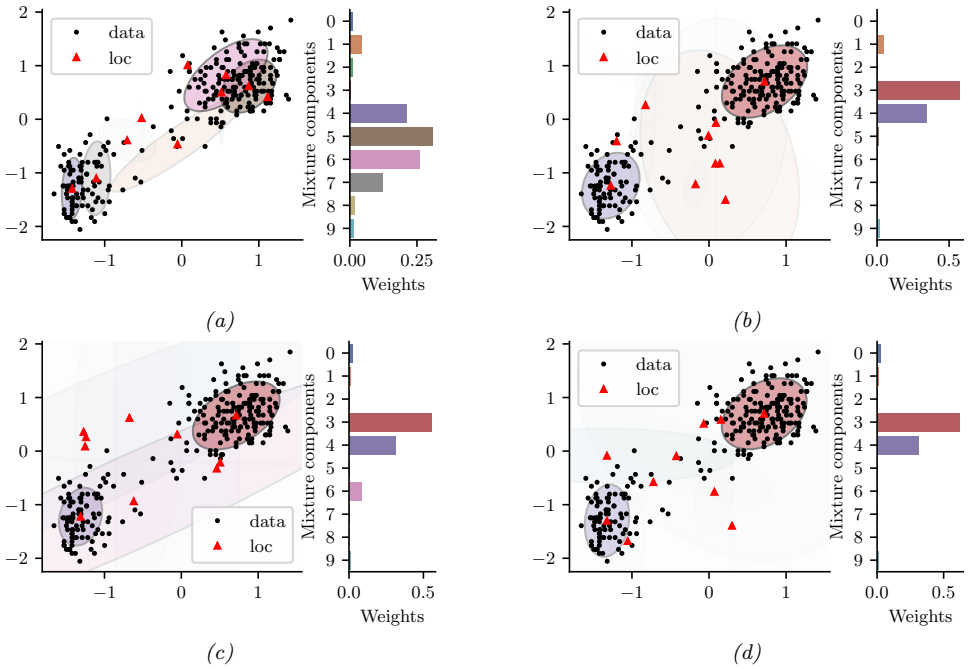


Figure 10.12: Posterior over the mixing weights (histogram) and the means and covariances of each Gaussian mixture component, using $K = 10$, when fitting the model to the Old Faithful dataset from Figure 10.7. (a) MAP approximation. (b-d) 3 samples from the Gaussian approximation. The intensity of the shading is proportional to the mixture weight. Generated by gmm_advi_bijax.ipynb.

for a definition of the model in STAN syntax.) The posterior approximation for the unconstrained parameters is a block-diagonal gaussian. $q(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\psi}_{\boldsymbol{\mu}}, \boldsymbol{\psi}_{\boldsymbol{\Sigma}})$, where the unconstrained parameters are computed using suitable bijections (see code for details).

We apply this method to the Old Faithful dataset from Figure 10.7, using $K = 10$ mixture components. The results are shown in Figure 10.12. 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).

10.3.5.4 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].

10.3.6 Amortized inference

Suppose we want to perform parameter estimation in a model with local latent variables:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \sum_{\boldsymbol{z}_n} p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n) \tag{10.147}$$

To compute the marginal likelihood, we first need to compute the posteriors $p_{\boldsymbol{\theta}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$ for each example n; the normalization constant then gives us $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n)$. When using BBVI or RVI for this, we therefore have to solve an optimization problem for each $q(\boldsymbol{z}_n|\boldsymbol{\psi}_n)$, which can be 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 $\boldsymbol{\psi}_n = f^{\text{inf}}_{\boldsymbol{\phi}}(\boldsymbol{x}_n)$. This technique is known as **amortized 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

$$q(\boldsymbol{z}_n|\boldsymbol{\psi}_n) = q(\boldsymbol{z}_n|f^{\text{inf}}_{\boldsymbol{\phi}}(\boldsymbol{x}_n)) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n) \tag{10.148}$$

The "**amortized ELBO**", for a model with local latents and fixed global parameters, becomes

$$Ł(\boldsymbol{\phi}, \boldsymbol{\theta}|\mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_n)\right]\right] \tag{10.149}$$

We can approximate this by sampling a single data point $\boldsymbol{x}_n \sim p_{\mathcal{D}}$, and then sampling a single latent $\boldsymbol{z}_n \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$, as in DSVI, to get

$$Ł(\boldsymbol{\phi}, \boldsymbol{\theta}|\boldsymbol{x}_n, \boldsymbol{z}_n) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{z}_n^s) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_n) \tag{10.150}$$

(We call this the "**per-sample ELBO**", although [Ble17] call it the **instantaneous ELBO**.) If the posteriors are reparameterizable, we can push gradients inside and then apply SGD. See Algorithm 23 for the resulting pseudocode.

Algorithm 23: Amortized SVI

```
Initialize θ, φ
repeat
    Sample x_n ~ p_D
    Sample z_n ~ q_φ(z|x_n)
    θ := θ + η ∇_θ L̃(φ, θ|x_n, z_n)
    φ := φ + η ∇_φ L̃(φ, θ|x_n, z_n)
    Update learning rate η
until converged
```

This method is very 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]. The key insight is that the local SVI procedure is itself differentiable, so the inference network and generative model can be trained end-to-end. (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).

10.4 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.

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.,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{a}) q_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{x}) d\boldsymbol{a} \neq \prod_k q_{\boldsymbol{\phi}}(z_k|\boldsymbol{x}) \tag{10.151}$$

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

$$\nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \nabla_{\boldsymbol{z}} \log \left[p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] = \nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) \tag{10.152}$$

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

$$\hat{p}_S(\boldsymbol{x}|\boldsymbol{z}_{1:S}) \triangleq \frac{1}{S}\sum_{k=1}^{S}\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})} = \frac{1}{S}\sum_{k=1}^{S} w_s \tag{10.153}$$

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 $q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x}) = \prod_{k=1}^S q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})$. In addition, since the estimator is always positive, we can take logarithms, and thus obtain a stochastic lower bound on the log likelihood:

$$Ł_S(\boldsymbol{\phi},\boldsymbol{\theta}|\boldsymbol{x}) \triangleq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\log\left(\frac{1}{S}\sum_{s=1}^{S} w_s\right)\right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\log \hat{p}_S(\boldsymbol{z}_{1:S})\right] \tag{10.154}$$

$$\leq \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:S}|\boldsymbol{x})}\left[\hat{p}_S(\boldsymbol{z}_{1:S})\right] = \log p(\boldsymbol{x}) \tag{10.155}$$

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]. If $S = 1$, $Ł_S$ reduces to the standard ELBO:

$$Ł_1(\boldsymbol{\phi},\boldsymbol{\theta}|\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log w\right] = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \tag{10.156}$$

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

$$Ł(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{\mathbb{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.157}$$

By analogy, we can define the **evidence upper bound** or **EUBO** as follows:

$$\text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) + D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right) \geq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{10.158}$$

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 (10.173), to get

$$\nabla_{\boldsymbol{\phi}} \text{EUBO}(\boldsymbol{\theta}, \boldsymbol{\phi}|\boldsymbol{x}) = \sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}^s|\boldsymbol{x}) \tag{10.159}$$

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 reparamterization trick; thus we have to use higher variance estimators, such as REINFORCE (see Section 10.3.2). Second, even in the case where we can use the reparamterization 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:

$$\mathcal{L}(\boldsymbol{\theta}) = D_{\mathbb{KL}}\left(p_{\mathcal{D}}(\boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x})\right) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] + \text{const} \tag{10.160}$$

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:

$$\ell(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \tag{10.161}$$

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:

$$\ell(\boldsymbol{\theta}|\boldsymbol{\phi}, \boldsymbol{x}) = \log\left(\frac{1}{S}\sum_{s=1}^{S} w_s\right) \tag{10.162}$$

where $w_s = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})}$.

We now discuss how to compute the gradient of this objective. Using the log-derivative trick, we have that

$$\nabla_{\boldsymbol{\theta}} \log w_s = \frac{1}{w_s}\nabla_{\boldsymbol{\theta}} w_s = \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_s) \tag{10.163}$$

Hence

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}|\boldsymbol{\phi}, \boldsymbol{x}) = \frac{1}{\frac{1}{S}\sum_{s=1}^{S} w_s} \left(\frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\theta}} w_s \right) \tag{10.164}$$

$$= \frac{1}{\sum_{s=1}^{S} w_s} \left(\sum_{s=1}^{S} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_s, \boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_s, \boldsymbol{x}) \right) \tag{10.165}$$

$$= \sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_s, \boldsymbol{x}) \tag{10.166}$$

where $\overline{w}_s = w_s / (\sum_{s'=1}^{S} w_{s'})$.

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:

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \left[D_{\mathbb{KL}} \left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right) \right] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})} \left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] + \text{const} \tag{10.167}$$

Equivalently, we can maximize the following loglikelihood objective:

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{z},\boldsymbol{x}) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})} \left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \right] \tag{10.168}$$

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

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s) \tag{10.169}$$

where $(\boldsymbol{z}'_s, \boldsymbol{x}'_s) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$. The gradient of this is given by

$$\nabla_{\boldsymbol{\phi}} \ell(\boldsymbol{\phi}|\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s) \tag{10.170}$$

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.[4] [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

$$\mathcal{L}(\boldsymbol{\phi}|\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathcal{D}}(\boldsymbol{x})}\left[D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \parallel q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right)\right] \tag{10.171}$$

We can compute a single sample approximation to the negative of the above expression as follows:

$$\ell(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] \tag{10.172}$$

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:

$$\nabla_{\boldsymbol{\phi}} \ell(\boldsymbol{\phi}|\boldsymbol{\theta}, \boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} \approx \sum_{s=1}^{S} \overline{w}_s \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}_s|\boldsymbol{x}) \tag{10.173}$$

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 (10.173) is very similar to Equation (10.170). 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 data point, 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 data point.

10.6.4 Summary of algorithm

Algorithm 24: 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 $\frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}'_s|\boldsymbol{x}'_s)$

We summarize the RWS algorithm in Algorithm 24. 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

4. We thank Rif Saurous for suggesting this term.

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 $q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$:

$$\text{wake phase } \min_{\boldsymbol{\theta}} D_{\mathbb{KL}}\left(q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right) \tag{10.174}$$

$$\text{daydream phase } \min_{\boldsymbol{\phi}} D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \parallel q_{\mathcal{D},\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})\right) \tag{10.175}$$

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 5.1.3.3. 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 5.1.3.3. 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:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z_p}\hat{p}(\boldsymbol{\theta}), \;\; \hat{p}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^{K} f_k(\boldsymbol{\theta}) \tag{10.176}$$

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:

$$q(\boldsymbol{\theta}) = \frac{1}{Z_q}\hat{q}(\boldsymbol{\theta}), \;\; \hat{q}(\boldsymbol{\theta}) = p_0(\boldsymbol{\theta}) \prod_{k=1}^{K} \tilde{f}_k(\boldsymbol{\theta}) \tag{10.177}$$

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:

$$q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta})}{\tilde{f}_i(\boldsymbol{\theta})} \propto \prod_{k \neq i} \tilde{f}_k(\boldsymbol{\theta}) \tag{10.178}$$

This division operation can be implemented by subtracting the natural parameters, as explained in Section 2.2.7.2. 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:

$$q_i^{\text{tilted}}(\boldsymbol{\theta}) = \frac{1}{Z_i} f_i(\boldsymbol{\theta}) q_{-i}^{\text{cavity}}(\boldsymbol{\theta}) \tag{10.179}$$

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:

$$q_i^{\text{proj}}(\boldsymbol{\theta}) = \text{proj}(q_i^{\text{tilted}}) \triangleq \underset{\tilde{q} \in \mathcal{Q}}{\text{argmin}}\, D(q_i^{\text{tilted}} || \tilde{q}) \tag{10.180}$$

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 5.1.3.4. 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

$$\log q_i^{\text{tilted}}(\boldsymbol{\theta}) = \alpha \log f_i(\boldsymbol{\theta}) - \frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}} \mathbf{Q}_{-i} \boldsymbol{\theta} + \boldsymbol{r}_{-i}^{\mathsf{T}} \boldsymbol{\theta} + \text{const} \tag{10.181}$$

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:

$$\mathbf{Q}_{\setminus i} = -\nabla^2_{\boldsymbol{\theta}} \log q_i^{\text{tilted}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, \;\; \boldsymbol{r}_{\setminus i} = \mathbf{Q}_{\setminus i} \hat{\boldsymbol{\theta}} \tag{10.182}$$

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:

$$\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = \frac{q_i^{\text{proj}}(\boldsymbol{\theta})}{q_{-i}^{\text{cavity}}(\boldsymbol{\theta})} \tag{10.183}$$

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 10.13 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.

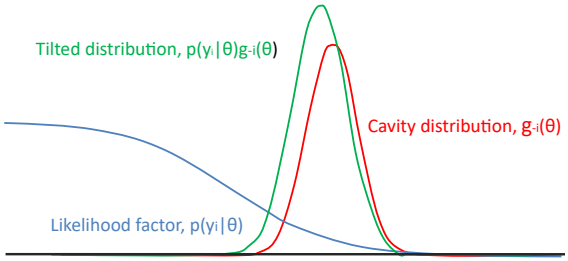


Figure 10.13: Combining a logistic likelihood factor $f_i = p(y_i|\boldsymbol{\theta})$ with the cavity prior, $q_{-i}^{cavity} = g_{-i}(\boldsymbol{\theta})$, to get the tilted distribution, $q_i^{tilted} = p(y_i|\boldsymbol{\theta})g_{-i}(\boldsymbol{\theta})$. Adapted from Figure 2 of [Gel+14b].

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 8.10. 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 $q_{t+1} = \text{argmin}_{q\in\mathcal{Q}} D_{\mathbb{KL}}\left(\hat{p}_{t+1} \parallel q\right)$. 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:

$$\tilde{f}_i^{\text{new}}(\boldsymbol{\theta}) = \left(\tilde{f}_i(\boldsymbol{\theta})\right)^{1-\delta}\left(\frac{q_i^{\text{proj}}(\boldsymbol{\theta})}{q_{-i}^{\text{cavity}}}\right)^{\delta} \tag{10.184}$$

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 2.19. 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.,

$$\prod_{n=1}^{N} f_n(\boldsymbol{\theta}) \approx \tilde{f}(\boldsymbol{\theta})^N \tag{10.185}$$

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

$$q_{-1}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})/\tilde{f}(\boldsymbol{\theta}) \tag{10.186}$$

We then compute the tilted distribution

$$q_{\backslash n}(\boldsymbol{\theta}) \propto f_n(\boldsymbol{\theta}) q_{-1}(\boldsymbol{\theta}) \tag{10.187}$$

Next we derive the new local factor for this datapoint using moment matching:

$$\tilde{f}_n(\boldsymbol{\theta}) = \text{proj}(q_{\backslash n}(\boldsymbol{\theta}))/q_{-1}(\boldsymbol{\theta}) \tag{10.188}$$

Finall we perform a damped update of the average likelihood $\tilde{f}(\boldsymbol{\theta})$ using this new local factor:

$$\tilde{f}_{\text{new}}(\boldsymbol{\theta}) = \tilde{f}_{\text{old}}(\boldsymbol{\theta})^{1-1/N} \tilde{f}_n(\boldsymbol{\theta})^{1/N} \tag{10.189}$$

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 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 inference

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) compute a grid, and then evaluate 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].

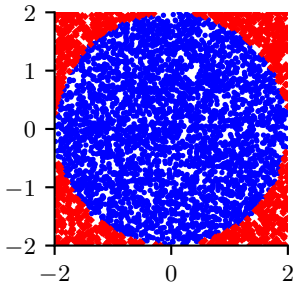


Figure 11.1: Estimating π by Monte Carlo integration using 5000 samples. Blue points are inside the circle, red points are outside. Generated by mc_estimate_pi.ipynb.

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,

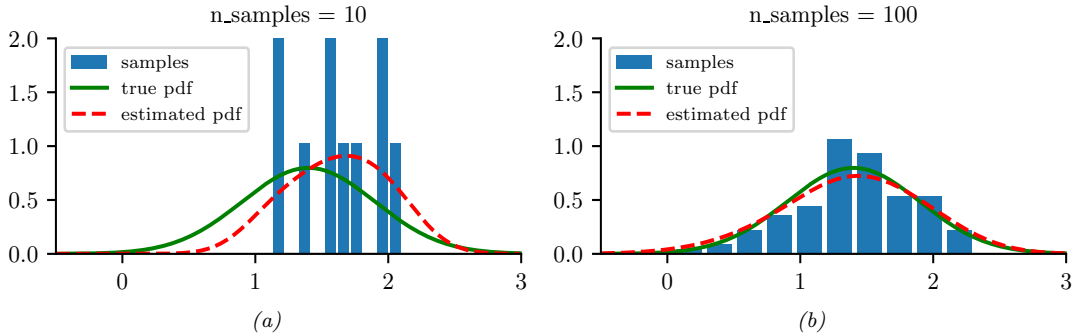


Figure 11.2: 10 and 100 samples from a Gaussian distribution, $\mathcal{N}(\mu = 1.5, \sigma^2 = 0.25)$. A dotted red line denotes kernel density estimate derived from the samples. Generated by mc_accuracy_demo.ipynb.

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.

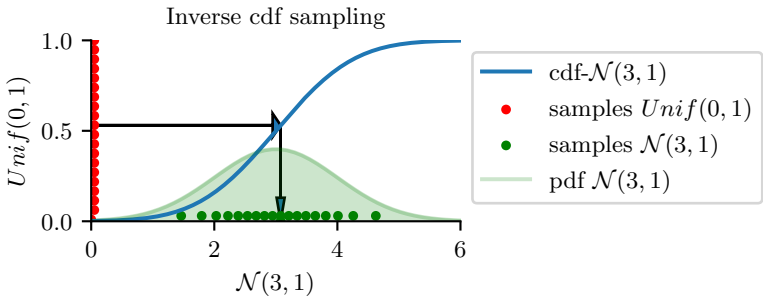


Figure 11.3: Sampling from $\mathcal{N}(3,1)$ using an inverse CDF.

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 **pseudo random 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

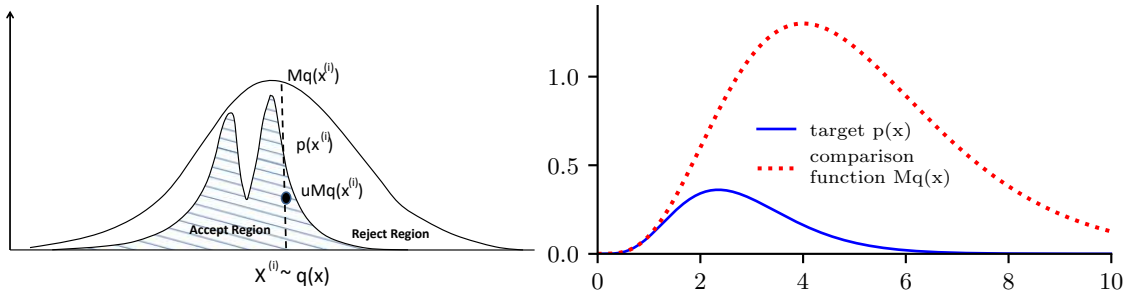
Suppose we want to sample from the **target distribution**

$$p(\boldsymbol{x}) = \tilde{p}(\boldsymbol{x})/Z_p \tag{11.19}$$

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.



(a) *(b)*

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

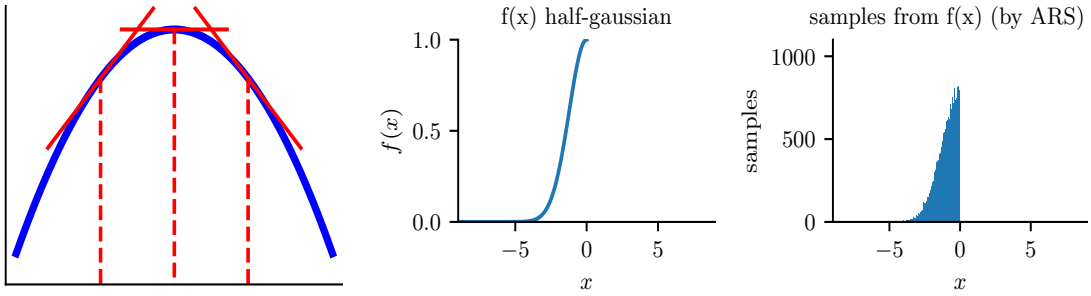
We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to any log concave 1d density $p(x)$. The idea is to upper bound the log density with a piecewise linear function, as illustrated in Figure 11.5(a). We choose the initial locations for the pieces based on a fixed 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 10.1.2). 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

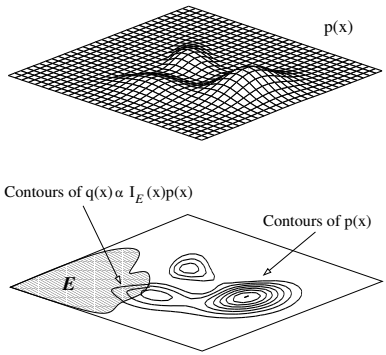


Figure 11.6: In importance sampling, we should sample from a distribution that takes into account regions where $\pi(\boldsymbol{x})$ has high probability and *where $\varphi(\boldsymbol{x})$ is large. Here the function to be evaluated is an indicator function of a set, corresponding to a set of rare 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.

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-Blackwellisation

In this section, we discuss a useful technique for reducing the variance of MC estimators known as **Rao-Blackwellisation**. 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-Blackwellised 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 6.5.3.2, 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 10.3.2 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.

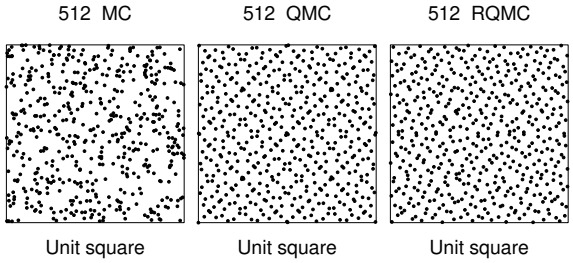


Figure 11.7: Illustration of Monte Carlo (MC), Quasi MC (QMC) from a Sobol sequence, and Randomized QMC using a scrambling method. Adapted from Figure 1 of [OR20]. Used with kind permission of Art Owen.

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 quadature, 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 6.8).

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 inference

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 **Metropolis 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 25.

Algorithm 25: 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 analyse 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:

$$q(\boldsymbol{x}'|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}'|\boldsymbol{x}, \tau^2\mathbf{I}) \tag{12.15}$$

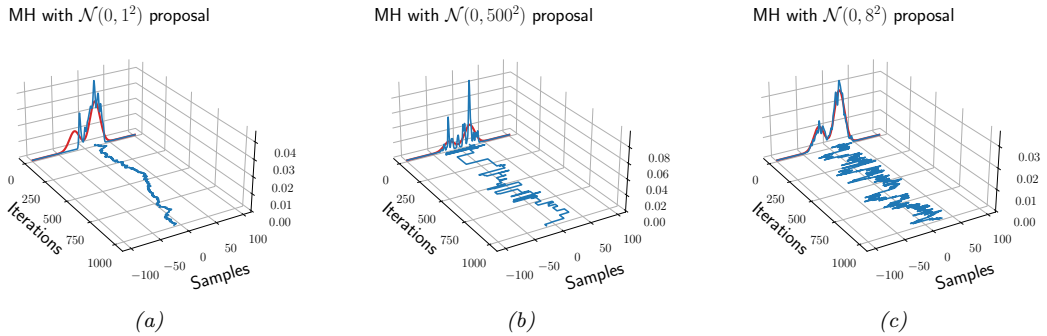


Figure 12.1: An example of the Metropolis Hastings algorithm for sampling from a mixture of two 1D Gaussians ($\boldsymbol{\mu} = (-20, 20)$, $\boldsymbol{\pi} = (0.3, 0.7)$, $\boldsymbol{\Sigma} = (100, 100)$), using a Gaussian proposal with standard deviation of $\tau \in \{1, 8, 500\}$. (a) When $\tau = 1$, 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 (10.28).)

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.

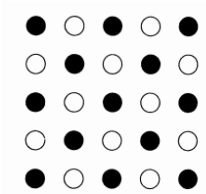


Figure 12.2: Illustration of checkerboard pattern for a 2d MRF. This allows for parallel updates.

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

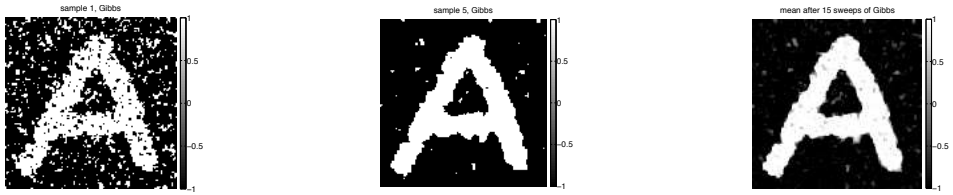
$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_{i \sim j} \psi_{ij}(x_i, x_j|\boldsymbol{\theta}) \tag{12.23}$$

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:

$$p(x_i|\boldsymbol{x}_{-i}) \propto \prod_{j \in \text{nbr}(i)} \psi_{ij}(x_i, x_j) \tag{12.24}$$

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 10.3 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 10.3.

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

$$p(z = k, \boldsymbol{x}|\boldsymbol{\theta}) = \pi_k\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{12.33}$$

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha})\prod_{k=1}^{K}\mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{m}_0, \mathbf{V}_0)\text{IW}(\boldsymbol{\Sigma}_k, \mathbf{S}_0, \nu_0) \tag{12.34}$$

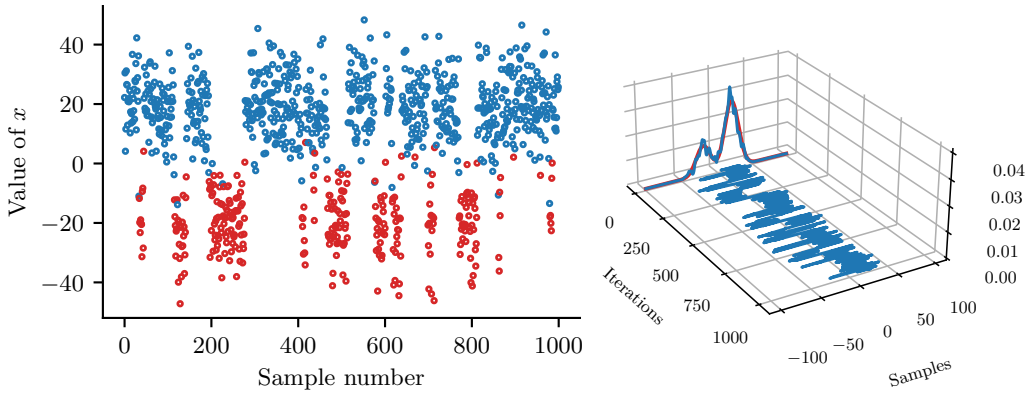
12.3.5.1 Known parameters

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 $p(\mathcal{D}|\boldsymbol{\theta})$.



(a) (b)

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 3.2.2)

$$p(\boldsymbol{\pi}|\boldsymbol{z}) = \text{Dir}(\{\alpha_k + \sum_{i=1}^{N} \mathbb{I}(z_i = k)\}_{k=1}^{K}) \tag{12.40}$$

For the means, we have (using results from Section 3.3.1)

$$p(\boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k, \boldsymbol{z}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{m}_k, \mathbf{V}_k) \tag{12.41}$$

$$\mathbf{V}_k^{-1} = \mathbf{V}_0^{-1} + N_k \boldsymbol{\Sigma}_k^{-1} \tag{12.42}$$

$$\boldsymbol{m}_k = \mathbf{V}_k(\boldsymbol{\Sigma}_k^{-1} N_k \overline{\boldsymbol{x}}_k + \mathbf{V}_0^{-1}\boldsymbol{m}_0) \tag{12.43}$$

$$N_k \triangleq \sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right) \tag{12.44}$$

$$\overline{\boldsymbol{x}}_k \triangleq \frac{\sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right) \boldsymbol{x}_i}{N_k} \tag{12.45}$$

For the covariances, we have (using results from Section 3.3.2)

$$p(\boldsymbol{\Sigma}_k|\boldsymbol{\mu}_k, \boldsymbol{z}, \boldsymbol{x}) = \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_k, \nu_k) \tag{12.46}$$

$$\mathbf{S}_k = \mathbf{S}_0 + \sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^{\mathsf{T}} \tag{12.47}$$

$$\nu_k = \nu_0 + N_k \tag{12.48}$$

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.

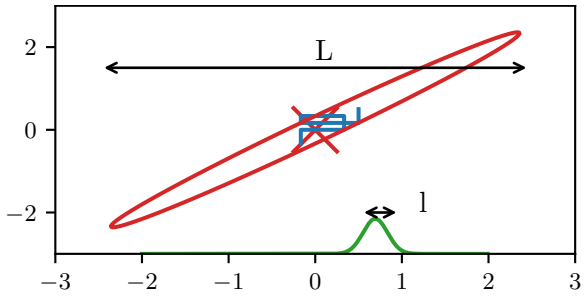


Figure 12.5: Illustration of potentially slow sampling when using Gibbs sampling for a skewed 2D Gaussian. Adapted from Figure 11.11 of [Bis06]. Generated by gibbs_gauss_demo.ipynb.

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 the forwards-filtering backwards-sampling (Section 8.2.8).

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 explai below. In particular, the full conditional for the latent indicators is given by

$$\begin{aligned} p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}, \cancel{\boldsymbol{\beta}}) p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \cancel{\boldsymbol{\alpha}}, \boldsymbol{\beta}) && (12.51) \\ &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) \\ & p(\boldsymbol{x}_{-i}|\cancel{z_i = k}, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) && (12.52) \\ &\propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\beta}) && (12.53) \end{aligned}$$

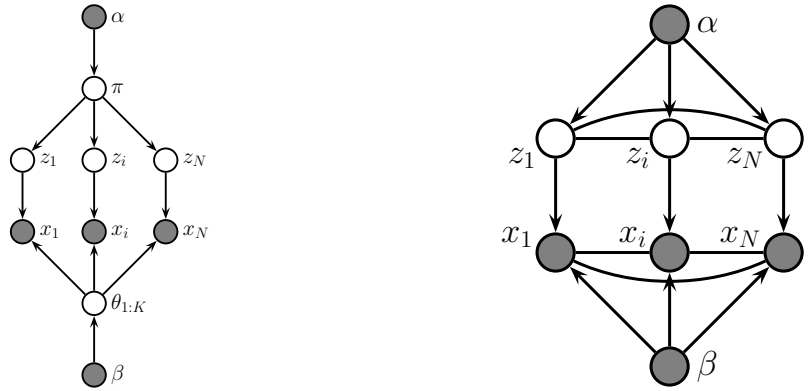


Figure 12.6: (a) A mixture model represented as an "unrolled" DPGM. (b) After integrating out the continuous latent parameters.

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 (3.29), 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 26, 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 26: 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 Section 31.3.2.

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].

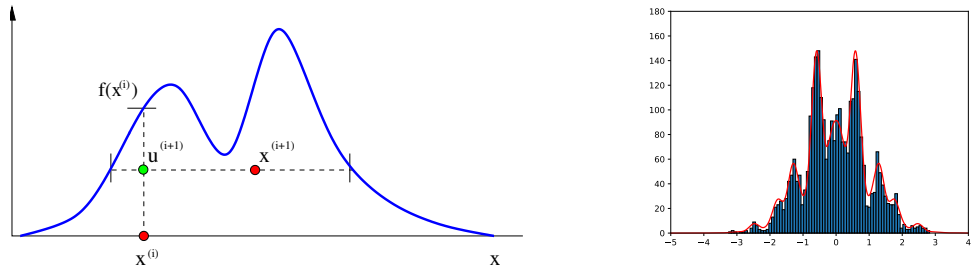


Figure 12.7: Slice sampling. (a) Illustration of one step of the algorithm in 1d. Given a previous sample x^i, we sample u^{i+1} uniformly on $[0, f(x^i)]$, where $f = \tilde{p}$ is the (unnormalized) target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. From Figure 15 of [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.

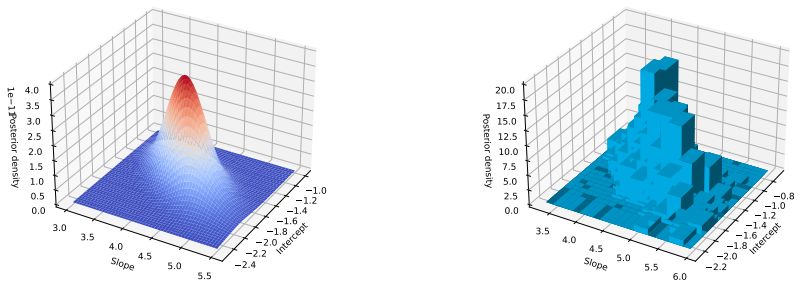


Figure 12.8: Posterior for binomial regression for 1d data. Left: Slice sampling approximation. Right: Grid approximation. Generated by slice_sampling_demo_2d.ipynb.

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

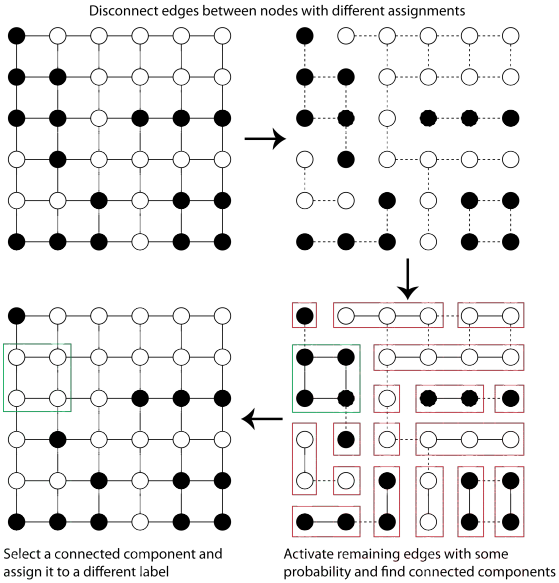


Figure 12.9: Illustration of the Swendsen Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

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**m 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 27 for the pseudocode.[6]

Algorithm 27: 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 **warmup** 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 **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 2.4), 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 **Metropolis Adjusted Langevin Algorithm** (**MALA**) [RT96]. This gives rise to the simplified algorithm shown in Algorithm 28.

Algorithm 28: 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 $\mathbf{\Sigma} = \mathbf{I}$, 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 **Unadjusted 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

$$\mathbb{V}\left[\nabla_B \mathcal{L}(\boldsymbol{\theta})\right] = \frac{1}{B^2} \sum_{j=1}^{B} \mathbb{V}\left[\nabla \mathcal{L}_{i_j}(\boldsymbol{\theta})\right] \tag{12.89}$$

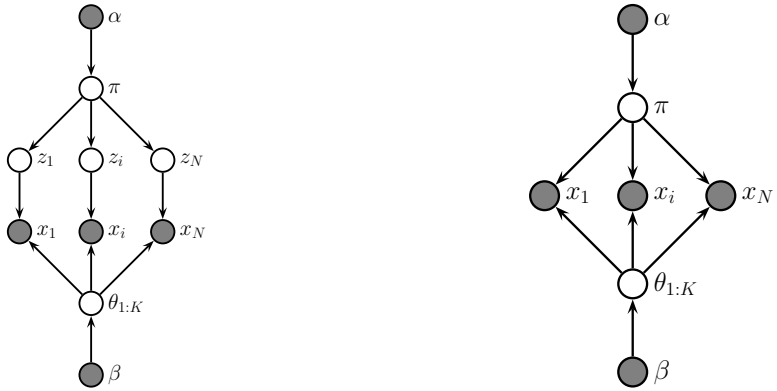


Figure 12.10: (a) A mixture model. (b) After integrating out the discrete latent variables.

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 Lipshitz 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 8.2.2) 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.

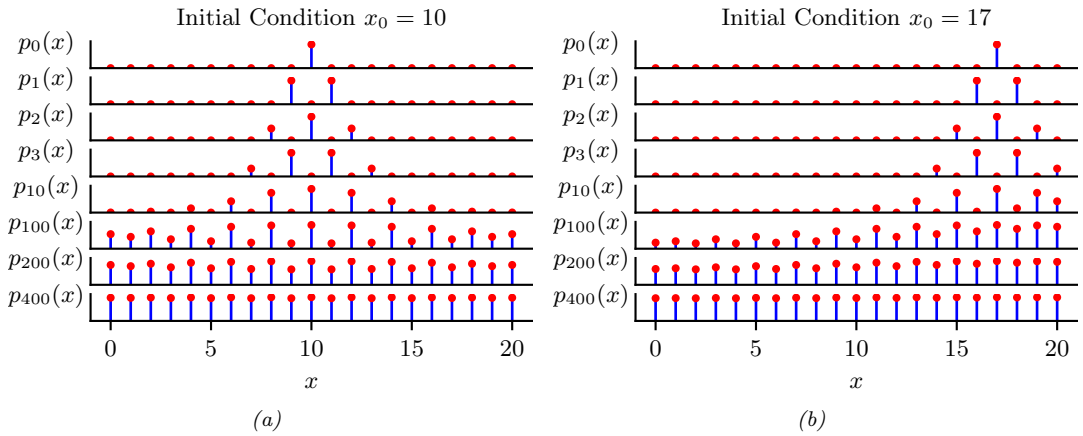


Figure 12.11: Illustration of convergence to the uniform distribution over $\{0, 1, \ldots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Adapted from Figures 29.14 and 29.15 of [Mac03]. Generated by random_walk_integers.ipynb.

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:

$$\phi \triangleq \min_{S: 0 \leq p^*(S) \leq 0.5} \frac{\sum_{x \in S, x' \in S^c} T(x \to x')}{p^*(S)}, \tag{12.96}$$

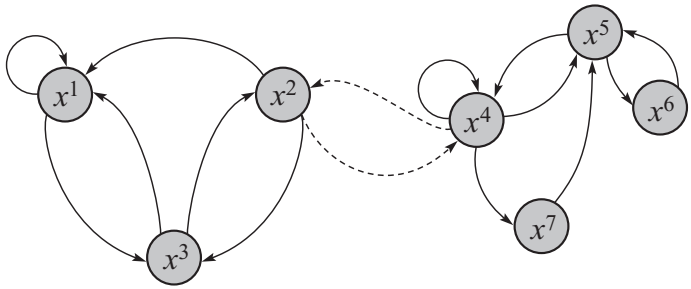


Figure 12.12: A Markov chain with low conductance. The dotted arcs represent transitions with very low probability. From Figure 12.6 of [KF09a]. Used with kind permission of Daphne Koller.

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 data points. 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

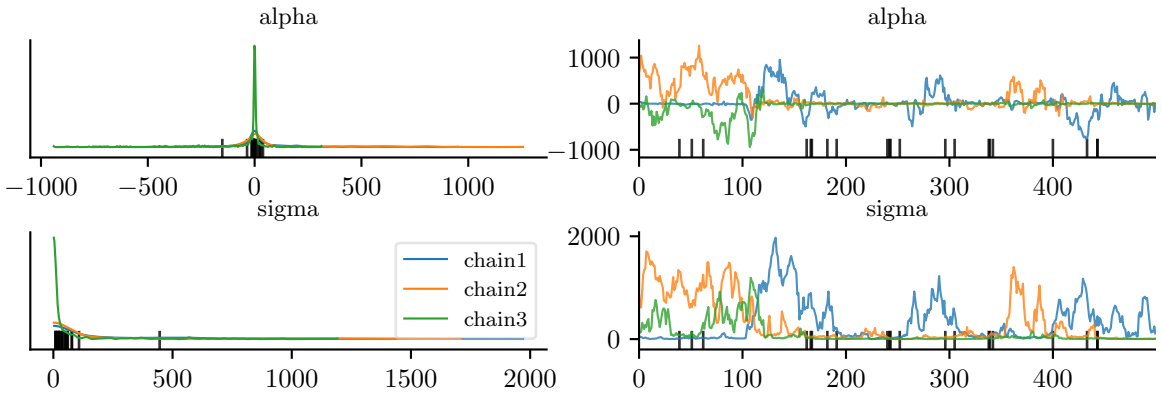


Figure 12.13: Marginals (left) and trace plot (right) for the univariate Gaussian using the diffuse prior. Black vertical lines indicate HMC divergences. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

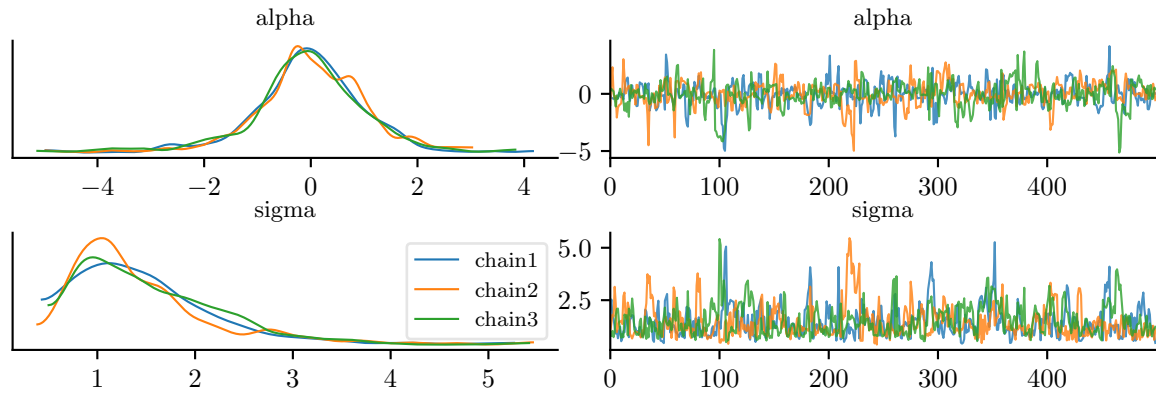


Figure 12.14: Marginals (left) and trace plot (right) for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

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

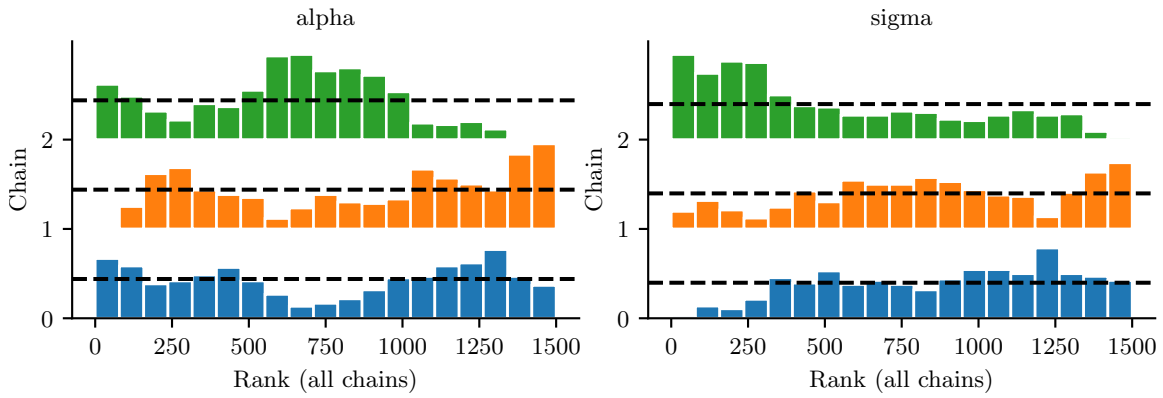


Figure 12.15: Trace rank plot for the univariate Gaussian using the diffuse prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

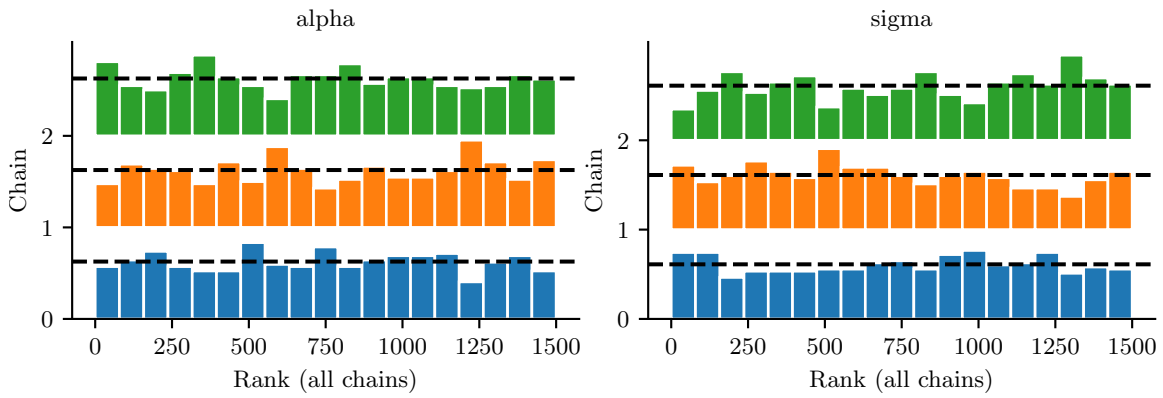


Figure 12.16: Trace rank plot for the univariate Gaussian using the sensible prior. Adapted from Figures 9.9–9.10 of [McE20]. Generated by mcmc_traceplots_unigauss.ipynb.

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.

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.

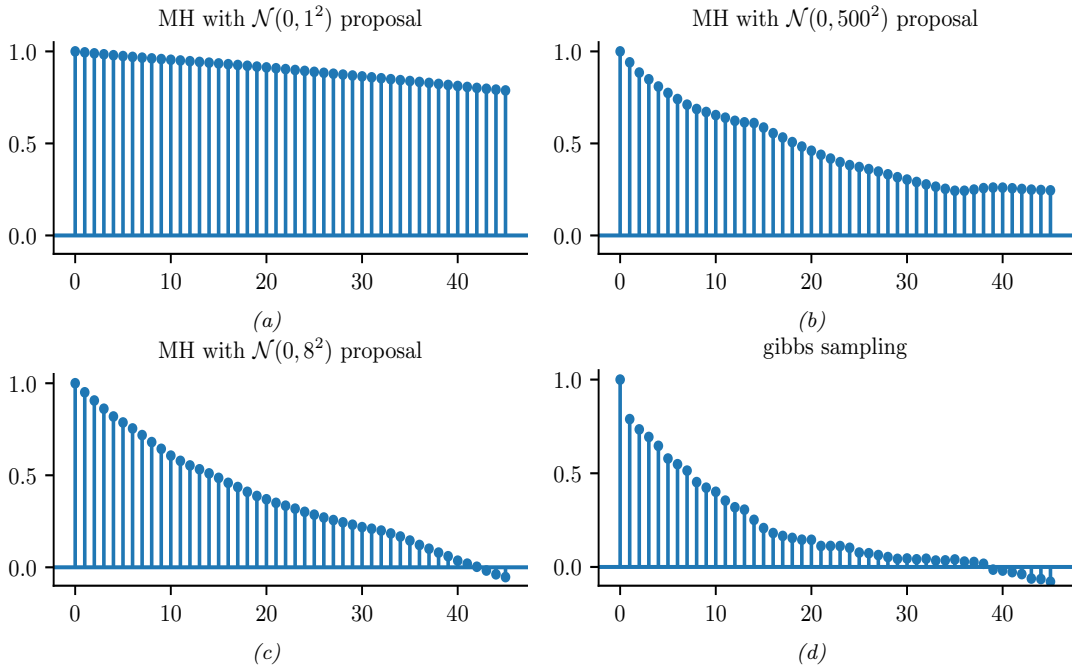


Figure 12.17: Autocorrelation functions for various MCMC samplers for the mixture of two 1D Gaussians. (a-c) These are the MH samplers in Figure 12.1. (d) This is the Gibbs sampler in Figure 12.4. Generated by mcmc_gmm_demo.ipynb.

Recall that for N (not necessarily independent) random variables we have

$$\mathbb{V}\left[\sum_{n=1}^{N} x_n\right] = \sum_{i=1}^{N}\sum_{j=1}^{N} \text{Cov}\left[x_i, x_j\right] = \sum_{i=1}^{N} \mathbb{V}\left[x_i\right] + 2 \sum_{1 \le i < j \le N} \text{Cov}\left[x_i, x_j\right] \tag{12.102}$$

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

$$\mathbb{V}\left[\overline{x}\right] = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \text{Cov}\left[x_i, x_j\right] \tag{12.103}$$

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

$$\mathbb{V}[\overline{x}] = \frac{\sigma^2}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \text{corr}[x_i, x_j] \tag{12.105}$$

For a fixed i, we can think of $\text{corr}[x_i, x_j]$ 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

$$\sum_{j=1}^{N} \text{corr}[x_i, x_j] \to \sum_{\ell=-\infty}^{\infty} \text{corr}[x_i, x_{i+\ell}] = 1 + 2\sum_{\ell=1}^{\infty} \text{corr}[x_i, x_{i+\ell}] \tag{12.106}$$

since $\text{corr}[x_i, x_i] = 1$ and $\text{corr}[x_i, x_{i-\ll}] = \text{corr}[x_i, x_{i+\ell}]$ for lag $\ell > 0$. Since we assume the samples are coming from a stationary distribution, the index i does not matter. This 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

$$\rho(\ell) \triangleq \text{corr}[x_0, x_\ell] \tag{12.108}$$

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: $\mathbb{V}[\overline{x}] = \frac{\sigma^2}{N^2} \sum_{i=1}^{N} \rho = \frac{\sigma^2}{N}\rho$. By contrast, the variance of the estimate from independent samples is $\mathbb{V}[\hat{x}] = \frac{\sigma^2}{N}$. 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:

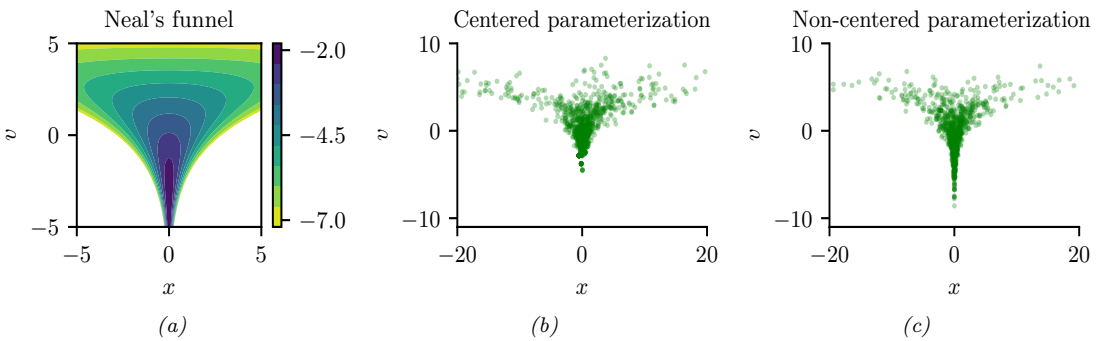


Figure 12.18: Neal's funnel. (a) Joint density. (b) HMC samples from centered representation. (c) HMC samples from non-centered representation. Generated by neals_funnel.ipynb.

- 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-Blackwellisation, 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 Section 3.7.2.2 and the hierarchical radon regression example in Section 15.5.3.2. 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 29 for the pseudo-code.

Algorithm 29: 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 $\mathcal{B}_t \in \{1, \ldots, N\}$
  $\boldsymbol{g}_t = \tilde{\boldsymbol{g}} + \frac{N}{B} \sum_{n \in \mathcal{B}_t} \left(\nabla \mathcal{E}_n(\boldsymbol{\theta}_t) - \nabla \mathcal{E}_n(\tilde{\boldsymbol{\theta}}) \right)$
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \boldsymbol{g}_t + \sqrt{2\eta_t} \mathcal{N}(\mathbf{0}, \mathbf{I})$

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 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.

12.8 Reversible jump (trans-dimensional) 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 [Gre98] 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

$$f_{m \to n}(f_{n \to m}(\boldsymbol{x}_n, \boldsymbol{u}_{n,m})) = (\boldsymbol{x}_n, \boldsymbol{u}_{n,m}) \tag{12.135}$$

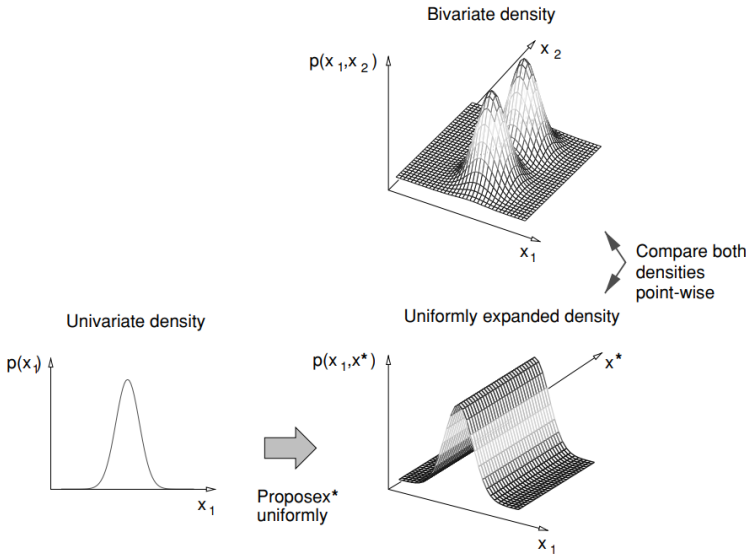


Figure 12.19: To compare a 1d model against a 2d model, we first have to map the 1d model to 2d space so the two have a common measure. Note that we assume the ridge has finite support, so it is integrable. From Figure 17 of [And+03]. Used with kind permission of Nando de Freitas.

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.

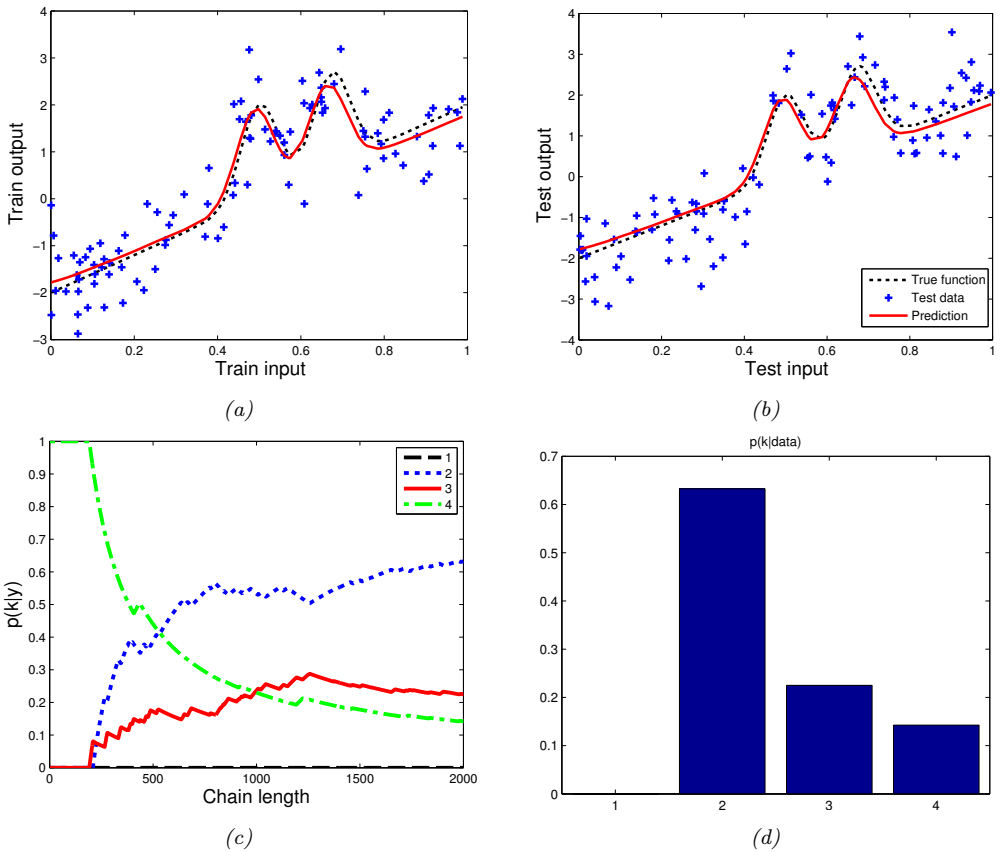


Figure 12.20: Fitting an RBF network to some 1d data using RJMCMC. (a) Prediction on train set. (b) Prediction on test set. (c) Plot of $p(k|\mathcal{D})$ vs iteration. (d) Final posterior $p(k|\mathcal{D})$. Adapted from Figure 4 of [AFD01]. Generated by rjmcmc_rbf, written by Nando de Freitas.

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

$$\mu = \frac{\mu_1 + \mu_2}{2} \tag{12.140}$$

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 $|\det \mathbf{J}_{merge}| = 1/(2\beta)$.

Algorithm 30: Generic reversible jump MCMC (single step)

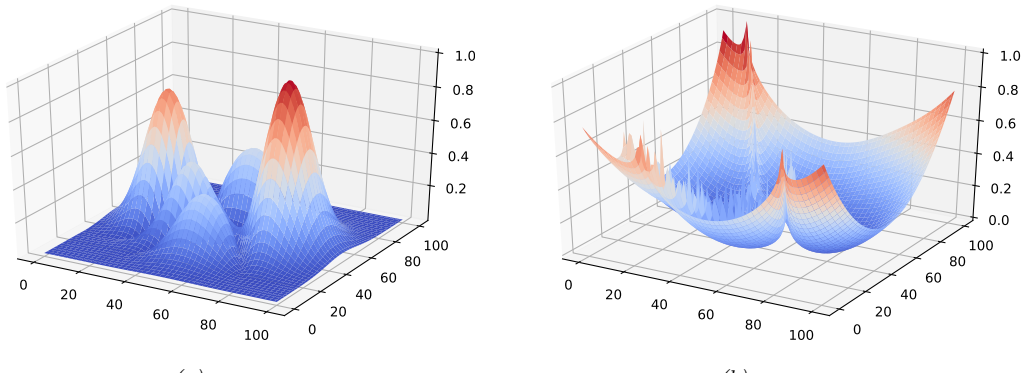
1 Sample $u \sim U(0, 1)$
2 If $u \leq b_k$
3 then birth move
4 else if $u \leq (b_k + d_k)$ then death move
5 else if $u \leq (b_k + d_k + s_k)$ then split move
6 else if $u \leq (b_k + d_k + s_k + m_k)$ then merge move
7 else update parameters

The overall pseudo-code for the algorithm, assuming the current model has index k, is given in Algorithm 30. Here b_k is the probability of a birth move, d_k is the probability of a death move, s_k 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.



(a) *(b)*

Figure 12.21: (a) A peaky distribution. (b) Corresponding energy function. Generated by simulated_annealing_2d_demo.ipynb.

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 6.9.)

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 worse or better the quality. We can apply this approach to probability distributions, to control the number of modes (low energy states) 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

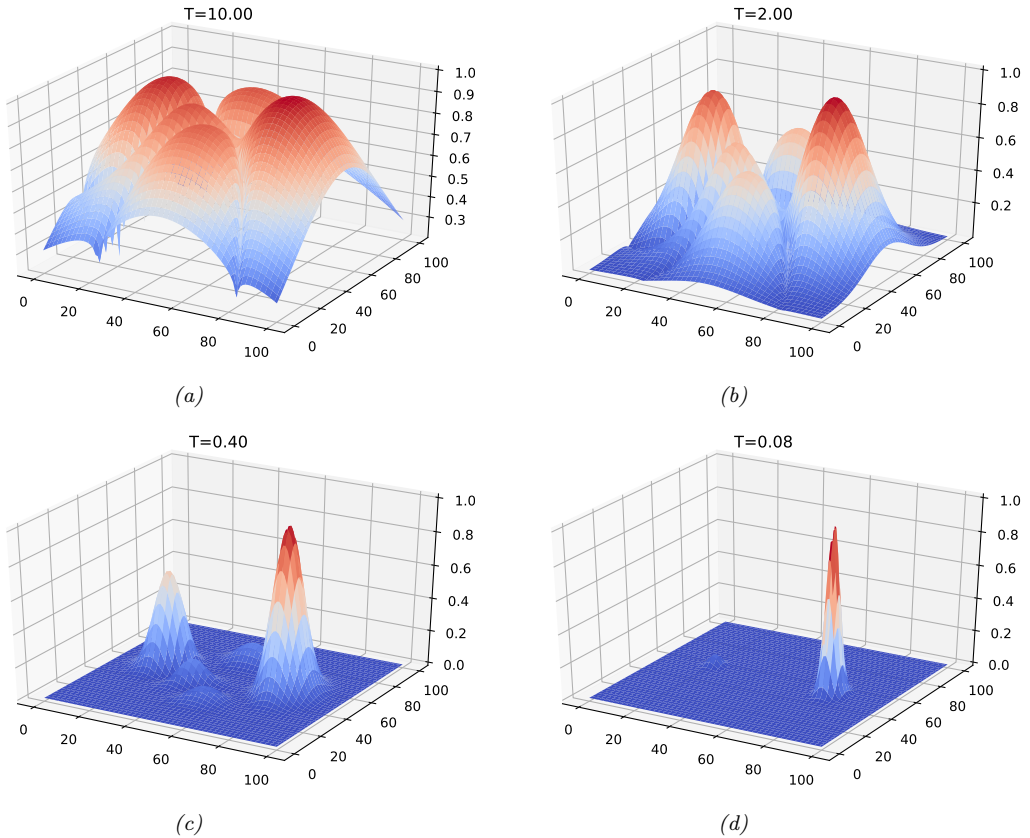


Figure 12.22: Annealed version of the distribution in Figure 12.21a at different temperatures. Generated by simulated_annealing_2d_demo.ipynb.

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

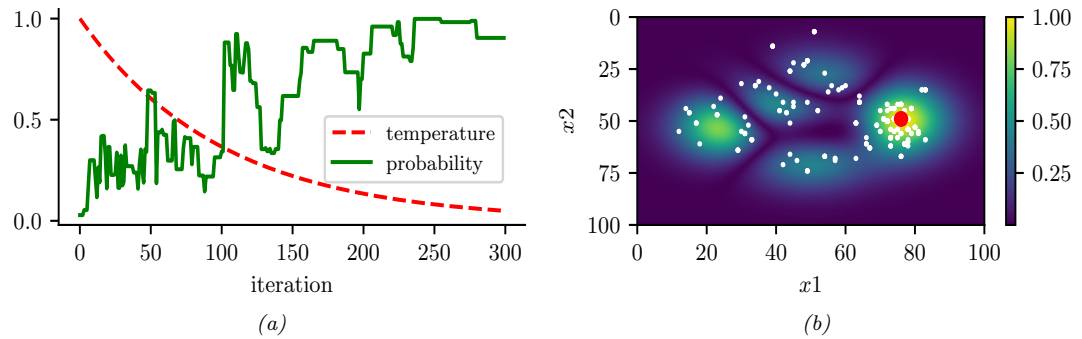


Figure 12.23: Simulated annealing applied to the distribution in Figure 12.21a. (a) Temperature vs iteration and probability of each visited point vs iteration. (b) Visited samples, superimposed on the target distribution. The big red dot is the highest probability point found. Generated by simulated_annealing_2d_demo.ipynb.

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.

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).

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.

13 Sequential Monte Carlo inference

13.1 Introduction

In this chapter, we discuss **sequential Monte Carlo** or **SMC** algorithms, which can be used to sample from a sequence of related probability distributions. SMC is most commonly used to solve filtering in state-space models (SSM, Chapter 29), but it can also be applied to other problems, such as sampling from a static (but possibly multi-modal) distribution, or for sampling rare events from some process.

Our presentation is based on the excellent tutorial [NLS19], and differs from traditional presentations, such as [Aru+02], by emphasizing the fact that we are sampling sequences of related variables, not just computing the filtering distribution of an SSM. This more general perspective will let us tackle static estimation problems, as we will see. For another good introduction to SMC, see [DJ11]. For a more formal (measure theoretic) treatment of SMC, using the **Feynman-Kac** formalism, see [CP20b].

13.1.1 Problem statement

In SMC, the goal is to sample from a sequence of related distributions of the form

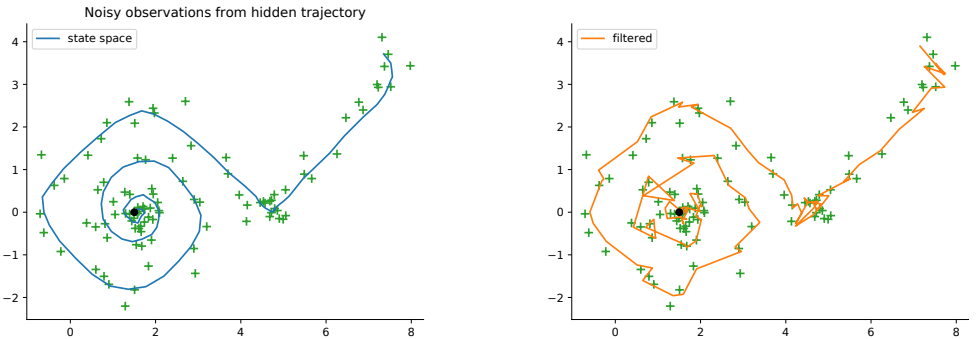
$$\pi_t(\boldsymbol{z}_{1:t}) = \frac{1}{Z_t}\tilde{\gamma}_t(\boldsymbol{z}_{1:t}) \tag{13.1}$$

for $t = 1 : T$, where $\tilde{\gamma}_t$ is the unnormalized **target distribution**, π_t is the normalized version, and $\boldsymbol{z}_{1:t}$ are the random variables of interest. In some applications (e.g., filtering in an SSM), we care about each intermediate marginal distribution, $\pi_t(\boldsymbol{z}_t)$, for $t = 1 : T$; this is called **particle filtering**. (The word "particle" just means "sample".) In other applications, we only care about the final distribution, $\pi_T(\boldsymbol{z}_T)$, and the intermediate steps are introduced just for computational reasons; this is called an **SMC sampler**. We briefly review both of these below, and go into more detail in later sections.

13.1.2 Particle filtering for state-space models

An important application of SMC is to sequential (online) inference (state estimation) in SSMs. As an example, consider a Markovian state-space model with the following joint distribution:

$$\pi_T(\boldsymbol{z}_{1:T}) \propto p(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}) = p(\boldsymbol{z}_1)p(\boldsymbol{y}_1|\boldsymbol{z}_1)\prod_{t=1}^{T} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{y}_t|\boldsymbol{z}_t) \tag{13.2}$$



(a) *(b)*

Figure 13.1: Illustration of particle filtering (using the dynamical prior as the proposal) applied to a 2d nonlinear dynamical system. (a) True underlying state and observed data. (b) PF estimate of the posterior mean. Generated by bootstrap_filter_spiral.ipynb.

A common choice is to define the unnormalized target distribution at step t to be

$$\tilde{\gamma}_t(\boldsymbol{z}_{1:t}) = p(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_1)p(\boldsymbol{y}_1|\boldsymbol{z}_1)\prod_{s=1}^{t} p(\boldsymbol{z}_s|\boldsymbol{z}_{s-1})p(\boldsymbol{y}_s|\boldsymbol{z}_s) \tag{13.3}$$

Note that this a distribution over an (ever growing) sequence of latent variables. However, we often only care about the most recent marginal of this distribution, in which case we just need to compute $\tilde{\gamma}_t(\boldsymbol{z}_t)$, which avoids having to store the full history.

For example, consider the following 2d nonlinear tracking problem (the same one as in Section 8.5.5.1):

$$\begin{aligned} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) &= \mathcal{N}(\boldsymbol{z}_t|f(\boldsymbol{z}_{t-1}), q\mathbf{I}) \\ p(\boldsymbol{y}_t|\boldsymbol{z}_t) &= \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{z}_t, r\mathbf{I}) \\ \boldsymbol{f}(\boldsymbol{z}) &= (z_1 + \Delta\sin(z_2), z_2 + \Delta\cos(z_1)) \end{aligned} \tag{13.4}$$

where Δ is the step size of the underlying continuous system, q is the variance of the system noise, and r is the variance of the observation noise. (We treat Δ, q and r as fixed constants; see Supplementary Section 13.1.3 for a discussion of joint state and parameter estimation.) The true underlying state trajectory, and the corresponding noisy measurements, are shown in Figure 13.1a. The posterior mean estimate of the state, computed using 2000 samples in a simple form of SMC called the bootstrap filter (Section 13.2.3.1), is shown in Figure 13.1b.

Particle filtering can also be applied to **non-Markovian models**, where $\boldsymbol{z}_t$ may depend on all the past hidden states, $\boldsymbol{z}_{1:t-1}$, and $\boldsymbol{y}_t$ depends on the current $\boldsymbol{z}_t$ and possibly also all the past hidden states, $\boldsymbol{z}_{1:t-1}$, and optionally the past observatiobns, $\boldsymbol{y}_{1:t-1}$. In this case, the unnormalized target

distribution at step t is

$$\tilde{\gamma}_t(\boldsymbol{z}_{1:t}) = p(\boldsymbol{z}_1)p(\boldsymbol{y}_1|\boldsymbol{z}_1)\prod_{s=1}^{t} p(\boldsymbol{z}_s|\boldsymbol{z}_{1:s-1})p(\boldsymbol{y}_s|\boldsymbol{z}_{1:s}) \tag{13.5}$$

For example, consider a 1d Gaussian sequence model where the dynamics are first-order Markov, but the observations depend on the entire past sequence (this is example 1.2.1 from [NLS19]):

$$\begin{aligned} p(z_t|\boldsymbol{z}_{1:t-1}) &= \mathcal{N}(z_t|\phi z_{t-1}, q) \\ p(y_t|\boldsymbol{z}_{1:t}) &= \mathcal{N}(y_t|\sum_{s=1}^{t}\beta^{t-s}z_s, r) \end{aligned} \tag{13.6}$$

If we set $\beta = 0$, we get $p(y_t|\boldsymbol{z}_{1:t}) = \mathcal{N}(y_t|z_t, r)$ (where we define $0^0 = 1$), so the model becomes a linear-Gaussian SSM. As β gets larger, the dependence on the past increases, making the inference problem harder. (We will revisit this example below.)

13.1.3 SMC samplers for static parameter estimation

Now consider the problem of parameter estimation from a fixed dataset, $\mathcal{D} = \{\boldsymbol{y}_n : n = 1 : N_{\mathcal{D}}\}$. We suppose the observations are conditionally iid, so the posterior has the form $p(\boldsymbol{z}|\mathcal{D}) \propto p(\boldsymbol{z})\prod_{n=1}^{N_{\mathcal{D}}} p(\boldsymbol{y}_n|\boldsymbol{z})$, where $\boldsymbol{z}$ is the unknown parameter. It is not immediately obvious how to approximate $p(\boldsymbol{z}|\mathcal{D})$ using SMC, since we just have one distribution. However, we can convert this into a sequential inference problem in several different ways. One approach, known as **data tempering**, defines the (marginal) target distribution at step t as $\tilde{\gamma}_t(\boldsymbol{z}_t) = p(\boldsymbol{z}_t)p(\boldsymbol{y}_{1:t}|\boldsymbol{z}_t)$. In this case, the number of time steps T is the same as the number of data samples, $N_{\mathcal{D}}$. Another approach, known as **likelihood tempering**, defines the (marginal) target distribution at step t as $\tilde{\gamma}_t(\boldsymbol{z}_t) = p(\boldsymbol{z}_t)p(\mathcal{D}|\boldsymbol{z}_t)^{\tau_t}$, where $0 = \tau_t < \cdots < \tau_T = 1$ is a temperature parameter. In this case, the number of steps T depends on how quickly we anneal the distibution from the initial prior $p(\boldsymbol{z}_1)$ to the final target $p(\boldsymbol{z}_T)p(\mathcal{D}|\boldsymbol{z}_T)$.

Once we have defined the marginal target distributions $\tilde{\gamma}_t(\boldsymbol{z}_t)$, we need a way to expand this to a joint target distribution over a *sequence* of variables, $\tilde{\gamma}_t(\boldsymbol{z}_{1:t})$, so the distributions become connected to each other. We explain how to do this in Section 13.6. We can then treat the model as an SSM and apply particle filtering. At the end, we extract the final joint target distribution, $\tilde{\gamma}_T(\boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_{1:T})p(\mathcal{D}|\boldsymbol{z}_T)$, from which we can compute the marginal target distribution $\tilde{\gamma}_T(\boldsymbol{z}_T) = p(\boldsymbol{z}_T, \mathcal{D})$, from which we can get the posterior $p(\boldsymbol{z}|\mathcal{D})$ by normalizing. We give the details in Section 13.6.

13.2 Particle filtering

In this section, we cover the basics of SMC for state space models, culiminating in a method known as the **particle filter**.

13.2.1 Importance sampling

We start by reviewing the self-normalized importance sampling method (SNIS, Section 11.5), which is the foundation of the particle filter.

Suppose we are interested in estimating the expectation of some function φ_t with respect to a target distribution π_t, which we denote by

$$\pi_t(\varphi) \triangleq \mathbb{E}_{\pi_t}\left[\varphi_t(\boldsymbol{z}_{1:t})\right] = \int \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{Z_t}\varphi_t(\boldsymbol{z}_{1:t})d\boldsymbol{z}_{1:t} \tag{13.7}$$

where $Z_t = \int \tilde{\gamma}_t(\boldsymbol{z}_{1:t})d\boldsymbol{z}_{1:t}$. Suppose we use SNIS with proposal $q_t(\boldsymbol{z}_{1:t})$. We then get the following approximation:

$$\pi_t(\varphi) \approx \frac{1}{\hat{Z}_t}\frac{1}{N_s}\sum_{i=1}^{N_s}\tilde{w}_t(\boldsymbol{z}_{1:t}^i)\varphi_t(\boldsymbol{z}_{1:t}^i) \tag{13.8}$$

where $\boldsymbol{z}_{1:t}^i \overset{\text{iid}}{\sim} q_t$ are independent samples from the proposal, $\tilde{w}_t^i$ are the **unnormalized weights** defined by

$$\tilde{w}_t^i = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t}^i)}{q_t(\boldsymbol{z}_{1:t}^i)} \tag{13.9}$$

and $\hat{Z}_t$ is the approximate normalization constant defined by

$$\hat{Z}_t \triangleq \frac{1}{N_s}\sum_{i=1}^{N_s}\tilde{w}_t^i \tag{13.10}$$

To simplify notation, let us define the **normalized weights** by

$$W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j} \tag{13.11}$$

Then we can write

$$\mathbb{E}_{\pi_t}\left[\varphi_t(\boldsymbol{z}_{1:t})\right] \approx \sum_{i=1}^{N_s} W_t^i \varphi_t(\boldsymbol{z}_{1:t}^i) \tag{13.12}$$

Alternatively, instead of computing the expectation of a specific target function, we can just approximate the target distribution itself, using a sum of weighted samples:

$$\pi_t(\boldsymbol{z}_{1:t}) \approx \sum_{i=1}^{N_s} W_t^i \delta(\boldsymbol{z}_{1:t} - \boldsymbol{z}_{1:t}^i) \triangleq \hat{\pi}_t(\boldsymbol{z}_{1:t}) \tag{13.13}$$

The problem with importance sampling when applied in the context of sequential models is that the dimensionality of the state space is very large, and increases with t. This makes it very hard to define a good proposal that covers the high probability regions, resulting in most samples getting negligible weight. In the sections below, we discuss solutions to this problem.

13.2.2 Sequential importance sampling

In this section, we discuss **sequential importance sampling** or **SIS**, in which the proposal has the following autoregressive structure:

$$q_t(\boldsymbol{z}_{1:t}) = q_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) \tag{13.14}$$

We can obtain samples from $q_{t-1}(\boldsymbol{z}_{1:t-1})$ by reusing the $\boldsymbol{z}^i_{1:t-1}$ samples, which we then extend by one step by sampling from the conditional $q_t(\boldsymbol{z}_t|\boldsymbol{z}^i_{1:t-1})$. We can think of this as "growing" the chain (sequence of states). The unnormalized weights can be computed recursively as follows:

$$\tilde{w}_t(\boldsymbol{z}_{1:t}) = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{q_t(\boldsymbol{z}_{1:t})} = \frac{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1})} \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) q_{t-1}(\boldsymbol{z}_{1:t-1})} \tag{13.15}$$

$$= \frac{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1})}{q_{t-1}(\boldsymbol{z}_{1:t-1})} \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})} \tag{13.16}$$

$$= \tilde{w}_{t-1}(\boldsymbol{z}_{1:t-1}) \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})} \tag{13.17}$$

The ratio factors are sometimes called the **incremental importance weights**:

$$\alpha_t(\boldsymbol{z}_{1:t}) = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})} \tag{13.18}$$

See Algorithm 31 for pseudocode for the resulting SIS algorithm. (In practice we compute the weights in log-space, and convert back using the log-sum-exp trick.)

Note that, in the special case of state space models, the weight computation can be further simplified. In particular, suppose we have

$$\tilde{\gamma}_t(\boldsymbol{z}_{1:t}) = p(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) = p(\boldsymbol{y}_t|\boldsymbol{z}_{1:t}) p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) p(\boldsymbol{z}_{1:t-1}, \boldsymbol{y}_{1:t-1}) \tag{13.19}$$

$$= p(\boldsymbol{y}_t|\boldsymbol{z}_{1:t}) p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) \tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}) \tag{13.20}$$

Then the incremental weight is given by

$$\alpha_t(\boldsymbol{z}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{z}_{1:t}) p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) \tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})} = \frac{p(\boldsymbol{y}_t|\boldsymbol{z}_{1:t}) p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})}{q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})} \tag{13.21}$$

Unfortunately SIS suffers from a problem known as **weight degeneracy** or **particle impoverishment**, in which most of the weights become very small (near zero), so the posterior ends up being approximated by a single particle. This is illustrated in Figure 13.2a, where we apply SIS to the non-Markovian example in Equation (13.6) using $N_s = 5$ particles. The reason for degeneracy is that each particle has to "explain" (generate) the entire sequence of observations. Each sequence of guessed states becomes increasingly improbable over time, due to the product of likelihood terms, and the differences between the weights of each hypothesis will grow exponentially. Of course, there has to be a best sequence amongst the set of candidates, so when we normalize the weights, the best one will get weight 1 and the rest will get weight 0. But this is a waste of most of the particles. We discuss a solution to this in Section 13.2.3.

Algorithm 31: Sequential importance sampling (SIS)

Initialization: $\boldsymbol{z}_1^i \sim q_1(\boldsymbol{z}_1)$, $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\boldsymbol{z}_1^i)}{q_1(\boldsymbol{z}_1^i)}$, $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$, $\hat{\pi}_1(\boldsymbol{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\boldsymbol{z}_1 - \boldsymbol{z}_1^i)$

for $t = 2 : T$ **do**

  for $i = 1 : N_s$ **do**

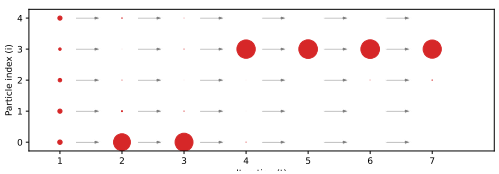
    Sample $\boldsymbol{z}_t^i \sim q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}^i)$

    Compute incremental weight $\alpha_t^i = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}^i) q_t(\boldsymbol{z}_t^i | \boldsymbol{z}_{1:t-1}^i)}$

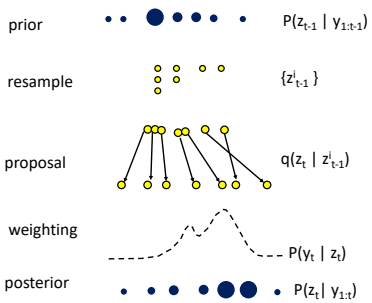
    Compute unnormalized weight $\tilde{w}_t^i = \tilde{w}_{t-1}^i \alpha_t^i$

  Compute normalized weights $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$ for $i = 1 : N_s$

  Compute MC posterior $\hat{\pi}_t(\boldsymbol{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\boldsymbol{z}_{1:t} - \boldsymbol{z}_{1:t}^i)$



(a)



(b)

Figure 13.2: (a) Illustration of weight degeneracy for SIS applied to the model in Equation (13.6). with parameters $(\phi, q, \beta, r) = (0.9, 10.0, 0.5, 1.0)$. We use $T = 6$ steps and $N_s = 5$ samples. We see that as t increases, almost all the probability mass concentrates on particle 3. Generated by sis_vs_smc.ipynb. Adapted from Figure 2 of [NLS19]. (b) Illustration of the bootstrap particle filtering algorithm.

13.2.3 Sequential importance sampling with resampling

In this section, we describe **sequential importance sampling with resampling** (**SISR**). The basic idea is this: instead of "growing" all of the old particle sequences by one step, we first select the N_s "fittest" particles, by sampling from the old posterior, and then we let these survivors grow by one step.

In more detail, at step t, we sample from

$$q_t^{\text{SISR}}(\boldsymbol{z}_{1:t}) = \hat{\pi}_{t-1}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}) \tag{13.22}$$

where $\hat{\pi}_{t-1}(\boldsymbol{z}_{1:t-1})$ is the previous weighted posterior approximation. By contrast, in SIS, we sample from

$$q_t^{\text{SIS}}(\boldsymbol{z}_{1:t}) = q_{t-1}^{\text{SIS}}(\boldsymbol{z}_{1:t-1}) q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}) \tag{13.23}$$

Algorithm 32: Sequential importance sampling with resampling (SISR)

Initialization: $\boldsymbol{z}_1^i \sim q_1(\boldsymbol{z}_1)$, $\tilde{w}_1^i = \frac{\tilde{\gamma}_1(\boldsymbol{z}_1^i)}{q_1(\boldsymbol{z}_1^i)}$, $W_1^i = \frac{\tilde{w}_1^i}{\sum_j \tilde{w}_1^j}$, $\hat{\pi}_1(\boldsymbol{z}_1) = \sum_{i=1}^{N_s} W_1^i \delta(\boldsymbol{z}_1 - \boldsymbol{z}_1^i)$
for $t = 2 : T$ **do**
  Compute ancestors $\boldsymbol{a}_{t-1}^{1:N_s} = \text{resample}(\tilde{w}_{t-1}^{1:N_s})$
  Select $\boldsymbol{z}_{t-1}^{1:N_s} = \text{permute}(\boldsymbol{a}_{t-1}^{1:N_s}, \boldsymbol{z}_{t-1}^{1:N_s})$
  Reset unnormalized weights $\tilde{w}_{t-1}^{1:N_s} = 1/N_s$
  for $i = 1 : N_s$ **do**
    Sample $\boldsymbol{z}_t^i \sim q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}^i)$
    Compute unnormalized weight $\tilde{w}_t^i = \alpha_t^i = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}^i) q_t(\boldsymbol{z}_t^i | \boldsymbol{z}_{1:t-1}^i)}$
  Compute normalized weights $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$ for $i = 1 : N_s$
  Compute MC posterior $\hat{\pi}_t(\boldsymbol{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\boldsymbol{z}_{1:t} - \boldsymbol{z}_{1:t}^i)$

We can sample from Equation (13.22) in two steps. First we **resample** N_s samples from $\hat{\pi}_{t-1}(\boldsymbol{z}_{1:t-1})$ to get a *uniformly weighted* set of new samples $\boldsymbol{z}_{1:t-1}^i$. (See Section 13.2.4 for details on how to do this.) Then we extend each sample using $\boldsymbol{z}_t^i \sim q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}^i)$, and concatenate $\boldsymbol{z}_t^i$ to $\boldsymbol{z}_{1:t-1}^i$,

After making a proposal, we compute the unnormalized weights. We use the standard SNIS method, except we "pretend" that the proposal is given by $\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}^i) q_t(\boldsymbol{z}_t^i | \boldsymbol{z}_{1:t-1}^i)$ even though we used $\hat{\pi}_{t-1}(\boldsymbol{z}_{1:t-1}^i) q_t(\boldsymbol{z}_t^i | \boldsymbol{z}_{1:t-1}^i)$. The intuitive reason why this is valid is because the previous weighted approximation, $\hat{\pi}_{t-1}(\boldsymbol{z}_{1:t-1}^i)$, was an unbiased estimate of the previous target distribution, $\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1})$. (See e.g., [CP20b] for more theoretical details.) We then compute the unnormalized weights, which are the same as the incremental weights, since the resampling step sets $\tilde{w}_{t-1}^i = 1$. We then normalize these weights and compute the new approximationg to the target posterior $\hat{\pi}_t(\boldsymbol{z}_{1:t})$. See Algorithm 32 for the pseudocode.

13.2.3.1 Bootstrap filter

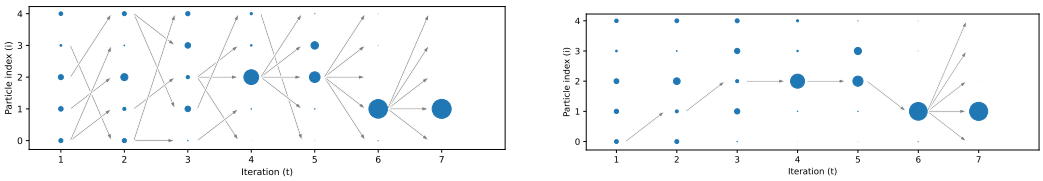
We now consider a special case of SISR, in which the model is an SSM, and the proposal distribution is equal to the dynamical prior:

$$q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}) = p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}) \tag{13.24}$$

In this case, the corresponding incremental weight in Equation (13.21) simplifies to

$$\alpha_t(\boldsymbol{z}_{1:t}) = \frac{p(\boldsymbol{y}_t | \boldsymbol{z}_{1:t}) p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1})}{q(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1})} = \frac{p(\boldsymbol{y}_t | \boldsymbol{z}_t) p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1})}{p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1})} = p(\boldsymbol{y}_t | \boldsymbol{z}_{1:t}) \tag{13.25}$$

This special case is called the **bootstrap filter** [Gor93] or the **survival of the fittest** algorithm [KKR95]. (In the computer vision literature, this is called the **condensation** algorithm, which stands for "conditional density propagation" [IB98].) See Figure 13.2b for an illustration of how this algorithm works, and Figure 13.1b for some sample results on real data.



(a) *(b)*

Figure 13.3: (a) Illustration of diversity of samples in SMC applied to the model in Equation (13.6). (b) Illustration of the path degeneracy problem. Generated by sis_vs_smc.ipynb. Adapted from Figure 3 of [NLS19].

The bootstrap filter is useful for models where we can sample from the dynamics, but cannot evaluate the transition model pointwise. This occurs in certain implicit dynamical models, such as those defined using differential equatons (see e.g., [IBK06]); such models are often used in epidemiology. However, in general it is much more efficient to use proposals that take the current evidence $\boldsymbol{y}_t$ into account. We discuss ways to approximate such "locally optimal" proposals in Section 13.3.

13.2.3.2 Path degeneracy problem

In Figure 13.3a we show how particle filtering can result in a much more diverse set of active particles, with more balanced weights when applied to the non-Markovian example in Equation (13.6).

While particle filtering does not suffer from weight degeneracy, it does suffer from another problem known as **path degeneracy**. This refers to the fact that the number of particles that "survive" (have non-negligible weight) over many steps may drop rapidly over time, resulting in a loss of diversity when we try to represent the distribution over the past. We illustrate this in Figure 13.3b, where we only include arrows for samples that have been resampled at each step up until the final step. We see that we have $N_s = 5$ identical copies of $\boldsymbol{z}_1^1$ in the final set of surviving sequences. (The time at which all the paths meet at a common ancestor, when tracing backwards in time, is known as the **coalescence** time.) We discuss some ways to ameliorate this issue in Section 13.2.4 and Section 13.2.5.

13.2.3.3 Estimating the normalizing constant

We can use particle filtering to approximate the normalization constant $Z_T = p(\boldsymbol{y}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})$ as follows:

$$\hat{Z}_T = \prod_{t=1}^{T} \hat{Z}_t \tag{13.26}$$

where, from Equation (13.10), we have

$$\hat{Z}_t = \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{w}_t^i = \hat{Z}_{t-1} \left(\widehat{Z_t / Z_{t-1}} \right) \tag{13.27}$$

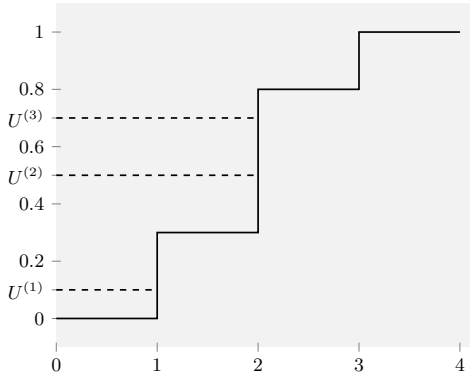


Figure 13.4: Illustration of how to sample from the empirical CDF $P(x) = \sum_{n=1}^{N} W^n \mathbb{I}(x \geq n)$ shown in black. The height of step n is W_n. If U^m picks step n, then we set the ancestor of m to be n, i.e., $A^m = n$. In this example, $A^{1:3} = (1, 2, 2)$. Adapted from Figure 9.3 of [CP20b].

where

$$\widehat{Z_t/Z_{t-1}} = \frac{\sum_{i=1}^{N_s} \tilde{w}_t^i}{\sum_{i=1}^{N_s} \tilde{w}_{t-1}^i} \tag{13.28}$$

This estimate of the marginal likelihood is very useful for tasks such as parameter estimation.

13.2.4 Resampling methods

In this section, we discuss various resampling methods, which can help reduce weight degeneracy and path degeneracy.

13.2.4.1 Multinomial resampling

The simplest approach to resampling is known as **multinomial resampling**. This works as follows. First we form the cumulative distribution from the weights $W_{t-1}^{1:N_s}$, as illustrated by the staircase in Figure 13.4. Then then we sample N_s uniform random variables, $u^i \sim \{0, 1\}$. Finally, we see which bin (interval) u^i lands in; if it falls in bin a, we assign the new sample $\boldsymbol{z}_{t-1}^i$ to be the same as the old $\boldsymbol{z}_{t-1}^a$. We say that a is the **ancestor** of i. For precisely, we say a is the ancestor of sample i if

$$\sum_{j=1}^{a-1} W_{t-1}^j \leq u^i < \sum_{j=1}^{a} W_{t-1}^j \tag{13.29}$$

See Listing 13.1 for some Python code.

Listing 13.1: Multinomial resampling

```
def multinomial_resampling(w):
  N = w.shape[0]
  u = np.random.rand(N)
```

```
  bins = np.cumsum(w)
  ancestors = np.digitize(u, bins)
  return ancestors
```

Although this is a simple method, it can introduce a lot of variance into the representation of the distribution. For example, suppose all the weights are equal, $W^n = 1/N$. Let $\mathcal{W}^n = \sum_{m=1}^{N} \mathbb{I}(A^m = n)$ be the number of "offspring" for particle n (i.e., the number of times this particle is chosen in the resampling step). We have $\mathcal{W}^n \sim \text{Bin}(N, 1/N)$, so $P(\mathcal{W}^n = 0) = (1 - 1/N)^N \approx e^{-1} \approx 0.37$. So there is a 37% chance that any given particle will disappear even though they all had the same initial weight. In the sections below, we discuss some **low variance resampling** methods.

13.2.4.2 Stratified resampling

A simple approach to improve on multinomial resampling is to use **stratified resampling**, in which we divide the unit interval into N_s strata, $(0, 1/N_s)$, $(1/N_s, 2/N_s)$, up to $(1 - 1/N_s, 1)$. We then generate $u^i \sim \text{Unif}((i-1)/N_s, i/N_s)$ and derive the corresponding ancestor indexes using Equation (13.29). See Listing 13.2 for some Python code.

Listing 13.2: Stratified resampling

```
def stratified_resampling(w):
  N = w.shape[0]
  u = (np.arange(N) +  np.random.rand(N))/N
  bins = np.cumsum(w)
  ancestors = np.digitize(u, bins)
  return ancestors
```

13.2.4.3 Systematic resampling

We can further reduce the variance by forcing all the samples from $\hat{\pi}_{t-1}$ to be deterministically generated from a shared random source, $u \sim \text{Unif}(0, 1)$, by computing

$$u^i = \frac{i-1}{N_s} + \frac{u}{N_s} \tag{13.30}$$

We derive the corresponding ancestor indexes using Equation (13.29). See Listing 13.3 for some Python code. (The only difference from Listing 13.2 is the use of `np.random.rand()` instead of `np.random.rand(N)`.)

Listing 13.3: Systematic resampling

```
def systematic_resampling(w):
  N = w.shape[0]
  u = (np.arange(N) +  np.random.rand())/N
  bins = np.cumsum(w)
  ancestors = np.digitize(u, bins)
  return ancestors
```

13.2.4.4 Comparison

It can be proved that all of the above methods are unbiased. It can also be proved that stratified resampling is lower variance than multinomial resampling. Empirically it seems that systematic resampling is lower variance than other methods [HSG06]. A more complex resampling scheme, that is guaranteed to converge and which is also low variance, is described in [GCW19].

13.2.5 Adaptive resampling

The resampling step can result in loss of diversity, since each ancestor may generate multiple children, and some may generate no children, since the ancestor indices A_t^n are sampled independently; this is the path degeneracy problem mentioned above. On the other hand, if we never resample, we end up with SIS, which suffers from weight degeneracy (particles with negligible weight). A compromise is to use **adaptive resampling**, in which we resample whenever the **effective sample size** or **ESS** drops below some minimum, such as $N/2$. A common way to define the ESS is as follows:[1]

$$\text{ESS}(W^{1:N}) = \frac{1}{\sum_{n=1}^N (W^n)^2} \tag{13.31}$$

Alternatively we can compute the ESS using the unnormalized weights:

$$\text{ESS}(\tilde{w}^{1:N}) = \frac{\left(\sum_{n=1}^N \tilde{w}^n\right)^2}{\sum_{n=1}^N (\tilde{w}^n)^2} \tag{13.32}$$

Note that if we have k weights with $\tilde{w}^n = 1$ and $N-k$ weights with $\tilde{w}^n = 0$, then the ESS is k; thus ESS is between 1 and N.

The pseudocode for SISR with adaptive resampling is given in Algorithm 33. (We use the notation of [Law+22, App. B], in which we first sample new extensions of the sequences, and then optionally resample the sequences at the end of each step.)

Algorithm 33: SISR with adaptive resampling (generic SMC)

Initialization: $\tilde{w}_0^{1:N_s} = 1$, $\hat{Z}_0 = 1$
for $t = 1 : T$ **do**
  for $i = 1 : N_s$ **do**
    Sample particle $\boldsymbol{z}_t^i \sim q_t(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}^i)$
    Compute incremental weight $\alpha_t^i = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t}^i)}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{1:t-1}^i) q_t(\boldsymbol{z}_t^i | \boldsymbol{z}_{1:t-1}^i)}$
    Compute unnormalized weight $\tilde{w}_t^i = \tilde{w}_{t-1}^i \alpha_t^i$
  Estimate normalization constant: $\widehat{Z_t/Z_{t-1}} = \frac{\sum_{i=1}^{N_s} \tilde{w}_t^i}{\sum_{i=1}^{N_s} \tilde{w}_{t-1}^i}$, $\hat{Z}_t = \hat{Z}_{t-1}(\widehat{Z_t/Z_{t-1}})$
  if $ESS(\tilde{w}_{t-1}^{1:N}) < ESS_{\min}$ **then**
    Compute ancestors $\boldsymbol{a}_t^{1:N_s} = \text{resample}(\tilde{w}_t^{1:N_s})$
    Select $\boldsymbol{z}_t^{1:N_s} = \text{permute}(\boldsymbol{a}_t^{1:N_s}, \boldsymbol{z}_t^{1:N_s})$
    Reset unnormalized weights $\tilde{w}_t^{1:N_s} = 1/N_s$
  Compute normalized weights $W_t^i = \frac{\tilde{w}_t^i}{\sum_j \tilde{w}_t^j}$ for $i = 1 : N_s$
  Compute MC posterior $\hat{\pi}_t(\boldsymbol{z}_{1:t}) = \sum_{i=1}^{N_s} W_t^i \delta(\boldsymbol{z}_{1:t} - \boldsymbol{z}_{1:t}^i)$

1. Note that the ESS used in SMC is different than the ESS used in MCMC (Section 12.6.3); the latter takes into account auto-correlation of the MCMC samples.

13.3 Proposal distributions

The efficiency of PF is crucially dependent on the quality of the proposal distribution. We discuss some options below.

13.3.1 Locally optimal proposal

We define the (one-step) **locally optimal proposal distribution** $q_t^*(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})$ to be the one that minimizes

$$D_{\mathbb{KL}}\left(\pi_{t-1}(\boldsymbol{z}_{1:t-1})q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) \parallel \pi_t(\boldsymbol{z}_{1:t})\right) \tag{13.33}$$

$$= \mathbb{E}_{\pi_{t-1}q_t}\left[\log\left\{\pi_{t-1}(\boldsymbol{z}_{1:t-1})q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})\right\} - \log \pi_t(\boldsymbol{z}_{1:t})\right] \tag{13.34}$$

$$= \mathbb{E}_{\pi_{t-1}q_t}\left[\log q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) - \log \pi_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})\right] + \text{const} \tag{13.35}$$

$$= \mathbb{E}_{\pi_{t-1}q_t}\left[D_{\mathbb{KL}}\left(q_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) \parallel \pi_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1})\right)\right] + \text{const} \tag{13.36}$$

The KL is minimized by choosing

$$q_t^*(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) = \pi_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{1:t})}{\tilde{\gamma}_t(\boldsymbol{z}_{1:t-1})} \tag{13.37}$$

where $\tilde{\gamma}_t(\boldsymbol{z}_{1:t-1}) = \int \tilde{\gamma}_t(\boldsymbol{z}_{1:t})d\boldsymbol{z}_t$ is the probability of the past sequence under the current target distribution.

Note that the subscript t specifies the t'th distribution, so in the context of SSMs, we have $\pi_t(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}) = p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{y}_{1:t})$. Thus we see that when proposing $\boldsymbol{z}_t$, we should condition on all the data, including the most recent observation, $\boldsymbol{y}_t$; this is called a **guided particle filter**, and will will be better than the bootstrap filter, which proposes from the prior.

In general, it is intractable to compute the locally optimal proposal, so we consider various approximations below.

13.3.2 Proposals based on the extended and unscented Kalman filter

One way to approximate the locally optimal proposal distribution is based on the extended Kalman filter (Section 8.5.2) or the unscented Kalman filter (Section 13.3.2, which gives rise to the **extended particle filter** [DGA00] and **unscented particle filter** [Mer+00] respectively. To explain these methods, we follow the presentation of [NLS19, p36]. As usual, we assume the dynamical system can be written as $\boldsymbol{z}_t = \boldsymbol{f}(\boldsymbol{z}_{t-1}) + \boldsymbol{q}_t$ and $\boldsymbol{y}_t = \boldsymbol{h}(\boldsymbol{z}_t) + \boldsymbol{r}_t$, where $\boldsymbol{q}_t$ is the system noise and $\boldsymbol{r}_t$ is the observation noise. The EKF and UKF approximations assume that the joint distribution over neighboring time steps, given the i'th history, is Gaussian:

$$p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{z}_{1:t-1}^i) \approx \mathcal{N}\left(\begin{pmatrix}\boldsymbol{z}_t \\ \boldsymbol{y}_t\end{pmatrix} | \hat{\boldsymbol{\mu}}^i, \hat{\boldsymbol{\Sigma}}^i\right) \tag{13.38}$$

where

$$\hat{\boldsymbol{\mu}}^i = \begin{pmatrix}\hat{\boldsymbol{\mu}}_z^i \\ \hat{\boldsymbol{\mu}}_y^i\end{pmatrix}, \hat{\boldsymbol{\Sigma}}^i = \begin{pmatrix}\hat{\boldsymbol{\Sigma}}_{zz}^i & \hat{\boldsymbol{\Sigma}}_{zy}^i \\ \hat{\boldsymbol{\Sigma}}_{yz}^i & \hat{\boldsymbol{\Sigma}}_{yy}^i\end{pmatrix} \tag{13.39}$$

(See Section 8.8 for details.)

The EKF and UKF compute $\hat{\boldsymbol{\mu}}^i$ and $\hat{\boldsymbol{\Sigma}}^i$ differently. In the EKF, we linearize $\boldsymbol{f}$ and $\boldsymbol{h}$, and assume the noise terms are Gaussian. We then compute $p(\boldsymbol{z}_t, \boldsymbol{y}_t|\boldsymbol{z}^i_{1:t-1})$ exactly for this linearized model (see Section 8.5.1). In the UKF, we propagate sigma points through $\boldsymbol{f}$ and $\boldsymbol{h}$, and approximate the resulting means and covariances using the unscented transform, which can be more accurate (see Section 8.7). Once we have computed $\hat{\boldsymbol{\mu}}^i$ and $\hat{\boldsymbol{\Sigma}}^i$, we can use standard rules for Gaussian conditioning to compute the approximate proposal as follows:

$$q(\boldsymbol{z}_t|\boldsymbol{z}^i_{1:t-1}, \boldsymbol{y}_t) \approx \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}^i_t, \boldsymbol{\Sigma}^i_t) \tag{13.40}$$

$$\boldsymbol{\mu}^i_t = \hat{\boldsymbol{\mu}}^i_z + \hat{\boldsymbol{\Sigma}}^i_{zy}(\hat{\boldsymbol{\Sigma}}^i_{yy})^{-1}(\boldsymbol{y}_t - \hat{\boldsymbol{\mu}}^i_y) \tag{13.41}$$

$$\boldsymbol{\Sigma}^i_t = \hat{\boldsymbol{\Sigma}}^i_{zz} - \hat{\boldsymbol{\Sigma}}^i_{zy}(\hat{\boldsymbol{\Sigma}}^i_{yy})^{-1}\hat{\boldsymbol{\Sigma}}^i_{yz} \tag{13.42}$$

Note that the linearization (or sigma point) approximation needs to be performed for each particle sepatately.

13.3.3 Proposals based on the Laplace approximation

To handle non-Gaussian likelihoods in an SSM, we can use the Laplace approximation (Section 7.4.3), as suggested in [DGA00]. In particular, consider an SSM with linear-Gaussian latent dynamics and a GLM likelihood. At each step, we compute the maximum $\boldsymbol{z}^*_t = \text{argmax} \log p(\boldsymbol{y}_t|\boldsymbol{z}_t)$ as step t (e.g., using Newton-Raphson), and then approximate the likelihood using

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t) \approx \mathcal{N}(\boldsymbol{z}_t|\boldsymbol{z}^*_t, -\mathbf{H}^*_t) \tag{13.43}$$

where $\mathbf{H}^*_t$ is the Hessian of the log-likelihood at the mode. We now compute $p(\boldsymbol{z}_t|\boldsymbol{z}^i_{t-1}, \boldsymbol{y}_t)$ using the update step of the Kalman filter, using the same equations as in Section 13.3.2. This combination is called the the **Laplace Gaussian filter** [Koy+10]. We give an example in Section 13.3.3.1.

13.3.3.1 Example: neural decoding

In this section, we give an example where we apply the Laplace approximation to an SSM with linear-Gaussian dynamics and a Poisson likelihood. The application arises from neuroscience. In particular, assume we record the **neural spike trains** as a monkey moves its hand around in space. Let $\boldsymbol{z}_t \in \mathbb{R}^6$ represent the 3d location and velocity of the hand. We model the dynamics of the hand using a simple Brownian random walk model [CP20b, p157]:

$$\begin{pmatrix} z_t(i) \\ z_t(i+3) \end{pmatrix} | \boldsymbol{z}_{t-1} \sim \mathcal{N}_2\left(\begin{pmatrix} 1 & \Delta \\ 0 & 1 \end{pmatrix} \begin{pmatrix} z_{t-1}(i) \\ z_{t-1}(i+3) \end{pmatrix}, \sigma^2 \mathbf{Q} \right), \; i = 1:3 \tag{13.44}$$

where the covariance of the noise is given by the following, assuming a discretization step of Δ:

$$\mathbf{Q} = \begin{pmatrix} \Delta^3/3 & \Delta^2/2 \\ \Delta^2/2 & \Delta \end{pmatrix} \tag{13.45}$$

We assume the k'th observation at time t is the number of spikes for neuron k in this sensing interval:

$$p(y_t(k)|\boldsymbol{z}_t) = \text{Poi}(\lambda_k(\boldsymbol{z}_t)) \tag{13.46}$$

$$\log \lambda_k(\boldsymbol{z}_t) = \alpha_k + \boldsymbol{\beta}_k^\mathsf{T} \boldsymbol{z}_t \tag{13.47}$$

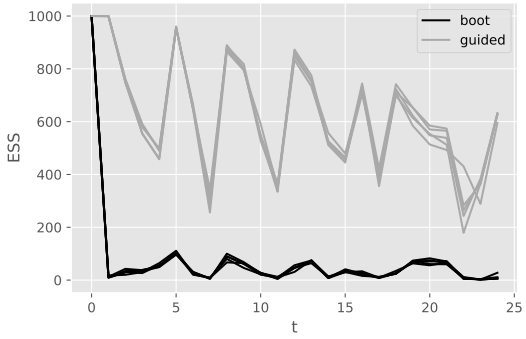


Figure 13.5: Effective sample size at each step for the bootstrap particle filter and a guided particle filter for a Gaussian SSM with Poisson likelihood. Adapted from Figure 10.4 of [CP20b]. Generated by pf_guided_neural_decoding.ipynb.

Our goal is to compute $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t})$, which lets us infer the position of the hand from the neural code. (Apart from its value for furthering basic science, this can be useful for applications such as helping disabled people control their arms using "mind control".)

To illustrate this, we sample a synthetic dataset from the model, to simulate a "monkey" moving its arm for $T = 25$ time steps; this generates $K = 50$ neuronal counts per time step. We then apply particle filtering to this dataset (using the true model), using either the bootstrap filter (i.e., proposal is the random walk prior) or the guided filter (i.e., proposal is the Laplace approximation mentioned above). In Figure 13.5, we see that the effective sample size of the guided filter is much higher than for the bootstrap filter.

13.3.4 Proposals based on SMC (nested SMC)

It is possible to use SMC as a subroutine to compute a proposal distribution for SMC: at each step t, for each particle i, we run an SMC algorithm where the target distribution is the optimal proposal, $p(\boldsymbol{z}_t|\boldsymbol{z}^i_{1:t-1}, \boldsymbol{y}_{1:t})$. This is called **nested SMC** [NLS15; NLS19].

This method can approximate the locally optimal proposal arbitrarily well, since it does not make any limiting parametric assumptions. However, the method can be slow, although the inner SMC algorithm can be run in parallel for each outer sample [NLS15; NLS19].

13.4 Rao-Blackwellised particle filtering (RBPF)

In some models, we can partition the hidden variables into two kinds, $\boldsymbol{m}_t$ and $\boldsymbol{z}_t$, such that we can analytically integrate out $\boldsymbol{z}_t$ provided we know the values of $\boldsymbol{m}_{1:t}$. This means we only have to sample $\boldsymbol{m}_{1:t}$, and can represent $p(\boldsymbol{z}_t|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t})$ parametrically. These hybrid particles are sometimes called **distributional particles** or **collapsed particles** [KF09a, Sec 12.4]. This combines techniques from particle filtering (Section 13.2) with deterministic methods such as Kalman filtering (Section 8.3.2).

The advantage of this approach is that we reduce the dimensionality of the space in which we are

sampling, which reduces the variance of our estimate. This technique is known as **Rao-Blackwellised particle filtering** or **RBPF** for short. (See Section 11.6.2 for more details on Rao-Blackwellisation.) In Section 13.4.1 we give an example of RBPF for inference in a switching linear dynamical systems. In Section 13.4.3 we illustrate RBPF for inference in the SLAM model for a mobile robot.

13.4.1 Mixture of Kalman filters

In this section, we consider the application of RBPF to a switching linear dynamical system (Section 29.9). This model has both continuous and discrete latent variables. This can be used to track a system that switches between discrete modes or operating regimes, represented by the discrete variable m_t.

For notational simplicity, we ignore the control inputs $\boldsymbol{u}_t$. Thus the model is given by

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, m_t = k) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_k\boldsymbol{z}_{t-1}, \mathbf{Q}_k) \tag{13.48}$$

$$p(\boldsymbol{y}_t|\boldsymbol{z}_t, m_t = k) = \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_k\boldsymbol{z}_t, \mathbf{R}_k) \tag{13.49}$$

$$p(m_t = k|m_{t-1} = j) = A_{jk} \tag{13.50}$$

We let $\boldsymbol{\theta}_k = (\mathbf{F}_k, \mathbf{H}_k, \mathbf{Q}_k, \mathbf{R}_k, \mathbf{A}_{:,k})$ represent all the parameters for state k.

Exact inference is intractable, but if we sample the discrete variables, we can infer the continuous variables conditoned on the discretes exactly, making this a good candidate for RBPF. In particular, if we sample trajectories $\boldsymbol{m}_{1:t}^n$, we can apply a Kalman filter to each particle. This can be thought of as a **mixture of Kalman filters** [CL00]. The resulting belief state is represented by

$$p(\boldsymbol{z}_t, \boldsymbol{m}_t|\boldsymbol{y}_{1:t}) \approx \sum_{n=1}^{N} W_t^n \delta(\boldsymbol{m}_t - \boldsymbol{m}_t^n)\mathcal{N}(\boldsymbol{z}_t|\boldsymbol{\mu}_t^n, \boldsymbol{\Sigma}_t^n) \tag{13.51}$$

To derive the filtering algorithm, note that the full posterior at time t can be written as follows:

$$p(\boldsymbol{m}_{1:t}, \boldsymbol{z}_{1:t}|\boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_{1:t}|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t})p(\boldsymbol{m}_{1:t}|\boldsymbol{y}_{1:t}) \tag{13.52}$$

The second term is given by the following:

$$p(\boldsymbol{m}_{1:t}|\boldsymbol{y}_{1:t}) \propto p(\boldsymbol{y}_t|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t-1})p(\boldsymbol{m}_{1:t}|\boldsymbol{y}_{1:t-1}) \tag{13.53}$$

$$= p(\boldsymbol{y}_t|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t-1})p(\boldsymbol{m}_t|\boldsymbol{m}_{1:t-1}, \boldsymbol{y}_{1:t-1})p(\boldsymbol{m}_{1:t-1}|\boldsymbol{y}_{1:t-1}) \tag{13.54}$$

$$= p(\boldsymbol{y}_t|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t-1})p(\boldsymbol{m}_t|\boldsymbol{m}_{t-1})p(\boldsymbol{m}_{1:t-1}|\boldsymbol{y}_{1:t-1}) \tag{13.55}$$

Note that, unlike the case of standard particle filtering, we cannot write $p(\boldsymbol{y}_t|\boldsymbol{m}_{1:t}, \boldsymbol{y}_{1:t-1}) = p(\boldsymbol{y}_t|\boldsymbol{m}_t)$, since $\boldsymbol{m}_t$ does not d-separate the past observations from $\boldsymbol{y}_t$, as is evident from Figure 29.25a.

Suppose we use the following recursive proposal distribution:

$$q(\boldsymbol{m}_{1:t}|\boldsymbol{y}_{1:t}) = q(\boldsymbol{m}_t|\boldsymbol{m}_{1:t-1}, \boldsymbol{y}_{1:t})q(\boldsymbol{m}_{1:t-1}|\boldsymbol{y}_{1:t}) \tag{13.56}$$

Then we get the unnormalized importance weights

$$\tilde{w}_t^n \propto \frac{p(\boldsymbol{y}_t|m_t^n, \boldsymbol{m}_{1:t-1}^n, \boldsymbol{y}_{1:t-1})p(m_t^n|m_{t-1}^n)}{q(\boldsymbol{m}_t^n|\boldsymbol{m}_{1:t-1}^n, \boldsymbol{y}_{1:t})}\tilde{w}_{t-1}^n \tag{13.57}$$

As a special case, suppose we propose from the prior, $q(m_t|\boldsymbol{m}_{t-1}^n, \boldsymbol{y}_{1:t}) = p(m_t|m_{t-1}^n)$. If we sample discrete state k, the weight update becomes

$$\tilde{w}_t^n \propto \tilde{w}_{t-1}^n p(\boldsymbol{y}_t|m_t^n = k, \boldsymbol{m}_{1:t-1}^n, \boldsymbol{y}_{1:t-1}) = \tilde{w}_{t-1}^n L_{tk}^n \tag{13.58}$$

where

$$L_{tk}^n = p(\boldsymbol{y}_t|m_t = k, \boldsymbol{m}_{1:t-1}^n, \boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{y}_t|m_t = k, \boldsymbol{z}_t) p(\boldsymbol{z}_t|m_t = k, \boldsymbol{y}_{1:t-1}, \boldsymbol{m}_{1:t-1}^n) d\boldsymbol{z}_t \tag{13.59}$$

The quantity L_{tk}^n is the predictive density for the new observation $\boldsymbol{y}_t$ conditioned on $m_t = k$ and the history of previous latents, $\boldsymbol{m}_{1:t-1}^n$. In the case of SLDS models, this can be computed using the normalization constant of the Kalman filter, Equation (8.80). The resulting algorithm is shown in Algorithm 34. The step marked "KFupdate" refers to the Kalman filter update equations in Section 8.3.2, and is applied to each particle separately.

Algorithm 34: One step of RBPF for SLDS using prior as proposal

for $n = 1 : N$ **do**
  $k \sim p(m_t|m_{t-1}^n)$
  $m_t^n := k$
  $(\boldsymbol{\mu}_t^n, \boldsymbol{\Sigma}_t^n, L_{tk}^n) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^n, \boldsymbol{\Sigma}_{t-1}^n, \boldsymbol{y}_t, \boldsymbol{\theta}_k)$
  $\tilde{w}_t^n = \tilde{w}_{t-1}^n L_{tk}^n$
Compute ESS $= \text{ESS}(\tilde{w}_t^{1:N_s})$
if $ESS < ESS_{\min}$ **then**
  $\boldsymbol{a}_t^{1:N} = \text{Resample}(\tilde{w}_t^{1:N})$
  $(\boldsymbol{m}_t^{1:N_s}, \boldsymbol{\mu}_t^{1:N_s}, \boldsymbol{\Sigma}_t^{1:N_s}) = \text{permute}(\boldsymbol{a}_t, \boldsymbol{m}_t^{1:N_s}, \boldsymbol{\mu}_t^{1:N_s}, \boldsymbol{\Sigma}_t^{1:N_s})$
  $\tilde{w}_t^n = 1/N_s$

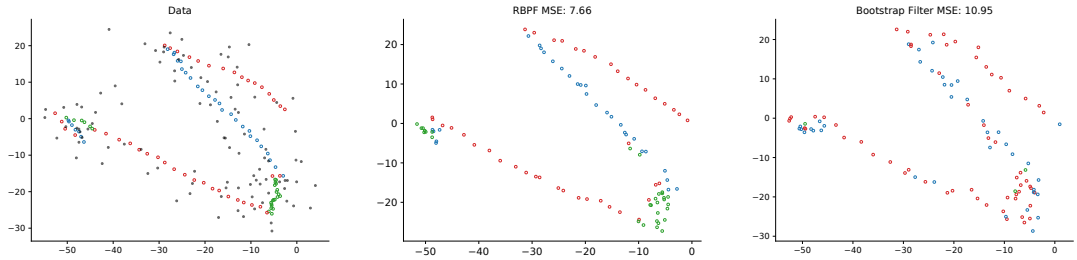
13.4.1.1 Improvements

An improved version of the algorithm can be developed based on the fact that we are sampling a discrete state space. At each step, we propagate each of the N old particles through all K possible transition models. We then compute the weight for all NK new particles, and sample from this to get the final set of N particles. This latter step can be done using the **optimal resampling** method of [FC03], which will stochastically select the particles with the largest weight, while also ensuring the result is an unbiased approximation. In addition, this approach ensures that we do not have duplicate particles, which is wasteful and unnecessary when the state space is discrete.

13.4.2 Example: tracking a maneuvering object

In this section we give an example of RBPF for an SLDS from [DGK01]. Our goal is to track an object that has the following motion model:

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, m_t = k) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}\boldsymbol{z}_{t-1} + \boldsymbol{b}_k, \mathbf{Q}) \tag{13.60}$$



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

Figure 13.6: Illustration of state estimation for a switching linear model. (a) Black dots are observations, hollow circles are the true location, colors represent the discrete state. (b) Estimate from RBPF. Generated by rbpf_maneuver.ipynb. (c) Estimate from bootstrap filter. Generated by bootstrap_filter_maneuver.ipynb.

where $\boldsymbol{z}_t = (x_{1t}, \dot{x}_{1t}, x_{2t}, \dot{x}_{2t})$ contains the 2d position and velocity. We define the observaton matrix by $\mathbf{H} = \mathbf{I}$ and the observation covariance by $\mathbf{R} = 10\text{diag}(2, 1, 2, 1)$. We define the dynamics matrix by

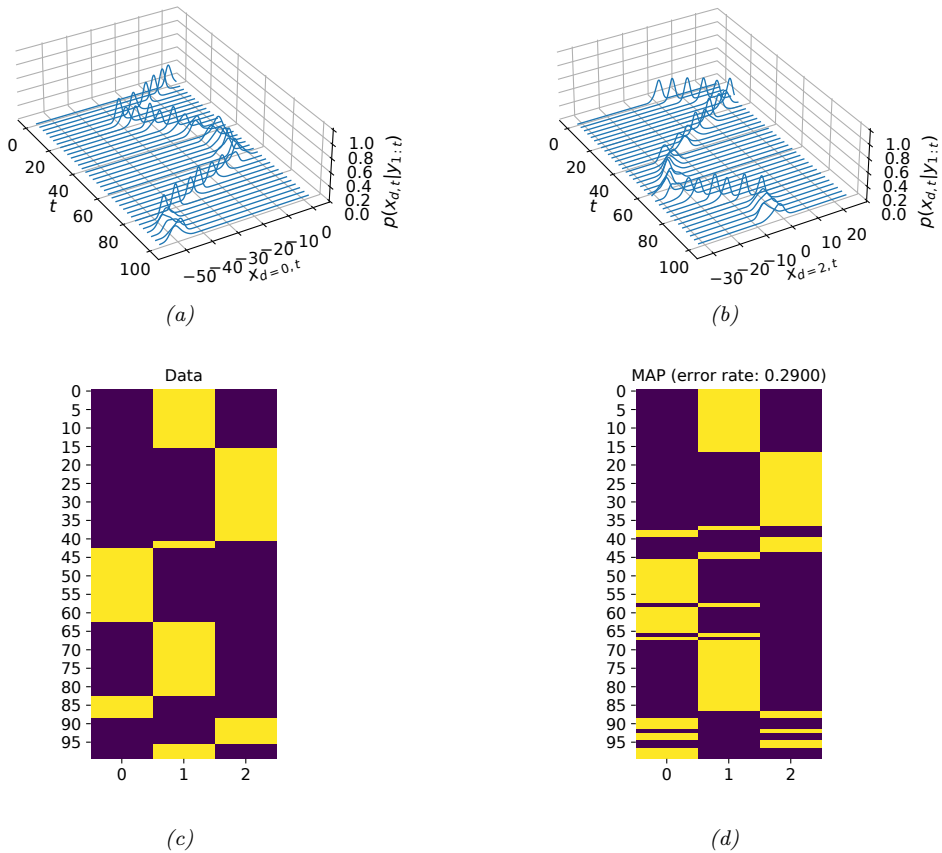
$$\mathbf{F} = \begin{pmatrix} 1 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{13.61}$$

where $\Delta = 0.1$,. We set the the noise covariance to $\mathbf{Q} = 0.2\mathbf{I}$ and the input bias vectors for each state to $\boldsymbol{b}_1 = (0, 0, 0, 0)$, $\boldsymbol{b}_2 = (-1.225, -0.35, 1.225, 0.35)$ and $\boldsymbol{b}_3 = (1.225, 0.35, -1.225, -0.35)$. Thus the system will turn in different directions depending on the discrete state. The discrete state transition matrix is given by

$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \tag{13.62}$$

Figure 13.6a shows some observations, and the true state of the system, from a sample run, for 100 steps. The colors denote the discrete state, and the location of the symbol denotes the (x, y) location. The small dots represent noisy observations. Figure 13.6b shows the estimate of the state computed using RBPF with the optimal proposal with 1000 particles. In Figure 13.6c, we show the analogous estimate using the boostrap filter, which does much worse.

In Figure 13.7a and Figure 13.7b, we show the posterior marginals of the (x, y) locations over time. In Figure 13.7c we show the true discrete state, and in Figure 13.7d we show the posterior marginal over discrete states. The overall state classification error rate is 29%, but it seems that occasionally misclassifying isolated time steps does not significantly hurt estimation of the continuous states, as we can see from Figure 13.6b.



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

Figure 13.7: Visualizing the posterior from the RBPF algorithm. Top row: Posterior marginals of the location of the object over time, derived from the mixture of Gaussian representation for (a) x location (dimension 0), (b) y location (dimension 2). Bottom row: visualization of the true (c) and predicted (d) discrete states. Generated by rbpf_maneuver.ipynb.

13.4.3 Example: FastSLAM

Consider a robot moving around an environment, such as a maze or indoor office environment. It needs to learn a map of the environment, and keep track of its location (pose) within that map. This problem is known as **simultaneous localization and mapping**, or **SLAM** for short. SLAM is widely used in mobile robotics (see e.g., [SC86; CN01; TBF06] for details). It is also useful in augmented reality, where the task is to recursively estimate the 3d pose of a handheld camera with respect to a set of 2d visual landmarks (this is known as **visual SLAM**, [TUI17; SMT18; Cza+20; DH22]).

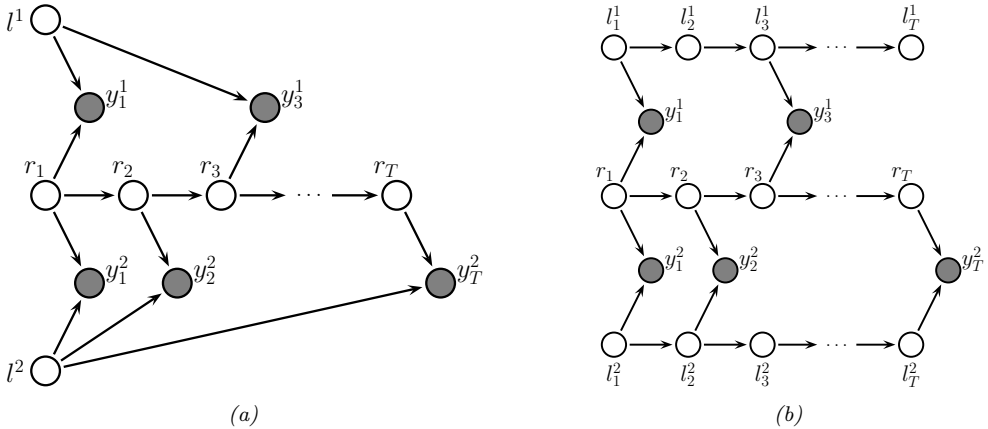


Figure 13.8: Graphical model representing the SLAM problem. $\boldsymbol{l}_t^k$ is the location of landmark k at time t, $\boldsymbol{r}_t$ is the location of the robot at time t, and $\boldsymbol{y}_t$ is the observation vector. In the model on the left, the landmarks are static (so they act like global shared parameters), on the right, their location can change over time. The robot's observations are based on the distance to the nearest landmarks from the current state, denoted $f(\boldsymbol{r}_t, \boldsymbol{l}_t^k)$. The number of observations per time step is variable, depending on how many landmarks are within the range of the sensor. Adapted from Figure 15.A.3 of [KF09a].

Let us assume we can represent the map as the 2d locations of a set of K landmarks, denote them by $\boldsymbol{l}^1, \ldots, \boldsymbol{l}^K$ (each is a vector in $\mathbb{R}^2$). (We can use data association to figure out which landmark generated each observation, as discussed in Section 29.9.3.2.) Let $\boldsymbol{r}_t$ represent the unknown location of the robot at time t. Let $\boldsymbol{z}_t = (\boldsymbol{r}_t, \boldsymbol{l}_t^{1:K})$ be the combined state space. We can then perform online inference so that the robot can update its estimate of its own location, and the landmark locations.

The state transition model is defined as

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{u}_t) = p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{l}_{t-1}^{1:K}, \boldsymbol{u}_t) \prod_{k=1}^{K} p(\boldsymbol{l}_t^k|\boldsymbol{l}_{t-1}^k) \tag{13.63}$$

where $p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{l}_{t-1}^{1:K}, \boldsymbol{u}_t)$ specifies how the robot moves given the control signal $\boldsymbol{u}_t$ and the location of the obstacles $\boldsymbol{l}_{t-1}^{1:K}$. (Note that in this section, we assume that a human is joysticking the robot through the environment, so $\boldsymbol{u}_{1:t}$ is given as input, i.e., we do not address the decision-theoretic issue of choosing where to move.)

If the obstacles (landmarks) are static, we can define $p(\boldsymbol{l}_t^k|\boldsymbol{l}_{t-1}^k) = \delta(\boldsymbol{l}_t^k - \boldsymbol{l}_{t-1}^k)$, which is equivalent to treating the map as an unknown parameter that is shared globally across all time steps. More generally, we can let the landmark locations evolve over time [Mur00].

The observations $\boldsymbol{y}_t$ measure the distance from $\boldsymbol{r}_t$ to the set of closest landmarks. Figure 13.8 shows the corresponding graphical model for the case where $K = 2$, and where on the first step it sees landmarks 1 and 2, then just landmark 2, then just landmark 1, etc.

If all the CPDs are linear-Gaussian, then we can use a Kalman filter to maintain our belief state about the location of the robot and the location of the landmarks, $p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t})$. In the more general case of a nonlinear model, we can use the EKF (Section 8.5.2) or UKF (Section 8.7.2).

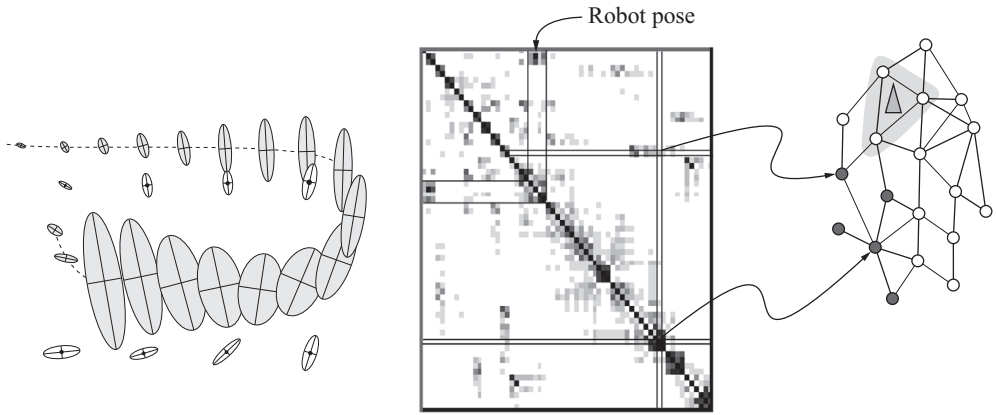


Figure 13.9: Illustration of the SLAM problem. (a) A robot starts at the top left and moves clockwise in a circle back to where it started. We see how the posterior uncertainty about the robot's location increases and then decreases as it returns to a familar location, closing the loop. If we performed smoothing, this new information would propagate backwards in time to disambiguate the entire trajectory. (b) We show the precision matrix, representing sparse correlations between the landmarks, and between the landmarks and the robot's position (pose). The conditional independencies encoded by the sparse precision matrix can be visualized as a Gaussian graphical model, as shown on the right. From Figure 15.A.3 of [KF09a]. Used with kind permission of Daphne Koller.

Over time, the uncertainty in the robot's location will increase, due to wheel slippage etc., but when the robot returns to a familiar location, its uncertainty will decrease again. This is called **closing the loop**, and is illustrated in Figure 13.9(a), where we see the uncertainty ellipses, representing $\text{Cov}\left[\boldsymbol{z}_t|\boldsymbol{y}_{1:t}, \boldsymbol{u}_{1:t}\right]$, grow and then shrink.

In addition to visualizing the uncertainty of the robot's location, we can visualize the uncertainty about the map. To do this, consider the posterior precision matrix, $\boldsymbol{\Lambda}_t = \boldsymbol{\Sigma}_t^{-1}$. Zeros in the precision matrix correspond to absent edges in the corresponding undirected Gaussian graphical model (GGM, see Section 4.3.5). Initially all the beliefs about landmark locations are uncorrelated (by assumption), so the GGM is a disconnected graph, and $\boldsymbol{\Lambda}_t$ is diagonal. However, as the robot moves about, it will induce correlation between nearby landmarks. Intuitively this is because the robot is estimating its position based on distance to the landmarks, but the landmarks' locations are being estimated based on the robot's position, so they all become interdependent. This can be seen more clearly from the graphical model in Figure 13.8: it is clear that $\boldsymbol{l}^1$ and $\boldsymbol{l}^2$ are not d-separated by $\boldsymbol{y}_{1:t}$, because there is a path between them via the unknown sequence of $\boldsymbol{r}_{1:t}$ nodes. Consequently, the precision matrix becomes denser over time. As a consequence of the precision matrix becoming denser, each inference step takes $O(K^3)$ time. This prevents the method from being applied to large maps.

One way to speed this up is based on the following observation: conditional on knowing the robot's path, $\boldsymbol{r}_{1:t}$, the landmark locations are independent, i.e., $p(\boldsymbol{l}_t|\boldsymbol{r}_{1:t}, \boldsymbol{y}_{1:t}) = \prod_{k=1}^{K} p(\boldsymbol{l}_t^k|\boldsymbol{r}_{1:t}, \boldsymbol{y}_{1:t})$. This can be seen by looking at the DGM in Figure 13.8. We can therefore sample the trajectory using

some proposal, and apply (2d) Kalman filtering to each landmark independently. This is an example of RBPF, and reduces the inference cost to $O(NK)$, where N is the number of particles and K is the number of landmarks.

The overall cost of this technique is $O(NK)$ per step. Fortunately, the number of particles N needed for good performance is quite small, so the algorithm is essentially linear in the number of landmarks, making it quite scalable. This idea was first suggested in [Mur00], who applied it to grid-structured occupancy grids (and used the HMM filter for each particle). It was subsequently extended to landmark-based maps in [Thr+04], using the Kalman filter for each particle; they called the technique **FastSLAM**.

13.5 Extensions of the particle filter

There are many extensions to the basic particle filtering algorithm, such as the following:

- We can increase particle diversity by applying one or more steps of MCMC sampling (Section 12.2) at each PF step using $\pi_t(\boldsymbol{z}_t)$ as the target distribution. This is called the **resample-move** algorithm [DJ11]. It is also possible to use SMC instead of MCMC to diversify the samples [GM17].
- We can extend PF to the case of offline inference; this is called **particle smoothing** (see e.g., [Kla+06]).
- We can extend PF to inference in general graphical models (not just chains) by combining PF with loopy belief propagation (Section 9.3); this is called **non-parametric BP** or **particle BP** (see e.g., [Sud+03; Isa03; Sud+10; Pac+14]).
- We can extend PF to perform inference in static models (e.g., for parameter inference), as we discuss in Section 13.6.

13.6 SMC samplers

In this section, we discuss **SMC samplers** (sequential Monte Carlo samplers), which are a way to apply particle filters to sample from a generic target distribution, $\pi(\boldsymbol{z}) = \tilde{\gamma}(\boldsymbol{z})/Z$, rather than requiring the model to be an SSM. Thus SMC is an alternative to MCMC.

The advantages of SMC samplers over MCMC are as follows: we can estimate the normalizing constant Z; we can more easily develop adaptive versions that tune the transition kernel using the current set of samples; and the method is easier to parallelize (see e.g., [CCS22; Gre+22]).

The method works by defining a sequence of intermediate distributions, $\pi_t(\boldsymbol{z}_t)$, which we expand to a sequence of distributions over all the past variables, $\overline{\pi}_t(\boldsymbol{z}_{1:t})$. We then use the particle filtering algorithm to sample from each of these intermediate distributions. By marginalizing all but the final state, we recover samples from the target distribution, $\pi(\boldsymbol{z}) = \sum_{\boldsymbol{z}_{1:T-1}} \overline{\pi}_T(\boldsymbol{z}_{1:T})$, as we explain below. (For more details, see e.g., [Dai+20a; CP20b].)

13.6.1 Ingredients of an SMC sampler

To define an SMC sampler, we need to specify several ingredients:

- A sequence of distributions defined on the same state space, $\pi_t(\boldsymbol{z}_t) = \tilde{\gamma}_t(\boldsymbol{z}_t)/Z_t$, for $t = 0 : T$;
- A **forwards kernel** $M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ (often written as $M_t(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t)$), which satisfies $\sum_{\boldsymbol{z}_t} M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = 1$. This can be used to propose new samples from our current estimate when we apply particle filtering.
- A **backwards kernel** $L_t(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})$ (often written as $L(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})$), which satisfies $\sum_{\boldsymbol{z}_t} L_t(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}) = 1$. This allows us to create a sequence of variables by working backwards in time from the final target value to the first time step. In particular, we create the following joint distribution:

$$\overline{\pi}_t(\boldsymbol{z}_{1:t}) = \pi_t(\boldsymbol{z}_t) \prod_{s=1}^{t-1} L_s(\boldsymbol{z}_s|\boldsymbol{z}_{s+1}) \tag{13.64}$$

This satisfies $\sum_{\boldsymbol{z}_{1:t-1}} \overline{\pi}_t(\boldsymbol{z}_{1:t}) = \pi_t(\boldsymbol{z}_t)$, so if we apply particle filtering to this for $t = 1 : T$, then samples from the "end" of such sequences will be from the target distribution π_t.

With the above ingredients, we can compute the incremental weight at step t using

$$\alpha_t = \frac{\overline{\pi}_t(\boldsymbol{z}_{1:t})}{\overline{\pi}_{t-1}(\boldsymbol{z}_{1:t-1}) M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})} \propto \frac{\tilde{\gamma}_t(\boldsymbol{z}_t)}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{t-1})} \frac{L_{t-1}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})} \tag{13.65}$$

This can be plugged into the generic SMC algorithm, Algorithm 33.

We still have to specify the forwards and backwards kernels. We will assume the forwards kernel M_t is an MCMC kernel that leaves π_t invariant. We can then define the backwards kernel to be the **time reversal** of the forwards kernel. More precisely, suppose we define L_{t-1} so it satisfies

$$\pi_t(\boldsymbol{z}_t) L_{t-1}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) = \pi_t(\boldsymbol{z}_{t-1}) M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) \tag{13.66}$$

In this case, the incremental weight simplifies as follows:

$$\alpha_t = \frac{Z_t \pi_t(\boldsymbol{z}_t) L_{t-1}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{Z_{t-1} \pi_{t-1}(\boldsymbol{z}_{t-1}) M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})} \tag{13.67}$$

$$= \frac{Z_t \pi_t(\boldsymbol{z}_{t-1}) M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}{Z_{t-1} \pi_{t-1}(\boldsymbol{z}_{t-1}) M_t(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})} \tag{13.68}$$

$$= \frac{\tilde{\gamma}_t(\boldsymbol{z}_{t-1})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{t-1})} \tag{13.69}$$

We can use any kind of MCMC kernel for M_t. For example, if the parameters are real valued and unconstrained, we can use a Markov kernel that corresponds to K steps of a random walk Metropolis-Hastings sampler. We can set the covariance of the proposal to $\delta^2 \hat{\boldsymbol{\Sigma}}_{t-1}$, where $\hat{\boldsymbol{\Sigma}}_{t-1}$ is the empirical covariance of the weighted samples from the previous step, $(W_{t-1}^{1:N}, \boldsymbol{z}_{t-1}^{1:N})$, and $\delta = 2.38 D^{-3/2}$ (which is the optimal scaling parameter for RWMH). In high dimensional problems, we can use gradient based Markov kernels, such as HMC [BCJ20] and NUTS [Dev+21]. For binary state spaces, we can use the method of [SC13].

13.6.2 Likelihood tempering (geometric path)

There are many ways to specify the intermediate target distributions. In the **geometric path** method, we specify the intermediate distributions to be

$$\tilde{\gamma}_t(\boldsymbol{z}) = \tilde{\gamma}_0(\boldsymbol{z})^{1-\lambda_t} \tilde{\gamma}(\boldsymbol{z})^{\lambda_t} \tag{13.70}$$

where $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_T = 1$ are **inverse temperature** parameters, and $\tilde{\gamma}_0$ is the initial proposal. If we apply particle filtering to this model, but "turn off" the resampling step, the method becomes equivalent to **annealed importance sampling** (Section 11.5.4).

In the context of Bayesian parameter inference, we often denote the latent variable $\boldsymbol{z}$ by $\boldsymbol{\theta}$, we define $\tilde{\gamma}_0(\boldsymbol{\theta}) \propto \pi_0(\boldsymbol{\theta})$ as the prior, and $\tilde{\gamma}(\boldsymbol{z}) = \pi_0(\boldsymbol{\theta}) p(\mathcal{D}|\boldsymbol{\theta})$ as the posterior. We can then define the intermediate distributions to be

$$\tilde{\gamma}_t(\boldsymbol{\theta}) = \pi_0(\boldsymbol{\theta})^{1-\lambda_t} \pi_0(\boldsymbol{\theta})^{\lambda_t} p(\mathcal{D}|\boldsymbol{\theta})^{\lambda_t} = \pi_0(\boldsymbol{\theta})^{1-\lambda_t} \exp[-\lambda_t \mathcal{E}(\boldsymbol{\theta})] \tag{13.71}$$

where $\mathcal{E}(\boldsymbol{\theta}) = -\log p(\mathcal{D}, \boldsymbol{\theta})$ is the energy (potential) function. The incremental weights are given by

$$\alpha_t(\boldsymbol{\theta}) = \frac{\pi_0(\boldsymbol{\theta})^{1-\lambda_t} \exp[-\lambda_t \mathcal{E}(\boldsymbol{\theta})]}{\pi_0(\boldsymbol{\theta})^{1-\lambda_t} \exp[-\lambda_{t-1} \mathcal{E}(\boldsymbol{\theta})]} = \exp[-\delta_t \mathcal{E}(\boldsymbol{\theta})] \tag{13.72}$$

where $\lambda_t = \lambda_{t-1} + \delta_t$.

For this method to work well, it is important to to choose the λ_t so that the successive distributions are "equidistant"; this is called **adaptive tempering**. In the case of a Gaussian prior and Gaussian energy, one can show [CP20b] that this can be achieved by picking $\lambda_t = (1+\gamma)^{t+1} - 1$, where $\gamma > 0$ is some constant. Thus we should increase λ slowly at first, and then make bigger and bigger steps.

In practice we can estimate λ_t by setting $\lambda_t = \lambda_{t-1} + \delta_t$, where

$$\delta_t = \underset{\delta \in [0, 1-\lambda_{t-1}]}{\operatorname{argmin}} \left(\text{ESSLW}(\{-\delta \, \mathcal{E}(\boldsymbol{\theta}_t^n)\}) - \text{ESS}_{\min} \right) \tag{13.73}$$

where $\text{ESSLW}(\{l_n\}) = \text{ESS}(\{e^{l_n}\})$ computes the ESS (Equation (13.32)) from the log weights, $l_n = \log \tilde{w}^n$. This ensures the change in the ESS across steps is close to the desired minimum ESS, typically $0.5N$. (If there is no solution for δ in the interval, we set $\delta_t = 1 - \lambda_{t-1}$.) See Algorithm 35 for the overall algorithm.

13.6.2.1 Example: sampling from a 1d bimodal distribution

Consider the simple distribution

$$p(\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \mathbf{I}) \exp(-\mathcal{E}(\boldsymbol{\theta})) \tag{13.74}$$

where $\mathcal{E}(\boldsymbol{\theta}) = c(||\boldsymbol{\theta}||^2 - 1)^2$. We plot this in 1d in Figure 13.10a for $c = 5$; we see that it has a bimodal shape, since the low energy states correspond to parameter vectors whose norm is close to 1.

SMC is particularly useful for sampling from multimodal distributions, which can be provably hard to efficiently sample from using other methods, including HMC [MPS18], since gradients only provide local information about the curvature. As an example, in Figure 13.11a and Figure 13.11b we show the result of applying HMC (Section 12.5) and NUTS (Section 12.5.4.1) to this problem. We see that both algorithms get stuck near the initial state of $\theta_0 = 1$.

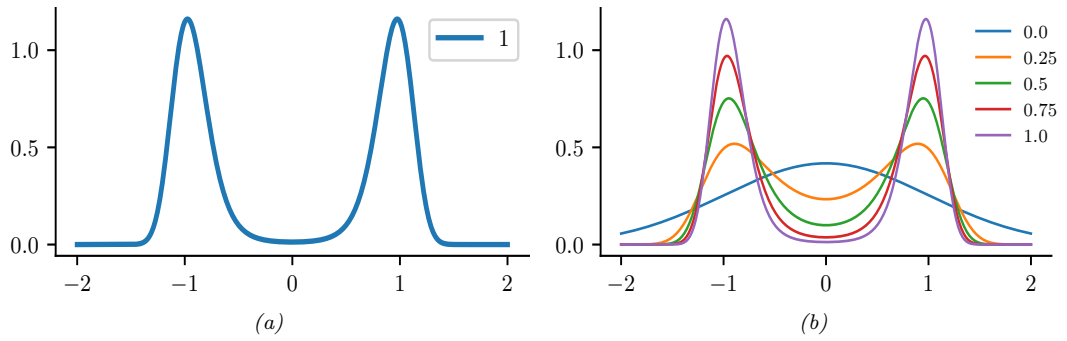


Figure 13.10: (a) Illustration of a bimodal target distribution. (b) Tempered versions of the target at different inverse temperatures, from $\lambda_T = 1$ down to $\lambda_1 = 0$. Generated by smc_tempered_1d_bimodal.ipynb.

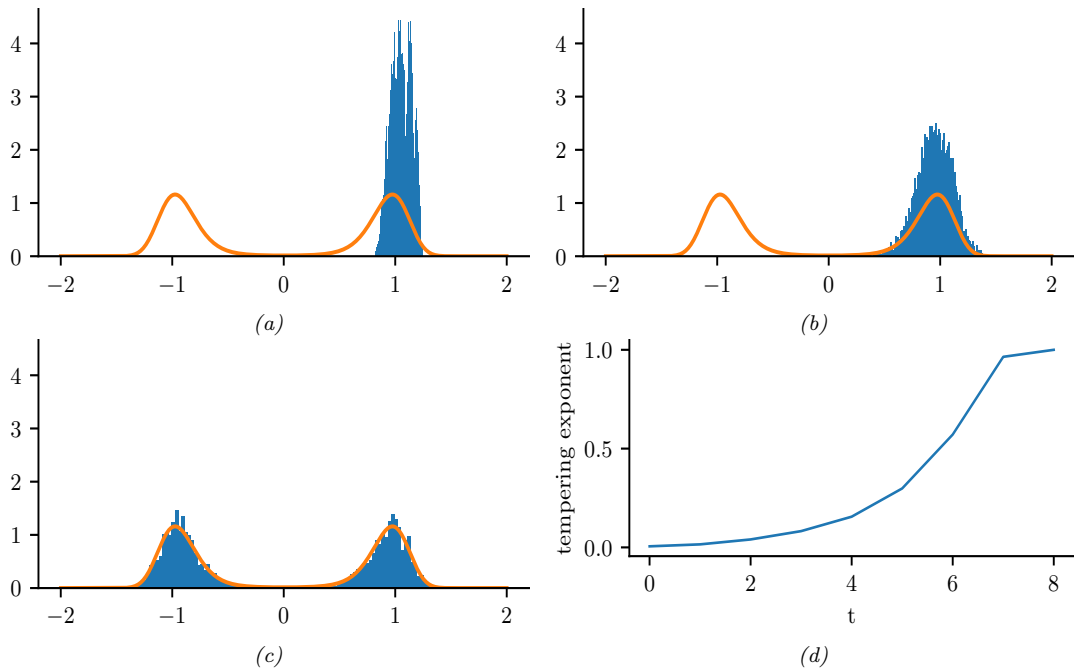


Figure 13.11: Sampling from the bimodal distribution in Figure 13.10a. (a) HMC. (b) NUTS. (c) Tempered SMC with HMC kernel (single step). (d) Adaptive inverse temperature schedule. Generated by smc_tempered_1d_bimodal.ipynb.

Algorithm 35: SMC with adaptive tempering

$\lambda_{-1} = 0$, $t = -1$, $W_{-1}^n = 1$
while $\lambda_t < 1$ **do**
  $t = t + 1$
  if $t = 0$ **then**
    $\boldsymbol{\theta}_0^n \sim \pi_0(\boldsymbol{\theta})$
  else
    $A_t^{1:N} = \text{Resample}(W_{t-1}^{1:N})$
    $\boldsymbol{\theta}_t^n \sim M_{\lambda_{t-1}}(\boldsymbol{\theta}_{t-1}^{A_t^n}, \cdot)$
  Compute δ_t using Equation (13.73)
  $\lambda_t = \lambda_{t-1} + \delta_t$
  $\tilde{w}_t^n = \exp[-\delta \mathcal{E}(\boldsymbol{\theta}_t^n)]$
  $W_t^n = \tilde{w}_t^n / (\sum_{m=1}^N \tilde{w}_t^m)$

In Figure 13.10b, we show tempered versions of the target distribution at 5 different temperatures, chosen uniformly in the interval $[0, 1]$. We see that at $\lambda_1 = 0$, the tempered target is equal to the Gaussian prior (blue line), which is easy to sample from. Each subsequent distribution is close to the previous one, so SMC can track the change until it ends up at the target distribution with $\lambda_T = 1$, as shown in Figure 13.11c.

These SMC results were obtained using the adaptive tempering scheme described above. In Figure 13.11d we see that initially the temperature is small, and then it increases exponentially. The algorithm takes 8 steps until $\lambda_T \geq 1$.

13.6.3 Data tempering

If we have a set of iid observations, we can define the t'th target to be

$$\tilde{\gamma}_t(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) p(\boldsymbol{y}_{1:t}|\boldsymbol{\theta}) \tag{13.75}$$

We can now apply SMC to this model. From Equation (13.69), the incremental weight becomes

$$\alpha_t(\boldsymbol{\theta}) = \frac{\tilde{\gamma}_t(\boldsymbol{z}_{t-1})}{\tilde{\gamma}_{t-1}(\boldsymbol{z}_{t-1})} = \frac{p(\boldsymbol{\theta}) p(\boldsymbol{y}_{1:t}|\boldsymbol{\theta})}{p(\boldsymbol{\theta}) p(\boldsymbol{y}_{1:t-1}|\boldsymbol{\theta})} = p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\theta}) \tag{13.76}$$

This can be plugged into the generic SMC algorithm in Algorithm 33.

Unfortunately, to sample from the MCMC kernel will typically take $O(t)$ time, since the MH accept/reject step requires computing $p(\boldsymbol{\theta}') \prod_{i=1}^t p(\boldsymbol{y}_{1i}|\boldsymbol{\theta}')$ for any proposed $\boldsymbol{\theta}'$. Hence the total cost is $O(T^2)$ if there are T observations. To reduce this, we can only sample parameters at times t when the ESS drops below a certain level; in the remaining steps, we just grow the sequence deterministically by repeating the previously sampled value. This technique was proposed in [Cho02], who called it the **iterated batch importance sampling** or **IBIS** algorithm.

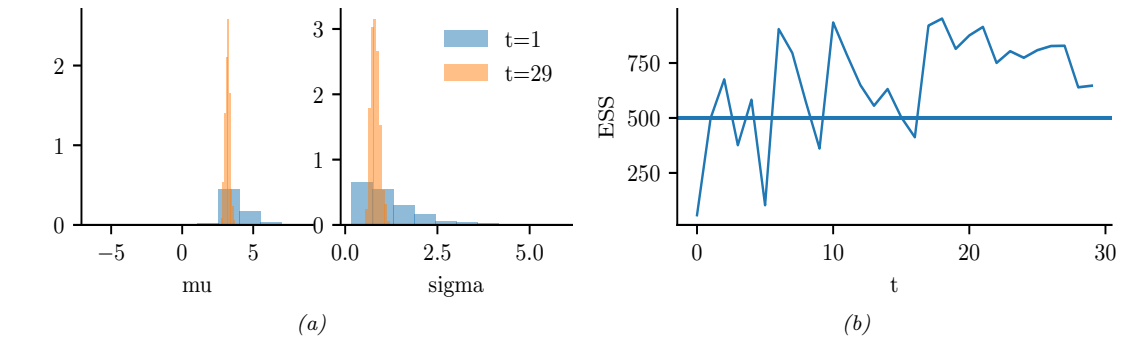


Figure 13.12: Illustration of IBIS applied to 30 samples from $\mathcal{N}(\mu = 3.14, \sigma = 1)$. (a) Posterior approximation after $t = 1$ and $t = 29$ observations. (b) Effective sample size over time. The sudden jumps up occur whenever resampling is triggered, which happens when the ESS drops below 500. Generated by smc_ibis_1d.ipynb.

13.6.3.1 Example: IBIS for a 1d Gaussian

In this section, we give a simple example of IBIS applied to data from a 1d Gaussian, $y_t \sim \mathcal{N}(\mu = 3.14, \sigma = 1)$ for $t = 1 : 30$. The unknowns are $\boldsymbol{\theta} = (\mu, \sigma)$. The prior is $p(\boldsymbol{\theta}) = \mathcal{N}(\mu|0, 1)\text{Ga}(\sigma|a = 1, b = 1)$. We use IBIS with an adaptive RWMH kernel. We use $N = 20$ particles, each updated for $K = 50$ MCMC steps, so we collect 1000 samples per time step.

Figure 13.12a shows the approximate posterior after $t = 1$ and $t = 29$ time steps. We see that the posterior concentrates on the true values of $\mu = 3.14$ and $\sigma = 1$.

Figure 13.12b plots the ESS vs time. The number of particles is 1000, and resampling (and MCMC moves) is triggered whenever this drops below 500. We see that we only need to invoke MCMC updates 3 times.

13.6.4 Sampling rare events and extrema

Suppose we want to sample values from $\pi_0(\boldsymbol{\theta})$ conditioned on the event that $S(\boldsymbol{\theta}) > \lambda^*$, where S is some score or "fitness" function. If λ^* is in the tail of the score distribution, this corresponds to sampling a **rare event**, which can be hard.

One approach is to use SMC to sample from a sequence of distributions with gradually increasing thresholds:

$$\pi_t(\boldsymbol{\theta}) = \frac{1}{Z_t}\mathbb{I}\left(S(\boldsymbol{\theta}) \geq \lambda_t\right)\pi_0(\boldsymbol{\theta}) \tag{13.77}$$

with $\lambda_0 < \cdots < \lambda_T = \lambda^*$. We can then use likelihood tempering, where the "likelihood" is the function

$$G_t(\boldsymbol{\theta}_t) = \mathbb{I}\left(S(\boldsymbol{\theta}_t) \geq \lambda_t\right) \tag{13.78}$$

We can use SMC to generate samples from the final distribution π_T. We may also be interested in estimating

$$Z_T = p(S(\boldsymbol{\theta}) \geq \lambda_T) \tag{13.79}$$

where the probability is taken wrt $\pi_0(\boldsymbol{\theta})$.

We can adaptively set the thresholds λ_t as follows: at each step, sort the samples by their score, and set λ_t to the α'th highest quantile. For example, if we set $\alpha = 0.5$, we keep the top 50% fittest particles. This ensures the ESS equals the minimum threshold at each step. For details, see [Cér+12].

Note that this method is very similar to the **cross-entropy method** (Section 6.9.5). The difference is that CEM fits a parametric distribution (e.g., a Gaussian) to the particles at each step and samples from that, rather than using a Markov kernel.

13.6.5 SMC-ABC and likelihood-free inference

The term **likelihood-free inference** refers to estimating the parameters $\boldsymbol{\theta}$ of a black box from which we can sample data, $\boldsymbol{y} \sim p(\cdot|\boldsymbol{\theta})$, but where we cannot evaluate $p(\boldsymbol{y}|\boldsymbol{\theta})$ pointwise. Such models are called simulators, so this approach to inference is also called **simulation-based inference** (see e.g., [Nea+08; CBL20; Gou+96]). These models are also called **implicit models** (see Section 26.1).

If we want to approximate the posterior of a model with no known likelihood, we can use **Approximate Bayesian Computation** or **ABC** (see e.g., [Bea19; SFB18; Gut+14; Pes+21]). In this setting, we sample both parameters $\boldsymbol{\theta}$ and synthetic data $\boldsymbol{y}$ such that the synthetic data (generated from $\boldsymbol{\theta}$) is sufficiently close to the observed data $\boldsymbol{y}^*$, as judged by some distance score, $d(\boldsymbol{y}, \boldsymbol{y}^*) < \epsilon$. (For high dimensional problems, we typically require $d(\boldsymbol{s}(\boldsymbol{y}), \boldsymbol{s}(\boldsymbol{y}^*)) < \epsilon$, where $\boldsymbol{s}(\boldsymbol{y})$ is a low-dimensionary summary statistic of the data.)

In **SMC-ABC**, we gradually decrease the discrepancy ϵ to get a series of distributions as follows:

$$\pi_t(\boldsymbol{\theta}, \boldsymbol{y}) = \frac{1}{Z_t} \pi_0(\boldsymbol{\theta}) p(\boldsymbol{y}|\boldsymbol{\theta}) \mathbb{I}\left(d(\boldsymbol{y}, \boldsymbol{y}^*) < \epsilon_t\right) \tag{13.80}$$

where $\epsilon_0 > \epsilon_1 > \cdots$. This is similar to the rare event SMC samplers in Section 13.6.4, except that we can't directly evaluate the quality of a candidate $\boldsymbol{\theta}$, instead we must first convert it to data space and make the comparison there. For details, see [DMDJ12].

Although SMC-ABC is popular in some fields, such as genetics and epidemiology, this method is quite slow and does not scale to high dimensional problems. In such settings, a more efficient approach is to train a generative model to **emulate** the simulator; if this model is parametric with a tractable likelihood (e.g., a flow model), we can use the usual methods for posterior inference of its parameters (including gradient based methods like HMC). See e.g., [Bre+20a] for details.

13.6.6 SMC^2

We have seen how SMC can be a useful alternative to MCMC. However it requires that we can efficiently evaluate the likelihood ratio terms $\frac{\gamma_t(\boldsymbol{\theta}_t)}{\gamma_{t-1}(\boldsymbol{\theta}_t)}$. In cases where this is not possible (e.g., for latent variable models), we can use SMC (specifically the estimate $\hat{Z}_t$ in Equation (13.10)) as a subroutine to approximate these likelihoods. This is called **SMC^2**. For details, see [CP20b, Ch. 18].

13.6.7 Variational filtering SMC

One way to improve SMC is to learn a proposal distribution (e.g., using a neural network) such that the approximate posterior, $\hat{\pi}_T(\boldsymbol{z}_{1:T}; \boldsymbol{\phi}, \boldsymbol{\theta})$, is close to the target posterior, $\pi_T(\boldsymbol{z}_{1:T}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are

the model parameters, and $\boldsymbol{\phi}$ are the proposal parameters (which may depend on $\boldsymbol{\theta}$). One can show [Nae+18] that the KL divergence between these distributions can be bounded as follows:

$$0 \leq D_{\mathbb{KL}}\left(\mathbb{E}\left[\hat{\pi}_T(\boldsymbol{z}_{1:T})\right] \parallel \pi_T(\boldsymbol{z}_{1:T})\right) \leq -\mathbb{E}\left[\log \frac{\hat{Z}_T}{Z_T}\right] \tag{13.81}$$

where

$$Z_T(\boldsymbol{\theta}) = p_{\boldsymbol{\theta}}(\boldsymbol{y}_{1:T}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}) d\boldsymbol{z}_{1:T} \tag{13.82}$$

Hence

$$\mathbb{E}\left[\log \hat{Z}_T(\boldsymbol{\theta}, \boldsymbol{\phi})\right] \leq \mathbb{E}\left[\log Z_T(\boldsymbol{\theta})\right] = \log Z_T(\boldsymbol{\theta}) \tag{13.83}$$

Thus we can use SMC sampling to compute an unbiased approximation to $\mathbb{E}\left[\log \hat{Z}_T(\boldsymbol{\theta}, \boldsymbol{\phi})\right]$, which is a lower bound on the evidence (log marginal likelihood).

We can now maximize this lower bound wrt $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD, as a way to learn both proposals and the model. Unfortunately, computing the gradient of the bound is tricky, since the resampling step is non-differentiable. However, in practice one can ignore the dependence of the resampling operator on the parameters, or one can use differentiable approximations (see e.g., [Ros+22]). This overall approach was independently proposed in several papers: the **FIVO** (filtering variational objective) paper [Mad+17], the **variational SMC** paper [Nae+18] and the **auto-encoding SMC** paper [Le+18].

13.6.8 Variational smoothing SMC

The methods in Section 13.6.7 use SMC in which the target distributions are defined to be the filtered distributions, $\pi_t(\boldsymbol{z}_{1:t}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}|\boldsymbol{y}_{1:t})$; this is called **filtering SMC**. Unfortunately, this can work poorly when fitting models to offline sequence data, since at time t, all future observations are ignored in the objective, no matter how good the proposal. This can create situations where future observations are unlikely given the current set of sampled trajectories, which can result in particle impoverishment and high variance in the estimate of the lower bound.

Recently, a new method called **SIXO** (smoothing inference with twisted objectives) was proposed in [Law+22] that uses the smoothing distributions as targets, $\pi_t(\boldsymbol{z}_{1:t}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}|\boldsymbol{y}_{1:T})$, to create a much lower variance variational lower bound. Of course it is impossible to directly compute this posterior, but we can approximate it using **twisted particle filters** [WL14a; AL+16]. In this approach, we approximate the (unnormalized) posterior using

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:T}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) \tag{13.84}$$

$$= p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t) \tag{13.85}$$

$$\approx p_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}) r_{\boldsymbol{\psi}}(\boldsymbol{y}_{t+1:T}, \boldsymbol{z}_t) \tag{13.86}$$

where $r_{\boldsymbol{\psi}}(\boldsymbol{y}_{t+1:T}, \boldsymbol{z}_t) \approx p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t)$ is the **twisting function**, which acts as a "**lookahead function**".

One way to approximate the twisting function is to note that

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{y}_{t+1:T})p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t)} \propto \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{y}_{t+1:T})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t)} \tag{13.87}$$

where we drop terms that are independent of $\boldsymbol{z}_t$ since such terms will cancel out when we normalize the sampling weights. We can approximate the density ratio using the binary classifier method of Section 2.7.5. To do this, we define one distribution to be $p_1 = p_{\boldsymbol{\theta}}(\boldsymbol{z}_t, \boldsymbol{y}_{t+1:T})$ and the other to be $p_2 = p_{\boldsymbol{\theta}}(\boldsymbol{z}_t)p_{\boldsymbol{\theta}}(\boldsymbol{y}_{t+1:T})$, so that $p_1/p_2 = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{y}_{t+1:T})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t)}$. We can easily draw a sample $(\boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}) \sim p_{\boldsymbol{\theta}}$ using ancestral sampling, from which we can compute $(\boldsymbol{z}_t, \boldsymbol{y}_{t+1:T}) \sim p_1$ by marginalization. We can also sample a fresh sequence from $(\tilde{\boldsymbol{z}}_{1:T}, \tilde{\boldsymbol{y}}_{1:T}) \sim p_{\boldsymbol{\theta}}$ from which we can compute $(\tilde{\boldsymbol{z}}_t, \tilde{\boldsymbol{y}}_{t+1:T}) \sim p_2$ by marginalization. We then use $(\boldsymbol{z}_t, \boldsymbol{y}_{t+1:T})$ as a positive example and $(\tilde{\boldsymbol{z}}_t, \tilde{\boldsymbol{y}}_{t+1:T})$ as a negative example when training the binary classifier, $r_{\boldsymbol{\psi}}(\boldsymbol{y}_{t+1:T}, \boldsymbol{z}_t)$.

Once we have updated the twisting parameters $\boldsymbol{\psi}$, we can rerun SMC to get a tighter lower bound on the log marginal likelihood, which we can then optimize wrt the model parameters $\boldsymbol{\theta}$ and proposal parameters $\boldsymbol{\phi}$. Thus the overall method is a stochastic variational EM-like method for optimziing the bound

$$\mathcal{L}_{\text{SIXO}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{y}_{1:T}) \triangleq \mathbb{E}\left[\log \hat{Z}_{\text{SIXO}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{y}_{1:T})\right] \tag{13.88}$$

$$\leq \log \mathbb{E}\left[\hat{Z}_{\text{SIXO}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{y}_{1:T})\right] = \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_{1:T}) \tag{13.89}$$

In [Law+22] they prove the following: suppose the true model p^* is an SSM in which the optimal proposal function for the model satisfies $p^*(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{y}_{1:T}) \in \mathcal{Q}$, and the optimal lookahead function for the model satisfies $p^*(\boldsymbol{y}_{t+1:T}|\boldsymbol{z}_t) \in \mathcal{R}$. Furthermore, assume the SIXO objective has a unique maximizer. Then, at the optimum, we have that the learned proposal $q_{\boldsymbol{\phi}^*}(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{y}_{1:T}) \in \mathcal{Q}$ is equal to the optimal proposal, the learned twisting function $r_{\boldsymbol{\psi}^*}(\boldsymbol{y}_{t+1:T}, \boldsymbol{z}_t) \in \mathcal{R}$ is equal to the optimal lookahead, and the lower bound is tight (i.e., $\mathcal{L}_{\text{SIXO}}(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*, \boldsymbol{\psi}^*) = p^*(\boldsymbol{y}_{1:T})$) for any number of samples $N_s \geq 1$ and for any kind of SSM p^*. (This is in contrast to the FIVO bound, whiere the bound does not usually become tight.)

Part III

Prediction

14 Predictive models: an overview

14.1 Introduction

The vast majority of machine learning is concerned with tackling a single problem, namely learning to predict outputs $\boldsymbol{y}$ from inputs $\boldsymbol{x}$ using some function f that is estimated from a labeled training set $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$, for $\boldsymbol{x}_n \in \mathcal{X} \subseteq \mathbb{R}^D$ and $\boldsymbol{y}_n \in \mathcal{Y} \subseteq \mathbb{R}^C$. We can model our uncertainty about the correct output for a given input using a conditional probability model of the form $p(\boldsymbol{y}|f(\boldsymbol{x}))$. When $\mathcal{Y}$ is a discrete set of labels, this is called (in the ML literature) a **discriminative model**, since it lets us discriminate (distinguish) between the different possible values of $\boldsymbol{y}$. If the output is real-valued, $\mathcal{Y} = \mathbb{R}$, this is called a **regression model**. (In the statistics literature, the term "regression model" is used in both cases, even if $\mathcal{Y}$ is a discrete set.) We will use the more generic term "**predictive model**" to refer to such models.

A predictive model can be considered as a special case of a conditional generative model (discussed in Chapter 20). In a predictive model, the output is usually low dimensional, and there is a single best answer that we want to predict. However, in most generative models, the output is usually high dimensional, such as images or sentences, and there may be many correct outputs for any given input. We will discuss a variety of types of predictive model in Section 14.1.1, but we defer the details to subsequent chapters. The rest of this chapter then discusses issues that are relevant to all types of predictive model, regardless of the specific form, such as evaluation.

14.1.1 Types of model

There are many different kinds of predictive model $p(\boldsymbol{y}|\boldsymbol{x})$. The biggest distinction is between **parametric models**, that have a fixed number of parameters independent of the size of the training set, and **non-parametric models** that have a variable number of parameters that grows with the size of the training set. Non-parametric models are usually more flexible, but can be slower to use for prediction. Parametric models are usually less flexible, but are faster to use for prediction.

Most non-parametric models are based on comparing a test input $\boldsymbol{x}$ to some or all of the stored training examples $\{\boldsymbol{x}_n, n = 1 : N\}$, using some form of similarity, $s_n = \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_n) \geq 0$, and then predicting the output using some weighted combination of the training labels, such as $\hat{\boldsymbol{y}} = \sum_{n=1}^{N} s_n \boldsymbol{y}_n$. A typical example is a Gaussian process, which we discuss in Chapter 18. Other examples, such as K-nearest neighbor models, are discussed in the prequel to this book, [Mur22].

Most parametric models have the form $p(\boldsymbol{y}|\boldsymbol{x}) = p(\boldsymbol{y}|f(\boldsymbol{x}; \boldsymbol{\theta}))$, where f is some kind of function that predicts the parameters (e.g., the mean, or logits) of the output distribution (e.g., Gaussian or categorical). There are many kinds of function we can use. If f is a linear function of $\boldsymbol{\theta}$ (i.e.,

$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x})$ for some *fixed* feature transformation $\boldsymbol{\phi}$), then the model is called a generalized linear model or GLM, which we discuss in Chapter 15. If f is a non-linear, but differentiable, function of $\boldsymbol{\theta}$ (e.g., $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}_2^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}_1)$ for some learnable function $\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}_1)$), then it is common to represent f using a neural network (Chapter 16). Other types of predictive model, such as decision trees and random forests, are discussed in the prequel to this book, [Mur22].

14.1.2 Model fitting using ERM, MLE and MAP

In this section, we briefly discuss some methods used for fitting (parametric) models. The most common approach is to use **maximum likelihood estimation** or **MLE**, which amounts to solving the following optimization problem:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} p(\mathcal{D}|\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\mathcal{D}|\boldsymbol{\theta}) \tag{14.1}$$

If the dataset is N iid data samples, the likelihood decomposes into a product of terms, $p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta})$. Thus we can instead minimize the following (scaled) **negative log likelihood**:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \frac{1}{N}\sum_{n=1}^N [-\log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta})] \tag{14.2}$$

We can generalize this by replacing the **log loss** $\ell_n(\boldsymbol{\theta}) = -\log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta})$ with a more general loss function to get

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} r(\boldsymbol{\theta}) \tag{14.3}$$

where $r(\boldsymbol{\theta})$ is the **empirical risk**

$$r(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^N \ell_n(\boldsymbol{\theta}) \tag{14.4}$$

This approach is called **empirical risk minimization** or **ERM**.

ERM can easily result in **overfitting**, so it is common to add a penalty or regularizer term to get

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} r(\boldsymbol{\theta}) + \lambda C(\boldsymbol{\theta}) \tag{14.5}$$

where $\lambda \geq 0$ controls the degree of regularization, and $C(\boldsymbol{\theta})$ is some complexity measure. If we use log loss, and we define $C(\boldsymbol{\theta}) = -\log \pi_0(\boldsymbol{\theta})$, where $\pi_0(\boldsymbol{\theta})$ is some prior distribution, and we use $\lambda = 1$, we recover the **MAP estimate**

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log \pi_0(\boldsymbol{\theta}) \tag{14.6}$$

This can be solved using standard optimization methods (see Chapter 6).

14.1.3 Model fitting using Bayes, VI and generalized Bayes

Another way to prevent overfitting is to estimate a *probability distribution over parameters*, $q(\boldsymbol{\theta})$, instead of a point estimate. That is, we can try to estimate the ERM in expectation:

$$\hat{q} = \underset{q \in \mathcal{P}(\Theta)}{\operatorname{argmin}} \, \mathbb{E}_{q(\boldsymbol{\theta})} \left[r(\boldsymbol{\theta}) \right] \tag{14.7}$$

If $\mathcal{P}(\Theta)$ is the space of all probability distributions over parameters, then the solution will converge to a delta function that puts all its probability on the MLE. Thus this approach, on its own, will not prevent overfitting. However, we can regularize the problem by preventing the distribution from moving too far from the prior. If we measure the divergence between q and the prior using KL divergence, we get

$$\hat{q} = \underset{q \in \mathcal{P}(\Theta)}{\operatorname{argmin}} \, \mathbb{E}_{q(\boldsymbol{\theta})} \left[r(\boldsymbol{\theta}) \right] + \frac{1}{\lambda} D_{\mathbb{KL}} \left(q \parallel \pi_0 \right) \tag{14.8}$$

The solution to this problem is known as the **Gibbs posterior**, and is given by the following:

$$\hat{q}(\boldsymbol{\theta}) = \frac{e^{-\lambda r(\boldsymbol{\theta})} \pi_0(\boldsymbol{\theta})}{\int e^{-\lambda r(\boldsymbol{\theta}')} \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \tag{14.9}$$

This is widely used in the **PAC-Bayes** community (see e.g., [Alq21].

Now suppose we use log loss, and set $\lambda = N$, to get

$$\hat{q}(\boldsymbol{\theta}) = \frac{e^{-\sum_{n=1}^{N} \log p(\boldsymbol{y}_n | \boldsymbol{x}_n, \boldsymbol{\theta})} \pi_0(\boldsymbol{\theta})}{\int e^{-\sum_{n=1}^{N} \log p(\boldsymbol{y}_n | \boldsymbol{x}_n, \boldsymbol{\theta}')} \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \tag{14.10}$$

Then the resulting distribution is equivalent to the Bayes posterior:

$$\hat{q}(\boldsymbol{\theta}) = \frac{p(\mathcal{D}|\boldsymbol{\theta}) \pi_0(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta}') \pi_0(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \tag{14.11}$$

Often computing the Bayes posterior is intractable. We can simplify the problem by restricting attention to a limited family of distributions, $\mathcal{Q}(\Theta) \subset \mathcal{P}(\Theta)$. This gives rise to the following objective:

$$\hat{q} = \underset{q \in \mathcal{Q}(\Theta)}{\operatorname{argmin}} \, \mathbb{E}_{q(\boldsymbol{\theta})} \left[-\log p(\mathcal{D}|\boldsymbol{\theta}) \right] + D_{\mathbb{KL}} \left(q \parallel \pi_0 \right) \tag{14.12}$$

This is known as **variational inference**; see Chapter 10 for details. (See also Section 6.7, where we discuss the Bayesian learning rule.)

We can generalize this by replacing the negative log likelihood with a general risk, $r(\boldsymbol{\theta})$. Furthermore, we can replace the KL with a general divergence, $D(q||\pi_0)$, which we can weight using a general λ. This gives rise to the following objective:

$$\hat{q} = \underset{q \in \mathcal{Q}(\Theta)}{\operatorname{argmin}} \, \mathbb{E}_{q(\boldsymbol{\theta})} \left[r(\boldsymbol{\theta}) \right] + \lambda D(q||\pi_0) \tag{14.13}$$

This is called **generalized Bayesian inference** [BHW16; KJD19; KJD21].

14.2 Evaluating predictive models

In this section we discuss how to evaluate the quality of a trained discriminative model.

14.2.1 Proper scoring rules

It is common to measure performance of a predictive model using a **proper scoring rule** [GR07a], which is defined as follows. Let $S(p_{\boldsymbol{\theta}}, (y, \boldsymbol{x}))$ be the score for predictive distribution $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ when given an event $y|\boldsymbol{x} \sim p^*(y|\boldsymbol{x})$, where p^* is the true conditional distribution. (If we want to evaluate a Bayesian model, where we marginalize out $\boldsymbol{\theta}$ rather than condition on it, we just replace $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ with $p(y|\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(y|\boldsymbol{x})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}$.) The expected score is defined by

$$S(p_{\boldsymbol{\theta}}, p^*) = \int p^*(\boldsymbol{x})p^*(y|\boldsymbol{x})S(p_{\boldsymbol{\theta}}, (y, \boldsymbol{x}))dyd\boldsymbol{x} \tag{14.14}$$

A proper scoring rule is one where $S(p_{\boldsymbol{\theta}}, p^*) \leq S(p^*, p^*)$, with equality iff $p_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = p^*(y|\boldsymbol{x})$. Thus maximizing such a proper scoring rule will force the model to match the true probabilities.

The log-likelihood, $S(p_{\boldsymbol{\theta}}, (y, \boldsymbol{x})) = \log p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$, is a proper scoring rule. This follows from Gibbs inequality:

$$S(p_{\boldsymbol{\theta}}, p^*) = \mathbb{E}_{p^*(\boldsymbol{x})p^*(y|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(y|\boldsymbol{x})\right] \leq \mathbb{E}_{p^*(\boldsymbol{x})p^*(y|\boldsymbol{x})}\left[\log p^*(y|\boldsymbol{x})\right] \tag{14.15}$$

Therefore minimizing the NLL (aka log loss) should result in well-calibrated probabilities. However, in practice, log-loss can over-emphasize tail probabilities [QC+06].

A common alternative is to use the **Brier score** [Bri50], which is defined as follows:

$$S(p_{\boldsymbol{\theta}}, (y, \boldsymbol{x})) \triangleq \frac{1}{C}\sum_{c=1}^{C}(p_{\boldsymbol{\theta}}(y = c|\boldsymbol{x}) - \mathbb{I}(y = c))^2 \tag{14.16}$$

This is just the squared error of the predictive distribution $\boldsymbol{p} = p(1 : C|\boldsymbol{x})$ compared to the one-hot label distribution $\boldsymbol{y}$. Since it based on squared error, the Brier score is less sensitive to extremely rare or extremely common classes. The Brier score is also a proper scoring rule.

14.2.2 Calibration

A model whose predicted probabilities match the empirical frequencies is said to be **calibrated** [Daw82; NMC05; Guo+17]. For example, if a classifier predicts $p(y = c|\boldsymbol{x}) = 0.9$, then we expect this to be the true label about 90% of the time. A well-calibrated model is useful to avoid making the wrong decision when the outcome is too uncertain. In the sections below, we discuss some ways to measure and improve calibration.

14.2.2.1 Expected calibration error

To assess calibration, we divide the predicted probabilities into a finite set of bins or buckets, and then assess the discrepancy between the empirical probability and the predicted probability by counting. More precisely, suppose we have B bins. Let $\mathcal{B}_b$ be the set of indices of samples whose prediction

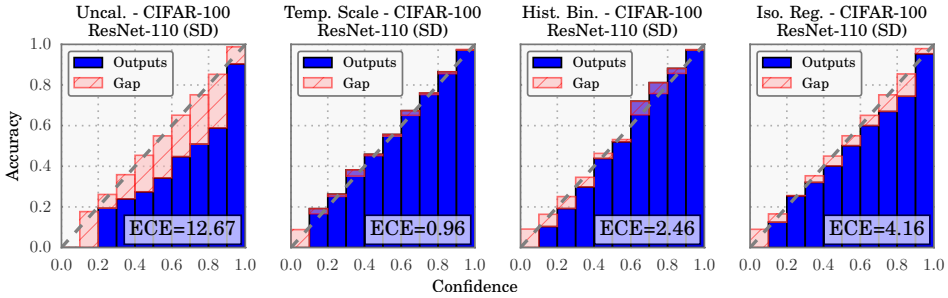


Figure 14.1: Reliability diagrams for the ResNet CNN image classifier [He+16b] applied to CIFAR-100 dataset. ECE is the expected calibration error, and measures the size of the red gap. Methods from left to right: original probabilities; after temperature scaling; after histogram binning; after isotonic regression. From Figure 4 of [Guo+17]. Used with kind permission of Chuan Guo.

confidence falls into the interval $I_b = (\frac{b-1}{B}, \frac{b}{B}]$. Here we use uniform bin widths, but we could also define the bins so that we can get an equal number of samples in each one.

Let $f(\boldsymbol{x})_c = p(y = c|\boldsymbol{x})$, $\hat{y}_n = \text{argmax}_{c \in \{1,\ldots,C\}} f(\boldsymbol{x}_n)_c$, and $\hat{p}_n = \max_{c \in \{1,\ldots,C\}} f(\boldsymbol{x}_n)_c$. The accuracy within bin b is defined as

$$\text{acc}(\mathcal{B}_b) = \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \mathbb{I}\left(\hat{y}_n = y_n\right) \tag{14.17}$$

The average confidence within this bin is defined as

$$\text{conf}(\mathcal{B}_b) = \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \hat{p}_n \tag{14.18}$$

If we plot accuracy vs confidence, we get a **reliability diagram**, as shown in Figure 14.1. The gap between the accuracy and confidence is shown in the red bars. We can measure this using the **expected calibration error** (**ECE**) [NCH15]:

$$\text{ECE}(f) = \sum_{b=1}^{B} \frac{|\mathcal{B}_b|}{B} \left|\text{acc}(\mathcal{B}_b) - \text{conf}(\mathcal{B}_b)\right| \tag{14.19}$$

In the multiclass case, the ECE only looks at the error of the MAP (top label) prediction. We can extend the metric to look at all the classes using the **marginal calibration error**, proposed in [KLM19]:

$$\text{MCE} = \sum_{c=1}^{C} w_c \mathbb{E}\left[\left(p(Y = c|f(\boldsymbol{x})_c) - f(\boldsymbol{x})_c\right)^2\right] \tag{14.20}$$

$$= \sum_{c=1}^{C} w_c \sum_{b=1}^{B} \frac{|\mathcal{B}_{b,c}|}{B} \left(\text{acc}(\mathcal{B}_{b,c}) - \text{conf}(\mathcal{B}_{b,c})\right)^2 \tag{14.21}$$

where $\mathcal{B}_{b,c}$ is the b'th bin for class c, and $w_c \in [0,1]$ denotes the importance of class c. (We can set $w_c = 1/C$ if all classes are equally important.) In [Nix+19], they call this metric **static calibration error**; they show that certain methods that have good ECE may have poor MCE. Other multi-class calibration metrics are discussed in [WLZ19].

14.2.2.2 Improving calibration

In principle, training a classifier so it optimizes a proper scoring rule (such as NLL) should automatically result in a well-calibrated classifier. In practice, however, unbalanced datasets can result in poorly calibrated predictions. Below we discuss various ways for improving the calibration of probabilistic classifiers, following [Guo+17].

14.2.2.3 Platt scaling

Let z be the log-odds, or logit, and $p = \sigma(z)$, produced by a probabilistic binary classifier. We wish to convert this to a more calibrated value q. The simplest way to do this is known as **Platt scaling**, and was proposed in [Pla00]. The idea is to compute $q = \sigma(az + b)$, where a and b are estimated via maximum likelihood on a validation set.

In the multiclass case, we can extend Platt scaling by using matrix scaling: $\boldsymbol{q} = \text{softmax}(\mathbf{W}\boldsymbol{z} + \boldsymbol{b})$, where we estimate $\mathbf{W}$ and $\boldsymbol{b}$ via maximum likelihood on a validation set. Since $\mathbf{W}$ has $K \times K$ parameters, where K is the number of classes, this method can easily overfit, so in practice we restrict $\mathbf{W}$ to be diagonal.

14.2.2.4 Nonparametric (histogram) methods

Platt scaling makes a strong assumption about how the shape of the calibration curve. A more flexible, nonparametric, method is to partion the predicted probabilities into bins, p_m, and to estimate an empirical probability q_m for each such bin; we then replace p_m with q_m; this is known as **histogram binning** [ZE01a]. We can regularize this method by requiring that $q = f(p)$ be a piecewise constant, monotonically non-decreasing function; this is known as **isotonic regression** [ZE01a]. An alternative approach, known as the **scaling-binning calibrator**, is to apply a scaling method (such as Platt scaling), and then to apply histogram binning to that. This has the advantage of using the average of the scaled probabilities in each bin instead of the average of the observed binary labels (see Figure 14.2). In [KLM19], they prove that this results in better calibration, due to the lower variance of the estimator.

In the multiclass case, $\boldsymbol{z}$ is the vector of logits, and $\boldsymbol{p} = \text{softmax}(\boldsymbol{z})$ is the vector of probabilities. We wish to convert this to a better calibrated version, $\boldsymbol{q}$. [ZE01b] propose to extend histogram binning and isotonic regression to this case by applying the above binary method to each of the K one-vs-rest problems, where K is the number of classes. However, this requires K separate calibration models, and results in an unnormalized probability distribution.

14.2.2.5 Temperature scaling

In [Guo+17], they noticed empirically that the diagonal version of Platt scaling, when applied to a variety of DNNs, often ended learning a vector of the form $\boldsymbol{w} = (c, c, \ldots, c)$, for some constant c. This suggests a simpler form of scaling, which they call **temperature scaling**: $\boldsymbol{q} = \text{softmax}(\boldsymbol{z}/T)$,

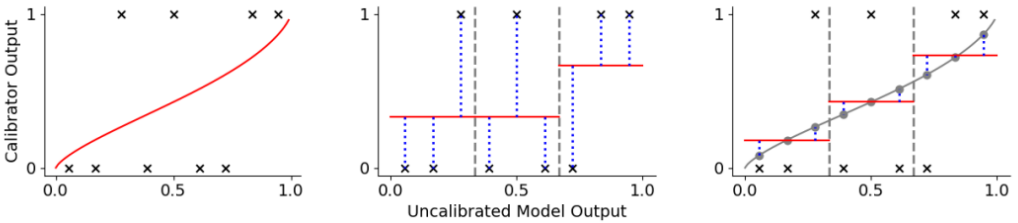


Figure 14.2: Visualization of 3 different approaches to calibrating a binary probabilistic classifier. Black crosses are the observed binary labels, red lines are the calibrated outputs. (a) Platt scaling. (b) Histogram binning with 3 bins. The output in each bin is the average of the binary labels in each bin. (c) The scaling-binning calibrator. This first applies Platt scaling, and then computes the average of the scaled points (gray circles) in each bin. From Figure 1 of [KLM19]. Used with kind permission of Ananya Kumar.

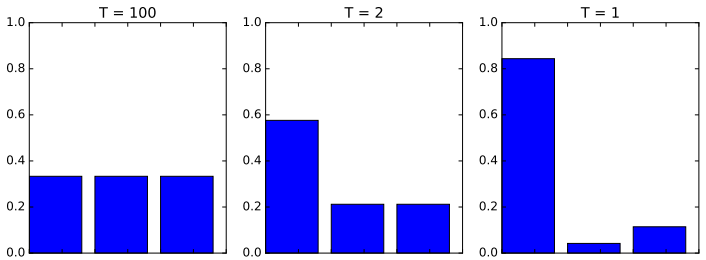


Figure 14.3: Softmax distribution $\text{softmax}(\boldsymbol{a}/T)$*, where* $\boldsymbol{a} = (3, 0, 1)$*, at temperatures of* $T = 100$*,* $T = 2$ *and* $T = 1$*. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is "spiky", with most of its mass on the largest element. Generated by softmax_plot.ipynb.*

where $T > 0$ is a temperature parameter, which can be estimated by maximum likelihood on the validation set. The effect of this temperature parameter is to make the distribution less peaky, as shown in Figure 14.3. [Guo+17] show empirically that this method produces the lowest ECE on a variety of DNN classification problems (see Figure 14.1 for a visualization). Furthermore, it is much simpler and faster than the other methods.

Note that Platt scaling and temperature scaling do not affect the identity of the most probable class label, so these methods have no impact on classification accuracy. However, they do improve calibration performance. A more recent multi-class calibration method is discussed in [Kul+19].

14.2.2.6 Label smoothing

When training classifiers, we usually represent the true target label as a one-hot vector, say $\boldsymbol{y} = (0, 1, 0)$ to represent class 2 out of 3. We can improve results if we "spread" some of the probability mass across all the bins. For example we may use $\boldsymbol{y} = (0.1, 0.8, 0.1)$. This is called **label smoothing** and

often results in better-calibrated models [MKH19].

14.2.2.7 Bayesian methods

Bayesian approaches to fitting classifiers often result in more calibrated predictions, since they represent uncertainty in the parameters. See Section 17.3.8 for an example. However, [Ova+19] shows that well-calibrated models (even Bayesian ones) often become mis-calibrated when applied to inputs that come from a different distribution (see Section 19.2 for details).

14.2.3 Beyond evaluating marginal probabilities

Calibration (Section 14.2.2) focuses on assessing properties of the marginal predictive distribution $p(y|\boldsymbol{x})$. But this can sometimes be insufficient to distinguish between a good and bad model, especially in the context of online learning and sequential decision making, as pointed out in [Lu+22; Osb+21; WSG21; KKG22]. For example, consider two learning agents who observe a sequence of coin tosses. Let the outcome at time t be $Y_t \sim \text{Ber}(\theta)$, where θ is the unknown parameter. Agent 1 believes $\theta = 2/3$, whereas agent 2 believes either $\theta = 0$ or $\theta = 1$, but is not sure which, and puts probabilities $1/3$ and $2/3$ on these events. Thus both agents, despite having different models, make identical predictions for the next outcome: $p(Y_1^i = 0) = 1/3$ for agents $i = 1, 2$. However, the predictions of the two agents about a *sequence* of τ future outcomes is very different: In particular, agent 1 predicts each individual coin toss is a random Bernoulli event, where the probability is due to irreducible noise or **aleatoric uncertainty**:

$$p(Y_1^1 = 0, \ldots, Y_\tau^1 = 0) = \frac{1}{3^\tau} \tag{14.22}$$

By contrast, agent 2 predicts that the sequence will either be all heads or all tails, where the probability is induced by **epistemic uncertainty** about the true parameters:

$$p(Y_1^2 = y_1, \ldots, Y_\tau^2 = y_\tau) = \begin{cases} 1/3 & \text{if } y_1 = \cdots = y_\tau = 0 \\ 2/3 & \text{if } y_1 = \cdots = y_\tau = 1 \\ 0 & \text{otherwise} \end{cases} \tag{14.23}$$

The difference in beliefs between these agents will impact their behavior. For example, in a casino, agent 1 incurs little risk on repeatedly betting on heads in the long run, but for agent 2, this would be a very unwise strategy, and some initial information gathering (exploration) would be worthwhile.

Based on the above, we see that it is useful to evaluate *joint* predictive distributions when assessing predictive models. In [Lu+22; Osb+21] they propose to evaluate the posterior predictive distributions over τ outcomes $\boldsymbol{y} = Y_{T+1:T+\tau}$, given a set of τ inputs $\boldsymbol{x} = X_{T:T+\tau-1}$, and the past T data samples, $\mathcal{D}_T = \{(X_t, Y_{t+1}) : t = 0, 1, \ldots, T-1\}$. The Bayes optimal predictive distribution is

$$P_T^B = p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T) \tag{14.24}$$

This is usually intractable to compute. Instead the agent will use an approximate distribution, known as a **belief state**, which we denote by

$$Q_T = p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T) \tag{14.25}$$

The natural performance metric is the KL between these distributions. Since this depend on the inputs $\boldsymbol{x}$ and $\mathcal{D}_T = (X_{0:T-1}, Y_{1:T})$, we will averaged the KL over these values, which are drawn iid from the true data generating distribution, which we denote by

$$P(X, Y, \mathcal{E}) = P(X|\mathcal{E})P(Y|X, \mathcal{E})P(\mathcal{E}) \tag{14.26}$$

where $\mathcal{E}$ is the true but unknown environment. Thus we define our metric as

$$d^{KL}_{B,Q} = \mathbb{E}_{P(\boldsymbol{x},\mathcal{D}_T)}\left[D_{\mathbb{KL}}\left(P^B(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T) \parallel Q(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T)\right)\right] \tag{14.27}$$

where

$$P(\boldsymbol{x}, \mathcal{D}_T, \mathcal{E}) = P(\mathcal{E}) \underbrace{\left[\prod_{t=0}^{T-1} P(X_t|\mathcal{E})P(Y_{t+1}|X_t, \mathcal{E})\right]}_{P(\mathcal{D}_T|\mathcal{E})} \underbrace{\left[\prod_{t=T}^{T+\tau-1} P(x_t|\mathcal{E})\right]}_{P(\boldsymbol{x}|\mathcal{E})} \tag{14.28}$$

and $P(\boldsymbol{x}, \mathcal{D}_T)$ marginalizes this over environments.

Unfortunately, it is usually intractable to compute the exact Bayes posterior, P^B_T, so we cannot evaluate $d^{KL}_{B,Q}$. However, in Section 14.2.3.1, we show that

$$d^{KL}_{B,Q} = d^{KL}_{\mathcal{E},Q} - \mathbb{I}(\mathcal{E}; \boldsymbol{y}|\mathcal{D}_T, \boldsymbol{x}) \tag{14.29}$$

where the second term is a constant wrt the agent, and the first term is given by

$$d^{KL}_{\mathcal{E},Q} = \mathbb{E}_{P(\boldsymbol{x},\mathcal{D}_T,\mathcal{E})}\left[D_{\mathbb{KL}}\left(P(\boldsymbol{y}|\boldsymbol{x}, \mathcal{E}) \parallel Q(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T)\right)\right] \tag{14.30}$$

$$= \mathbb{E}_{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})P(\boldsymbol{x},\mathcal{D}_T,\mathcal{E})}\left[\log \frac{P(\boldsymbol{y}|\boldsymbol{x}, \mathcal{E})}{Q(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_T)}\right] \tag{14.31}$$

Hence if we rank agents in terms of $d^{KL}_{\mathcal{E},Q}$, it will give the same results as ranking them by $d^{KL}_{B,Q}$.

To compute $d^{KL}_{\mathcal{E},Q}$ in practice, we can use a Monte Carlo approximation: we just have to sample J environments, $\mathcal{E}^j \sim P(\mathcal{E})$, sample a training set $\mathcal{D}_T$ from each environment, $\mathcal{D}^j_T \sim P(\mathcal{D}_T|\mathcal{E}^j)$, and then sample N data vectors of length τ, $(\boldsymbol{x}^j_n, \boldsymbol{y}^j_n) \sim P(X_{T:T+\tau-1}, Y_{T+1:T+\tau}|\mathcal{E}^j)$. We can then compute

$$\hat{d}^{KL}_{\mathcal{E},Q} = \frac{1}{JN}\sum_{j=1}^{J}\sum_{n=1}^{N}\left[\log P(\boldsymbol{y}^j_n|\boldsymbol{x}^j_n, \mathcal{E}^j) - \log Q(\boldsymbol{y}^j_n|\boldsymbol{x}^j_n, \mathcal{D}^j_T)\right] \tag{14.32}$$

where

$$p_{jn} = P(\boldsymbol{y}^j_n|\boldsymbol{x}^j_n, \mathcal{E}^j) = \prod_{t=T}^{T+\tau-1} P(Y^j_{n,t+1}|X^j_{n,t}, \mathcal{E}^j) \tag{14.33}$$

$$q_{jn} = Q(\boldsymbol{y}^j_n|\boldsymbol{x}^j_n, \mathcal{D}^j_T) = \int Q(\boldsymbol{y}^j_n|\boldsymbol{x}^j_n, \boldsymbol{\theta})Q(\boldsymbol{\theta}|\mathcal{D}^j_T)d\boldsymbol{\theta} \tag{14.34}$$

$$\approx \frac{1}{M}\sum_{m=1}^{M}\prod_{t=T}^{T+\tau-1} Q(Y^j_{n,t+1}|X^j_{n,t}, \boldsymbol{\theta}^j_m) \tag{14.35}$$

where $\boldsymbol{\theta}_m^j \sim Q(\boldsymbol{\theta}|\mathcal{D}_T^j)$ is a sample from the agent's posterior over the environment.

The above assumes that $P(Y|X)$ is known; this will be the case if we use a synthetic data generator, as in the the "neural testbed" in [Osb+21]. If we just have an J empirical distributions for $P^j(X,Y)$, we can replace the KL with the cross entropy, which only differs by an additive constant:

$$d_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\boldsymbol{x},\mathcal{D}_T,\mathcal{E})}\left[D_{\mathbb{KL}}\left(P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E}) \parallel Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)\right)\right] \tag{14.36}$$

$$= \underbrace{\mathbb{E}_{P(\boldsymbol{x},\boldsymbol{y},\mathcal{E})}\left[\log P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})\right]}_{\text{const}} - \underbrace{\mathbb{E}_{P(\boldsymbol{x},\boldsymbol{y},\mathcal{D}_T|\mathcal{E})P(\mathcal{E})}\left[\log Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)\right]}_{d_{\mathcal{E},Q}^{CE}} \tag{14.37}$$

where the latter term is just the empirical negative log likelihood (NLL) of the agent on samples from the environment. Hence if we rank agents in terms of their NLL or cross entropy $d_{\mathcal{E},Q}^{CE}$ we will get the same results as ranking them by $d_{\mathcal{E},Q}^{KL}$, which will in turn give the same results as ranking them by $d_{B,Q}^{KL}$.

In practice we can approximate the cross entropy as follows:

$$\hat{d}_{\mathcal{E},Q}^{CE} = -\frac{1}{JN}\sum_{j=1}^{J}\sum_{n=1}^{N}\log Q(\boldsymbol{y}_n^j|\boldsymbol{x}_n^j,\mathcal{D}_T^j) \tag{14.38}$$

where $\mathcal{D}_T^j \sim P^j$, and $(\boldsymbol{x}_n^j,\boldsymbol{y}_n^j) \sim P^j$.

An alternative to estimating the KL or NLL is to evaluate the joint predictive accuracy by using it in a downstream task. In [Osb+21], they show that good predictive accuracy (for $\tau > 1$) correlates with good performance on a bandit problem (see Section 34.4). In [WSG21] they show that good predictive accuracy (for $\tau > 1$) results in good performance on a transductive active learning task.

14.2.3.1 Proof of claim

We now prove Equation (14.29), based on [Lu+21a]. First note that

$$d_{\mathcal{E},Q}^{KL} = \mathbb{E}_{P(\boldsymbol{x},\mathcal{D}_T,\mathcal{E})P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})}\left[\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})}{Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}\right] \tag{14.39}$$

$$= \mathbb{E}\left[\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}{Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}\right] + \mathbb{E}\left[\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})}{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}\right] \tag{14.40}$$

For the first term in Equation (14.40) we have

$$\mathbb{E}\left[\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}{Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}\right] = \sum P(\boldsymbol{x},\boldsymbol{y},\mathcal{D}_T)\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}{Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)} \tag{14.41}$$

$$= \sum P(\boldsymbol{x},\mathcal{D}_T)\sum P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)\log\frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}{Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)} \tag{14.42}$$

$$= \mathbb{E}_{P(\boldsymbol{x},\mathcal{D}_T)}\left[D_{\mathbb{KL}}\left(P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T) \parallel Q(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)\right)\right] = d_{B,Q}^{KL} \tag{14.43}$$

We now show that the second term in Equation (14.40) reduces to the mutual information. We exploit the fact that

$$P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E}) = P(\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x},\mathcal{E}) = \frac{P(\mathcal{E},\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x})}{P(\mathcal{E}|\mathcal{D}_T,\boldsymbol{x})} \tag{14.44}$$

Figure 14.4: Prediction set examples on Imagenet. We show three progressively more difficult examples of the class fox squirrel and the prediction sets generated by conformal prediction. From Figure 1 of [AB21]. Used with kind permission of Anastasios Angelopoulos.

since $\mathcal{D}_T$ has no new information in beyond $\mathcal{E}$. From this we get

$$\mathbb{E}\left[\log \frac{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{E})}{P(\boldsymbol{y}|\boldsymbol{x},\mathcal{D}_T)}\right] = \mathbb{E}\left[\log \frac{P(\mathcal{E},\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x})/P(\mathcal{E}|\mathcal{D}_T,\boldsymbol{x})}{P(\boldsymbol{y}|\mathcal{D},\mathcal{D}_T)}\right] \tag{14.45}$$

$$= \sum P(\mathcal{D}_T,\boldsymbol{x}) \sum P(\mathcal{E},\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x}) \log \frac{P(\mathcal{E},\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x})}{P(\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x})P(\mathcal{E}|\mathcal{D}_T,\boldsymbol{x})} \tag{14.46}$$

$$= \mathbb{I}(\mathcal{E};\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x}) \tag{14.47}$$

Hence

$$d^{KL}_{\mathcal{E},Q} = d^{KL}_{B,Q} + \mathbb{I}(\mathcal{E};\boldsymbol{y}|\mathcal{D}_T,\boldsymbol{x}) \tag{14.48}$$

as claimed.

14.3 Conformal prediction

In this section, we briefly discuss **conformal prediction** [VGS05; SV08; ZFV20; AB21; KSB21; Man22]. This is a simple but effective way to create prediction intervals or sets with guaranteed frequentist coverage probability from any predictive method $p(y|\boldsymbol{x})$. This can be seen as a form of **distribution free uncertainty quantification**, since it works without making assumptions (beyond exchangeability of the data) about the true data generating process or the form of the model.[1] Our presentation is based on the excellent tutorial of [AB21].[2]

In conformal prediction, we start with some heuristic notion of uncertainty — such as the softmax score for a classification problem, or the variance for a regression problem — and we use it to define a **conformal score** $s(\boldsymbol{x}, y) \in \mathbb{R}$, which measures how badly the output y "conforms" to $\boldsymbol{x}$. (Large

1. The exchangeability assumption rules out time series data, which is serially correlated. However, extensions to conformal prediction have been developed for the time series case, see e.g., [Zaf+22]. The exchangeability assumption also rules out distribution shift, although this has also been partially addressed.
2. See also the easy-to-use **MAPIE** Python library at https://mapie.readthedocs.io/en/latest/index.html, and the list of papers at https://github.com/valeman/awesome-conformal-prediction.

values of the score are less likely, so it is better to think of it as a non-conformity score.) Next we apply this score to a **calibration set** of n labeled examples, that was not used to train f, to get $\mathcal{S} = \{s_i = s(\boldsymbol{x}_i, y_i) : i = 1 : n\}$.[3] The user specifies a desired confidence threshold α, say 0.1, and we then compute the $(1-\alpha)$ quantile $\hat{q}$ of $\mathcal{S}$. (In fact, we should replace $1-\alpha$ with $\frac{\lceil (n+1)(1-\alpha) \rceil}{n}$, to account for the finite size of $\mathcal{S}$.) Finally, given a new test input, $\boldsymbol{x}_{n+1}$, we compute the prediction set to be

$$\mathcal{T}(\boldsymbol{x}_{n+1}) = \{y : s(\boldsymbol{x}_{n+1}, y) \leq \hat{q}\} \tag{14.49}$$

Intuitively, we include all the outputs y that are plausible given the input. See Figure 14.4 for an illustration.

Remarkably, one can show the following general result

$$1 - \alpha \leq P^*(y^{n+1} \in \mathcal{T}(\boldsymbol{x}_{n+1})) \leq 1 - \alpha + \frac{1}{n+1} \tag{14.50}$$

where the probability is wrt the true distribution $P^*(\boldsymbol{x}_{n+1}, y_{n+1})$. We say that the prediction set has a **coverage** level of $1-\alpha$. This holds for any value of $n \geq 1$ and $\alpha \in [0, 1]$. The only assumption is that the values $(\boldsymbol{x}_i, y_i)$ are exchangeable, and hence the calibration scores s_i are also exchangeable.

To see why this is true, let us sort the scores so $s_1 < \cdots s_n$, so $\hat{q} = s_i$, where $i = \frac{\lceil (n+1)(1-\alpha) \rceil}{n}$. (We assume the scores are distinct, for simplicity.) The score s_{n+1} is equally likely to fall in anywhere between the calibration points $s_1, \ldots, s_n$, since the points are exchangeable. Hence

$$P^*(s_{n+1} \leq s_k) = \frac{k}{n+1} \tag{14.51}$$

for any $k \in \{1, \ldots, n+1\}$. The event $\{y_{n+1} \in \mathcal{T}(\boldsymbol{x}_{n+1})\}$ is equivalent to $\{s_{n+1} \leq \hat{q}\}$. Hence

$$P^*(y_{n+1} \in \mathcal{T}(\boldsymbol{x}_{n+1})) = P^*(s_{n+1} \leq \hat{q}) = \frac{\lceil (n+1)(1-\alpha) \rceil}{n+1} \geq 1 - \alpha \tag{14.52}$$

For the proof of the upper bound, see [Lei+18].

Although this result may seem like a "free lunch", it is worth noting that we can always achieve a desired coverage level by defining the prediction set to be all possible labels. In this case, the prediction set will be independent of the input, but it will cover the true label $1-\alpha$ of the time. To rule out some degenerate cases, we seek prediction sets that are as small as possible (although we allow for the set to be larger for harder examples), while meeting the coverage requirement. Achieving this goal requires that we define suitable conformal scores. Below we give some examples of how to compute conformal scores $s(\boldsymbol{x}, y)$ for different kinds of problem.[4]

14.3.1 Conformalizing classification

The simplest way to apply conformal prediction to multiclass classification is to derive the conformal score from the softmax score assigned to the label using $s(\boldsymbol{x}, y) = 1 - f(\boldsymbol{x})_y$, so large values are

3. Using a calibration sety is called **split conformal prediction**. If we don't have enough data to adopt this splitting approach, we can use **full conformal prediction** [VGS05], which requires fitting the model n times using a leave-one-out type procedure.

4. It is also possible to learn conformal scores in an end-to-end way, jointly with the predictive model, as discussed in [Stu+22].

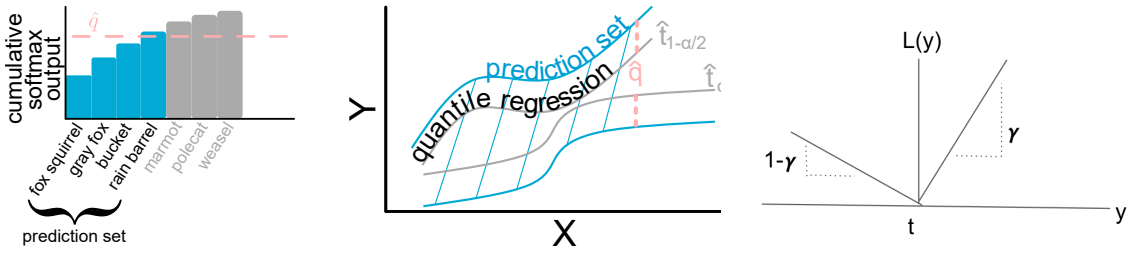


Figure 14.5: (a) Illusration of adaptive prediction set. From Figure 5 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (b) Illustrate of conformalized quantile regression. From Figure 6 of [AB21]. Used with kind permission of Anastasios Angelopoulos. (c) Illustration of pinball loss function.

considered less likely than small values. We compute the threshold $\hat{q}$ as described above, and then we define the prediction set to be $\mathcal{T}(\boldsymbol{x}) = \{y : f(\boldsymbol{x})_y \geq 1 - \hat{q}\}$, which matches Equation (14.49). That is, we take the set of all class labels above the specified threshold, as illustrated in Figure 14.4.

Although the above approach produces prediction sets with the smallest average size (as proved in [SLW19]), the size of the set tends to be too large for easy examples and too small for hard examples. We now present an improved method, known as **adaptive prediction sets**, due to [RSC20], which solves this problem. The idea is simple: we sort all the softmax scores, $f(\boldsymbol{x})_c$ for $c = 1 : C$, to get permutation $\pi_{1:C}$, and then we define $s(\boldsymbol{x}, y)$ to be the cumulative sum of the scores up until we reach label y: $s(\boldsymbol{x}, y) = \sum_{c=1}^{k} f(\boldsymbol{x})_{\pi_j}$, where $k = \pi_y$. We now compute $\hat{q}$ as before, and define the prediction set $\mathcal{T}(\boldsymbol{x})$ to be the set of all labels, sorted in order of decreasing probability, until we cover $\hat{q}$ of the probability mass. See Figure 14.5a for an illustration. This uses all the softmax scores output by the model, rather than just the top score, which accounts for its improved perforance.

14.3.2 Conformalizing regression

In this section, we consider conformalized regression problems. Since now $y \in \mathbb{R}$, computing the prediction set in Equation (14.49) is expensive, so instead we will compute a prediction interval, specified by a lower and upper bound.

14.3.2.1 Conformalizing quantile regression

In this section, we use **quantile regression** to compute the lower and upper bounds. We first fit a function of the form $t_\gamma(\boldsymbol{x})$, which predicts the γ quantile of the pdf $P(Y|\boldsymbol{x})$. For example, if we set $\gamma = 0.5$, we get the median. If we use $\gamma = 0.05$ and $\gamma = 0.95$, we can get an approximate 90% prediction interval using $[t_{0.05}(\boldsymbol{x}), t_{0.95}(\boldsymbol{x})]$, as illustrated by the gray lines in Figure 14.5b. To fit the quantile regression model, we just replace squared loss with the **quantile loss**, also called the **pinball loss**, which is defined as

$$\ell_\gamma(y, \hat{t}) = (y - \hat{t})\gamma \mathbb{I}\left(y > \hat{t}\right) + (\hat{t} - y)(1 - \gamma)\mathbb{I}\left(y < \hat{t}\right) \tag{14.53}$$

where y is the true output and $\hat{t}$ is the predicted value at quantile γ. See Figure 14.5c for an illustration.

The regression quantiles are only approximately a 90% interval because the model may be mismatched to the true distribution. However we can use conformal prediction to fix this. In particular, let us define the conformal score to be

$$s(\boldsymbol{x}, y) = \max\left(\hat{t}_{\alpha/2}(\boldsymbol{x}) - y, y - \hat{t}_{\alpha/2}(\boldsymbol{x})\right) \tag{14.54}$$

In other words, $s(\boldsymbol{x}, y)$ is a positive measure of how far the value y is outside the prediction interval, or is a negative measure if y is inside the prediction interval. We compute $\hat{q}$ as before, and define the conformal prediction interval to be

$$\mathcal{T}(\boldsymbol{x}) = [\hat{t}_{\alpha/2}(\boldsymbol{x}) - \hat{q}, \hat{t}_{\alpha/2}(\boldsymbol{x}) + \hat{q}] \tag{14.55}$$

This makes the quantile regression interval wider if $\hat{q}$ is positive (if the base method was overconfident), and narrower if $\hat{q}$ is negative (if the base method was underconfident). See Figure 14.5b for an illustration. This approach is called **conformalized quantile regression** or **CQR** [RPC19].

14.3.2.2 Conformalizing predicted variances

There are many ways to define uncertainty scores $u(\boldsymbol{x})$, such as the predicted standard deviation, from which we can derive a prediction interval using

$$\mathcal{T}(\boldsymbol{x}) = [f(\boldsymbol{x}) - u(\boldsymbol{x})\hat{q}, f(\boldsymbol{x}) + u(\boldsymbol{x})\hat{q}] \tag{14.56}$$

Here $\hat{q}$ is derived from the quantiles of the following conformal scores

$$s(\boldsymbol{x}, y) = \frac{|y - f(\boldsymbol{x})|}{u(\boldsymbol{x})} \tag{14.57}$$

The interval produced by this method tends to be wider than the one computed by CQR, since it extends an equal amount above and below the predicted value $f(\boldsymbol{x})$. In addition, the uncertainty measure $u(\boldsymbol{x})$ may not scale properly with α. Nevertheless, this is a simple post-hoc method that can be applied to many regression methods without needing to retrain them.

15 Generalized linear models

15.1 Introduction

A **generalized linear model** or **GLM** [MN89] is a conditional version of an exponential family distribution (Section 2.3). More precisely, the model has the following form:

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2) = \exp\left[\frac{y_n\eta_n - A(\eta_n)}{\sigma^2} + \log h(y_n, \sigma^2)\right] \tag{15.1}$$

where $\eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$ is the natural parameter for the distribution, $A(\eta_n)$ is the log normalizer, $\mathcal{T}(y) = y$ is the sufficient statistic, and σ^2 is the dispersion term. Based on the results in Section 2.3.3, we can show that the mean and variance of the response variable are as follows:

$$\mu_n \triangleq \mathbb{E}\left[y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2\right] = A'(\eta_n) \triangleq \ell^{-1}(\eta_n) \tag{15.2}$$

$$\mathbb{V}\left[y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2\right] = A''(\eta_n)\,\sigma^2 \tag{15.3}$$

We will denote the mapping from the linear inputs to the mean of the output using $\mu_n = \ell^{-1}(\eta_n)$, where the function ℓ is known as the **link function**, and ℓ^{-1} is known as the **mean function**. This relationship is usually written as follows:

$$\ell(\mu_n) = \eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n \tag{15.4}$$

15.1.1 Examples

In this section, we give some examples of widely used GLMs.

15.1.1.1 Linear regression

Recall that linear regression has the form

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}(y_n - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)^2) \tag{15.5}$$

Hence

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2) = -\frac{1}{2\sigma^2}(y_n - \eta_n)^2 - \frac{1}{2}\log(2\pi\sigma^2) \tag{15.6}$$

where $\eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$. We can write this in GLM form as follows:

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2) = \frac{y_n\eta_n - \frac{\eta_n^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y_n^2}{\sigma^2} + \log(2\pi\sigma^2)\right) \tag{15.7}$$

We see that $A(\eta_n) = \eta_n^2/2$ and hence

$$\mathbb{E}[y_n] = \eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n \tag{15.8}$$

$$\mathbb{V}[y_n] = \sigma^2 \tag{15.9}$$

See Section 15.2 for details on linear regression.

15.1.1.2 Binomial regression

If the response variable is the number of successes in N_n trials, $y_n \in \{0, \ldots, N_n\}$, we can use **binomial regression**, which is defined by

$$p(y_n|\boldsymbol{x}_n, N_n, \boldsymbol{w}) = \text{Bin}(y_n|\sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n), N_n) \tag{15.10}$$

We see that binary logistic regression is the special case when $N_n = 1$.

The log pdf is given by

$$\log p(y_n|\boldsymbol{x}_n, N_n, \boldsymbol{w}) = y_n \log \mu_n + (N_n - y_n)\log(1-\mu_n) + \log\binom{N_n}{y_n} \tag{15.11}$$

$$= y_n \log(\frac{\mu_n}{1-\mu_n}) + N_n \log(1-\mu_n) + \log\binom{N_n}{y_n} \tag{15.12}$$

where $\mu_n = \sigma(\eta_n)$. To rewrite this in GLM form, let us define

$$\eta_n \triangleq \log\left[\frac{\mu_n}{(1-\mu_n)}\right] = \log\left[\frac{1}{1+e^{-\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n}}\frac{1+e^{-\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n}}{e^{-\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n}}\right] = \log\frac{1}{e^{-\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n}} = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n \tag{15.13}$$

Hence we can write binomial regression in GLM form as follows

$$\log p(y_n|\boldsymbol{x}_n, N_n, \boldsymbol{w}) = y_n\eta_n - A(\eta_n) + h(y_n) \tag{15.14}$$

where $h(y_n) = \log\binom{N_n}{y_n}$ and

$$A(\eta_n) = -N_n \log(1-\mu_n) = N_n \log(1+e^{\eta_n}) \tag{15.15}$$

Hence

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = \frac{N_n e^{\eta_n}}{1+e^{\eta_n}} = \frac{N_n}{1+e^{-\eta_n}} = N_n\mu_n \tag{15.16}$$

and

$$\mathbb{V}[y_n] = \frac{d^2A}{d\eta_n^2} = N_n\mu_n(1-\mu_n) \tag{15.17}$$

See Supplementary Section 15.3 for an example of binomial regression.

15.1.1.3 Poisson regression

If the response variable is an integer count, $y_n \in \{0, 1, \ldots\}$, we can use **Poisson regression**, which is defined by

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = \text{Poi}(y_n|\exp(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)) \tag{15.18}$$

where

$$\text{Poi}(y|\mu) = e^{-\mu}\frac{\mu^y}{y!} \tag{15.19}$$

is the Poisson distribution. Poisson regression is widely used in bio-statistical applications, where y_n might represent the number of diseases of a given person or place, or the number of reads at a genomic location in a high-throughput sequencing context (see e.g., [Kua+09]).

The log pdf is given by

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = y_n \log \mu_n - \mu_n - \log(y_n!) \tag{15.20}$$

where $\mu_n = \exp(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)$. Hence in GLM form we have

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = y_n\eta_n - A(\eta_n) + h(y_n) \tag{15.21}$$

where $\eta_n = \log(\mu_n) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$, $A(\eta_n) = \mu_n = e^{\eta_n}$, and $h(y_n) = -\log(y_n!)$. Hence

$$\mathbb{E}[y_n] = \frac{dA}{d\eta_n} = e^{\eta_n} = \mu_n \tag{15.22}$$

and

$$\mathbb{V}[y_n] = \frac{d^2A}{d\eta_n^2} = e^{\eta_n} = \mu_n \tag{15.23}$$

15.1.1.4 Zero-inflated Poisson regression

In many forms of count data, the number of observed 0s is larger than what a model might expect, even after taking into account the predictors. Intuitively, this is because there may be many ways to produce no outcome. For example, consider predicting sales data for a product. If the sales are 0, does it mean the product is unpopular (so the demand is very low), or was it simply sold out (implying the demand is high, but exceed supply)? Similar problems arise in genomics, epidemiology, etc.

To handle such situations, it is common to use a **zero-inflated Poisson** or **ZIP** model. The likelihood for this model is a mixture of two distributions: a spike at 0, and a standard Poisson. Formally, we define

$$\text{ZIP}(y|\rho, \lambda) = \begin{cases} \rho + (1-\rho)\exp(-\lambda) & \text{if } y = 0 \\ (1-\rho)\frac{\lambda^y \exp(-\lambda)}{y!} & \text{if } y > 0 \end{cases} \tag{15.24}$$

Here ρ is the prior probability of picking the spike, and λ is the rate of the Poisson. We see that there are two "mechanisms" for generating a 0: either (with probability ρ) we chose the spike, or (with probability $1-\rho$) we simply generate a zero count just because the rate of the Poisson is so low. (This latter event has probability $\lambda^0 e^{-\lambda}/0! = e^{-\lambda}$.)

15.1.2 GLMs with non-canonical link functions

We have see how the mean parameters of the output distribution are given by $\mu = \ell^{-1}(\eta)$, where the function ℓ is the link function. There are several choices for this function, as we now discuss.

The **canonical link function** ℓ satisfies the property that $\theta = \ell(\mu)$, where θ are the canonical (natural) parameters. Hence

$$\theta = \ell(\mu) = \ell(\ell^{-1}(\eta)) = \eta \tag{15.25}$$

This is what we have assumed so far. For example, for the Bernoulli distribution, the canonical parameter is the log-odds $\eta = \log(\mu/(1-\mu))$, which is given by the logit transform

$$\eta = \ell(\mu) = \text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right) \tag{15.26}$$

The inverse of this is the sigmoid or logistic funciton

$$\mu = \ell^{-1}(\eta) = \sigma(\eta) = 1/(1+e^{-\eta}) \tag{15.27}$$

However, we are free to use other kinds of link function. For example, in Section 15.4 we use

$$\eta = \ell(\mu) = \Phi^{-1}(\mu) \tag{15.28}$$

$$\mu = \ell^{-1}(\eta) = \Phi(\eta) \tag{15.29}$$

This is known as the **probit link function**.

Another link function that is sometimes used for binary responses is the **complementary log-log** function

$$\eta = \ell(\mu) = \log(-\log(1-\mu)) \tag{15.30}$$

This is used in applications where we either observe 0 events (denoted by $y = 0$) or one or more (denoted by $y = 1$), where events are assumed to be governed by a Poisson distribution with rate λ. Let E be the number of events. The Poisson assumption means $p(E = 0) = \exp(-\lambda)$ and hence

$$p(y = 0) = (1-\mu) = p(E = 0) = \exp(-\lambda) \tag{15.31}$$

Thus $\lambda = -\log(1-\mu)$. When λ is a function of covariates, we need to ensure it is positive, so we use $\lambda = e^{\eta}$, and hence

$$\eta = \log(\lambda) = \log(-\log(1-\mu)) \tag{15.32}$$

15.1.3 Maximum likelihood estimation

GLMs can be fit using similar methods to those that we used to fit logistic regression. In particular, the negative log-likelihood has the following form (ignoring constant terms):

$$\text{NLL}(\boldsymbol{w}) = -\log p(\mathcal{D}|\boldsymbol{w}) = -\frac{1}{\sigma^2}\sum_{n=1}^{N} \ell_n \tag{15.33}$$

where

$$\ell_n \triangleq \eta_n y_n - A(\eta_n) \tag{15.34}$$

where $\eta_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n$. For notational simplicity, we will assume $\sigma^2 = 1$.

We can compute the gradient for a single term as follows:

$$\boldsymbol{g}_n \triangleq \frac{\partial \ell_n}{\partial \boldsymbol{w}} = \frac{\partial \ell_n}{\partial \eta_n}\frac{\partial \eta_n}{\partial \boldsymbol{w}} = (y_n - A'(\eta_n))\boldsymbol{x}_n = (y_n - \mu_n)\boldsymbol{x}_n \tag{15.35}$$

where $\mu_n = f(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)$, and f is the inverse link function that maps from canonical parameters to mean parameters. (For example, in the case of logistic regression, we have $\mu_n = \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x})$.)

The Hessian is given by

$$\mathbf{H} = \frac{\partial^2}{\partial \boldsymbol{w}\partial \boldsymbol{w}^\mathsf{T}}\text{NLL}(\boldsymbol{w}) = -\sum_{n=1}^{N}\frac{\partial \boldsymbol{g}_n}{\partial \boldsymbol{w}^\mathsf{T}} \tag{15.36}$$

where

$$\frac{\partial \boldsymbol{g}_n}{\partial \boldsymbol{w}^\mathsf{T}} = \frac{\partial \boldsymbol{g}_n}{\partial \mu_n}\frac{\partial \mu_n}{\partial \boldsymbol{w}^\mathsf{T}} = -\boldsymbol{x}_n f'(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)\boldsymbol{x}_n^\mathsf{T} \tag{15.37}$$

Hence

$$\mathbf{H} = \sum_{n=1}^{N} f'(\eta_n)\boldsymbol{x}_n\boldsymbol{x}_n^\mathsf{T} \tag{15.38}$$

For example, in the case of logistic regression, $f(\eta_n) = \sigma(\eta_n) = \mu_n$, and $f'(\eta_n) = \mu_n(1-\mu_n)$. In general, we see that the Hessian is positive definite, since $f'(\eta_n) > 0$; hence the negative log likelihood is convex, so the MLE for a GLM is unique (assuming $f(\eta_n) > 0$ for all n).

For small datasets, we can use the **iteratively reweighted least squares** or **IRLS** algorithm, which is a form Newton's method, to compute the MLE (see e.g., [Mur22, Sec 10.2.6]). For large datsets, we can use SGD. (In practice it is often useful to combine SGD with methods that automatically tune the step size, such as [Loi+21].)

15.1.4 Bayesian inference

To perform Bayesian inference of the parameters, we first need to specify a prior. Choosing a suitable prior depends on the form of link function. For example, a "flat" or "uninformative" prior on the offset term $\alpha \in \mathbb{R}$ will not translate to an uninformative prior on the probability scale if we pass α through a sigmoid, as we discuss in Section 15.3.3.

Once we have chosen the prior, we can compute the posterior using a variety of approximate inference methods. For small datasets, HMC (Section 12.5) is the easiest to use, since you just need to write down the log likelihood and log prior, and use autograd to compute derivatives and pass them to the HMC engine. We give some examples in the following sections. For large datasets, assumed density filtering (Section 8.10.3) stochastic variational inference (Section 10.3.1), or more specialized algorithms (e.g., [HAB17]) are the best choice.

15.2 Linear regression

Linear regression is the simplest case of a GLM. We gave a detailed introduction to this model in the prequel to this book, [Mur22]. In this section, we discuss this model from a Bayesian perspective.

15.2.1 Conjugate priors

In this section, we derive the posterior for the parameters of the model when we use a conjugate prior. We first consider the case where just $\boldsymbol{w}$ is unknown (so the observation noise variance parameter σ^2 is fixed), and then we consder the general case, where both σ^2 and $\boldsymbol{w}$ are unknown.

15.2.1.1 Noise variance is known

The conjugate prior for linear regression has the following form:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\, \breve{\boldsymbol{w}}, \breve{\boldsymbol{\Sigma}}) \tag{15.39}$$

We often use $\breve{\boldsymbol{w}}= \mathbf{0}$ as the prior mean and $\breve{\boldsymbol{\Sigma}}= \tau^2 \mathbf{I}_D$ as the prior covariance. (We assume the bias term is included in the weight vector, but often use a much weaker prior for it, since we typically do not want to regularize the overall mean level of the output.)

To derive the posterior, let us first rewrite the likelihood in terms of an MVN as follows:

$$\ell(\boldsymbol{w}) = p(\mathcal{D}|\boldsymbol{w}, \sigma^2) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w}^\mathsf{T}\boldsymbol{x}, \sigma^2) = \mathcal{N}(\boldsymbol{y}|\mathbf{X}\boldsymbol{w}, \sigma^2\mathbf{I}_N) \tag{15.40}$$

where $\mathbf{I}_N$ is the $N \times N$ identity matrix. We can then use Bayes rule for Gaussians (Equation (2.82)) to derive the posterior, which is as follows:

$$p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{y}, \sigma^2) \propto \mathcal{N}(\boldsymbol{w}|\, \breve{\boldsymbol{w}}, \breve{\boldsymbol{\Sigma}})\mathcal{N}(\boldsymbol{y}|\mathbf{X}\boldsymbol{w}, \sigma^2\mathbf{I}_N) = \mathcal{N}(\boldsymbol{w}|\, \widehat{\boldsymbol{w}}, \widehat{\Sigma}) \tag{15.41}$$

$$\widehat{\boldsymbol{w}} \triangleq \widehat{\boldsymbol{\Sigma}}\, (\breve{\boldsymbol{\Sigma}}^{-1}\breve{\boldsymbol{w}} + \frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\boldsymbol{y}) \tag{15.42}$$

$$\widehat{\boldsymbol{\Sigma}} \triangleq (\breve{\boldsymbol{\Sigma}}^{-1} + \frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{X})^{-1} \tag{15.43}$$

where $\widehat{\boldsymbol{w}}$ is the posterior mean, and $\widehat{\boldsymbol{\Sigma}}$ is the posterior covariance.

Now suppose $\breve{\boldsymbol{w}}= \mathbf{0}$ and $\breve{\boldsymbol{\Sigma}}= \tau^2\mathbf{I}$. In this case, the posterior mean becomes

$$\widehat{\boldsymbol{w}}= \frac{1}{\sigma^2}\, \widehat{\boldsymbol{\Sigma}}\, \mathbf{X}^\mathsf{T}\boldsymbol{y} = (\frac{\sigma^2}{\tau^2}\mathbf{I} + \mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} \tag{15.44}$$

If we define $\lambda = \frac{\sigma^2}{\tau^2}$, we see this is equivalent to **ridge regression**, which optimizes

$$\mathcal{L}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||^2 \tag{15.45}$$

where RSS is the residual sum of squares:

$$\mathrm{RSS}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n)^2 = \frac{1}{2}||\mathbf{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 = \frac{1}{2}(\mathbf{X}\boldsymbol{w} - \boldsymbol{y})^\mathsf{T}(\mathbf{X}\boldsymbol{w} - \boldsymbol{y}) \tag{15.46}$$

15.2.1.2 Noise variance is unknown

In this section, we assume $\boldsymbol{w}$ and σ^2 are both unknown. The likelihood is given by

$$\ell(\boldsymbol{w}, \sigma^2) = p(\mathcal{D}|\boldsymbol{w}, \sigma^2) \propto (\sigma^2)^{-N_{\mathcal{D}}/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N_{\mathcal{D}}}(y_n - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_n)^2\right) \tag{15.47}$$

Since the regression weights now depend on σ^2 in the likelihood, the conjugate prior for $\boldsymbol{w}$ has the form

$$p(\boldsymbol{w}|\sigma^2) = \mathcal{N}(\boldsymbol{w}|\, \breve{\boldsymbol{w}}, \sigma^2\, \breve{\boldsymbol{\Sigma}}) \tag{15.48}$$

For the noise variance σ^2, the conjugate prior is based on the inverse Gamma distrbution, which has the form

$$\mathrm{IG}(\sigma^2|\, \breve{a}, \breve{b}) = \frac{\breve{b}^{\breve{a}}}{\Gamma(\breve{a})}(\sigma^2)^{-(\breve{a}+1)} \exp(-\frac{\breve{b}}{\sigma^2}) \tag{15.49}$$

(See Section 2.2.3.4 for more details.) Putting these two together, we find that the joint conjugate prior is the **normal inverse Gamma** distribution:

$$\mathrm{NIG}(\boldsymbol{w}, \sigma^2|\, \breve{\boldsymbol{w}}, \breve{\boldsymbol{\Sigma}}, \breve{a}, \breve{b}) \triangleq \mathcal{N}(\boldsymbol{w}|\, \breve{\boldsymbol{w}}, \sigma^2\, \breve{\boldsymbol{\Sigma}})\mathrm{IG}(\sigma^2|\, \breve{a}, \breve{b}) \tag{15.50}$$

$$= \frac{\breve{b}^{\breve{a}}}{(2\pi)^{D/2}|\, \breve{\boldsymbol{\Sigma}}\, |^{\frac{1}{2}}\Gamma(\breve{a})}\, (\sigma^2)^{-(\breve{a}+(D/2)+1)}$$

$$\times \exp\left[-\frac{(\boldsymbol{w}-\, \breve{\boldsymbol{w}})^{\mathsf{T}}\, \breve{\boldsymbol{\Sigma}}^{-1}\, (\boldsymbol{w}-\, \breve{\boldsymbol{w}}) + 2\, \breve{b}}{2\sigma^2}\right] \tag{15.51}$$

This results in the following posterior:

$$p(\boldsymbol{w}, \sigma^2|\mathcal{D}) = \mathrm{NIG}(\boldsymbol{w}, \sigma^2|\, \widehat{\boldsymbol{w}}, \widehat{\boldsymbol{\Sigma}}, \widehat{a}, \widehat{b}) \tag{15.52}$$

$$\widehat{\boldsymbol{w}} = \widehat{\boldsymbol{\Sigma}}\, (\breve{\boldsymbol{\Sigma}}^{-1}\breve{\boldsymbol{w}} + \mathbf{X}^{\mathsf{T}}\boldsymbol{y}) \tag{15.53}$$

$$\widehat{\boldsymbol{\Sigma}} = (\breve{\boldsymbol{\Sigma}}^{-1} + \mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1} \tag{15.54}$$

$$\widehat{a} = \breve{a} + N/2 \tag{15.55}$$

$$\widehat{b} = \breve{b} + \frac{1}{2}\left(\breve{\boldsymbol{w}}^{\mathsf{T}}\breve{\boldsymbol{\Sigma}}^{-1}\breve{\boldsymbol{w}} + \boldsymbol{y}^{\mathsf{T}}\boldsymbol{y} - \widehat{\boldsymbol{w}}^{\mathsf{T}}\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{w}}\right) \tag{15.56}$$

The expressions for $\widehat{\boldsymbol{w}}$ and $\widehat{\boldsymbol{\Sigma}}$ are similar to the case where σ^2 is known. The expression for $\widehat{a}$ is also intuitive, since it just updates the counts. The expression for $\widehat{b}$ can be interpreted as follows: it is the prior sum of squares, $\breve{b}$, plus the empirical sum of squares, $\boldsymbol{y}^{\mathsf{T}}\boldsymbol{y}$, plus a term due to the error in the prior on $\boldsymbol{w}$.

The posterior marginals are as follows. For the variance, we have

$$p(\sigma^2|\mathcal{D}) = \int p(\boldsymbol{w}|\sigma^2, \mathcal{D})p(\sigma^2|\mathcal{D})d\boldsymbol{w} = \mathrm{IG}(\sigma^2|\, \widehat{a}, \widehat{b}) \tag{15.57}$$

For the regression weights, it can be shown that

$$p(\boldsymbol{w}|\mathcal{D}) = \int p(\boldsymbol{w}|\sigma^2, \mathcal{D})p(\sigma^2|\mathcal{D})d\sigma^2 = \mathcal{T}(\boldsymbol{w}|\, \widehat{\boldsymbol{w}}, \frac{\widehat{b}}{\widehat{a}}\, \widehat{\boldsymbol{\Sigma}}, 2\, \widehat{a}) \tag{15.58}$$

15.2.1.3 Posterior predictive distribution

In machine learning we usually care more about uncertainty (and accuracy) of our predictions, not our parameter estimates. Fortunately, one can derive the posterior predictive distribution in closed form. In particular, one can show that, given N' new test inputs $\tilde{\mathbf{X}}$, we have

$$p(\tilde{\boldsymbol{y}}|\tilde{\mathbf{X}}, \mathcal{D}) = \int\int p(\tilde{\boldsymbol{y}}|\tilde{\mathbf{X}}, \boldsymbol{w}, \sigma^2) p(\boldsymbol{w}, \sigma^2|\mathcal{D}) d\boldsymbol{w} d\sigma^2 \tag{15.59}$$

$$= \int\int \mathcal{N}(\tilde{\boldsymbol{y}}|\tilde{\mathbf{X}}\boldsymbol{w}, \sigma^2 \mathbf{I}_{N'}) \mathrm{NIG}(\boldsymbol{w}, \sigma^2|\, \widehat{\boldsymbol{w}}, \widehat{\boldsymbol{\Sigma}}, \widehat{a}, \widehat{b}) d\boldsymbol{w} d\sigma^2 \tag{15.60}$$

$$= \mathcal{T}(\tilde{\boldsymbol{y}}|\tilde{\mathbf{X}}\, \widehat{\boldsymbol{w}}, \frac{\widehat{b}}{\widehat{a}}(\mathbf{I}_{N'} + \tilde{\mathbf{X}}\, \widehat{\boldsymbol{\Sigma}}\, \tilde{\mathbf{X}}^{\mathsf{T}}), 2\,\widehat{a}) \tag{15.61}$$

The posterior predictive mean is equivalent to "normal" linear regression, but where we plug in $\hat{\boldsymbol{w}} = \mathbb{E}[\boldsymbol{w}|\mathcal{D}]$ instead of the MLE. The posterior predictive variance has two components: $\widehat{b}/\widehat{a}\mathbf{I}_{N'}$ due to the measurement noise, and $\widehat{b}/\widehat{a}\tilde{\mathbf{X}}\, \widehat{\boldsymbol{\Sigma}}\, \tilde{\mathbf{X}}^{\mathsf{T}}$ due to the uncertainty in $\boldsymbol{w}$. This latter term varies depending on how close the test inputs are to the training data. The results are similar to using a Gaussian prior (with fixed $\hat{\sigma}^2$), except the predictive distribution is even wider, since we are taking into account uncertainty about σ^2.

15.2.2 Uninformative priors

A common criticism of Bayesian inference is the need to use a prior. This is sometimes thought to "pollute" the inferences one makes from the data. We can minimize the effect of the prior by using an uninformative prior, as we discussed in Section 3.6. Below we discuss various uninformative priors for linear regression.

15.2.2.1 Jeffreys prior

From Section 3.6.3.1, we know that the Jeffreys prior for the location parameter has the form $p(\boldsymbol{w}) \propto 1$, and from Section 3.6.3.2, we know that the Jeffreys prior for the scale factor has the form $p(\sigma) \propto \sigma^{-1}$. We can emulate these priors using an improper NIG prior with $\breve{\boldsymbol{w}} = \mathbf{0}$, $\breve{\boldsymbol{\Sigma}} = \infty\mathbf{I}$, $\breve{a} = -D/2$ and $\breve{b} = 0$. The corresponding posterior is given by

$$p(\boldsymbol{w}, \sigma^2|\mathcal{D}) = \mathrm{NIG}(\boldsymbol{w}, \sigma^2|\, \widehat{\boldsymbol{w}}, \widehat{\boldsymbol{\Sigma}}, \widehat{a}, \widehat{b}) \tag{15.62}$$

$$\widehat{\boldsymbol{w}} = \hat{\boldsymbol{w}}_{\mathrm{mle}} = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\boldsymbol{y} \tag{15.63}$$

$$\widehat{\boldsymbol{\Sigma}} = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1} \triangleq \mathbf{C} \tag{15.64}$$

$$\widehat{a} = \frac{\nu}{2} \tag{15.65}$$

$$\widehat{b} = \frac{s^2\nu}{2} \tag{15.66}$$

$$s^2 \triangleq \frac{||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2}{\nu} \tag{15.67}$$

$$\nu = N - D \tag{15.68}$$

Hence the posterior distribution of the weights is given by

$$p(\boldsymbol{w}|\mathcal{D}) = \mathcal{T}(\boldsymbol{w}|\hat{\boldsymbol{w}}, s^2\mathbf{C}, \nu) \tag{15.69}$$

where $\hat{\boldsymbol{w}}$ is the MLE. The marginals for each weight therefore have the form

$$p(w_d|\mathcal{D}) = \mathcal{T}(w_d|\hat{w}_d, s^2 C_{dd}, \nu) \tag{15.70}$$

15.2.2.2 Connection to frequentist statistics

Interestingly, the posterior when using Jeffrey's prior is formally equivalent to the **frequentist sampling distribution** of the MLE, which has the form

$$p(\hat{w}_d|\mathcal{D}^*) = \mathcal{T}(\hat{w}_d|w_d, s^2 C_{dd}, \nu) \tag{15.71}$$

where $\mathcal{D}^* = (\mathbf{X}, \boldsymbol{y}^*)$ is hypothetical data generated from the true model given the fixed inputs $\mathbf{X}$. In books on frequentist statistics, this is more commonly written in the following equivalent way (see e.g., [Ric95, p542]):

$$\frac{\hat{w}_d - w_d}{s\sqrt{C_{dd}}} \sim t_{N-D} \tag{15.72}$$

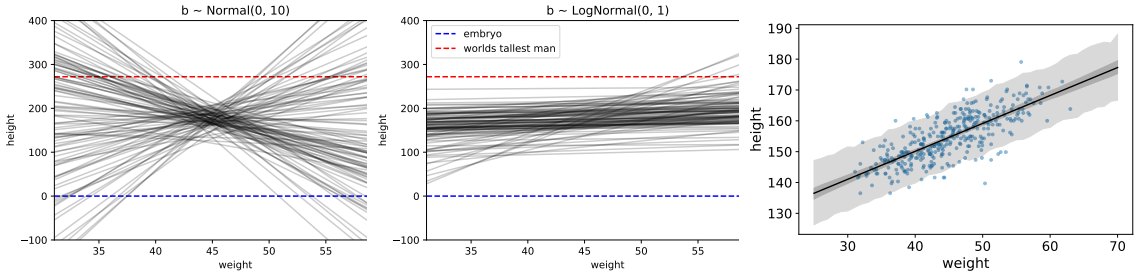
The sampling distribution is numerically the same as the posterior distribution in Equation (15.70) because $\mathcal{T}(w|\mu, \sigma^2, \nu) = \mathcal{T}(\mu|w, \sigma^2, \nu)$. However, it is semantically quite different, since the sampling distribution does not condition on the observed data, but instead is based on hypothetical data drawn from the model. See [BT73, p117] for more discussion of the equivalences between Bayesian and frequentist analysis of simple linear models when using uninformative priors.

15.2.2.3 Zellner's g-prior

It is often reasonable to assume an uninformative prior on σ^2, since that is just a scalar that does not have much influence on the results, but using an uninformative prior for $\boldsymbol{w}$ can be dangerous, since the strength of the prior controls how well regularized the model is, as we know from ridge regression.

A common compromise is to use an NIG prior with $\breve{a} = -D/2$, $\breve{b} = 0$ (to ensure $p(\sigma^2) \propto 1$) and $\breve{\boldsymbol{w}} = \mathbf{0}$ and $\breve{\boldsymbol{\Sigma}} = g(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$, where $g > 0$ plays a role analogous to $1/\lambda$ in ridge regression. This is called Zellner's **g-prior** [Zel86].[1] We see that the prior covariance is proportional to $(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$ rather than $\mathbf{I}$; this ensures that the posterior is invariant to scaling of the inputs, e.g., due to a change in the units of measurement [Min00a].

1. Note this prior is conditioned on the inputs $\mathbf{X}$, but not the outputs $\boldsymbol{y}$; this is totally valid in a conditional (discriminative) model, where all calculations are conditioned on $\mathbf{X}$, which is treated like a fixed constant input.



(a) (b) (c)

Figure 15.1: Linear regression for predicting height given weight, $y \sim \mathcal{N}(\alpha + \beta x, \sigma^2)$. (a) Prior predictive samples using a Gaussian prior for β. (b) Prior predictive samples using a Log-Gaussian prior for β. (c) Posterior predictive samples using the Log-Gaussian prior. The inner shaded band is the 95% credible interval for μ, representing epistemic uncertainty. The outer shaded band is the 95% credible interval for the observations y, which also adds data uncertainty due to σ. Adapted from Figures 4.5 and 4.10 of [McE20]. Generated by linreg_height_weight.ipynb.

With this prior, the posterior becomes

$$p(\boldsymbol{w}, \sigma^2 | g, \mathcal{D}) = \text{NIG}(\boldsymbol{w}, \sigma^2 | \boldsymbol{w}_N, \mathbf{V}_N, a_N, b_N) \tag{15.73}$$

$$\mathbf{V}_N = \frac{g}{g+1}(\mathbf{X}^T\mathbf{X})^{-1} \tag{15.74}$$

$$\boldsymbol{w}_N = \frac{g}{g+1}\hat{\boldsymbol{w}}_{mle} \tag{15.75}$$

$$a_N = N/2 \tag{15.76}$$

$$b_N = \frac{s^2}{2} + \frac{1}{2(g+1)}\hat{\boldsymbol{w}}_{mle}^T\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{w}}_{mle} \tag{15.77}$$

Various approaches have been proposed for setting g, including cross validation, empirical Bayes [Min00a; GF00], hierarchical Bayes [Lia+08], etc.

15.2.3 Informative priors

In many problems, it is possible to use domain knowledge to come up with plausible priors. As an example, we consider the problem of predicting the height of a person given their weight. We will use a dataset collected from Kalahari foragers by the anthropologist Nancy Howell (this example is from the book "Statistical Rethinking" [McE20, p93]).

Let x_i be the weight (in kg) and y_i be height (in cm) of the i'th person, and let $\overline{x}$ be the mean of the inputs. The observation model is given by

$$y_i \sim \mathcal{N}(\mu_i, \sigma) \tag{15.78}$$

$$\mu_i = \alpha + \beta(x_i - \overline{x}) \tag{15.79}$$

We see that the intercept α is the predicted output if $x_i = \overline{x}$, and the slope β is the predicted change in height per unit change in weight above or below the average weight.

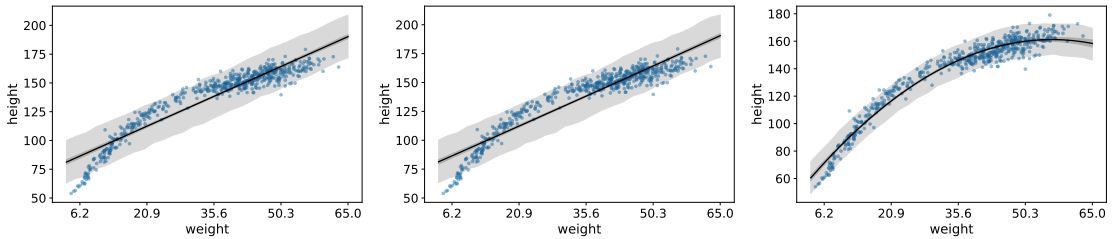


Figure 15.2: Linear regression for predicting height given weight for the full dataset (including children) using polynomial regression. (a) Posterior fit for linear model with log-Gaussian prior for β_1. (b) Posterior fit for quadratic model with log-Gaussian prior for β_2. (c) Posterior fit for quadratic model with Gaussian prior for β_2. Adapted from Figure 4.11 of [McE20]. Generated by linreg_height_weight.ipynb.

The question is: what priors should we use? To be truly Bayesian, we should set these before looking at the data. A sensible prior for α is the height of a "typical person", with some spread. We use $\alpha \sim \mathcal{N}(178, 20)$, since the author of the "Rethinking Statistics" book from which this example is taken is 178cm. By using a standard deviation of 20, the prior puts 95% probability on the broad range of 178 ± 40.

What about the prior for β? It is tempting to use a **vague prior**, or **weak prior**, such as $\beta \sim \mathcal{N}(0, 10)$, which is similar to a flat (uniform) prior, but more concentrated at 0 (a form of mild regularization). To see if this is reasonable, we can compute samples from the **prior predictive distribution**, i.e., we sample $(\alpha_s, \beta_s) \sim p(\alpha)p(\beta)$, and then plot $\alpha_s x + \beta_s$ for a range of x values, for different samples $s = 1 : S$. The results are shown in Figure 15.1a. We see that this is not a very sensible prior. For example, we see that it suggests that it is just as likely for the height to decrease with weight as increase with weight, which is not plausible. In addition, it predicts heights which are larger than the world's tallest person (272 cm) and smaller than the world's shortest person (an embryo, of size 0).

We can encode the monotonically increasing relationship between weight and height by restricting β to be positive. An easy way to do this is to use a log-normal or log-Gaussian prior. (If $\tilde{\beta} = \log(\beta)$ is Gaussian, then $e^{\tilde{\beta}}$ must be positive.) Specifically, we will assume $\beta \sim \mathcal{LN}(0, 1)$. Samples from this prior are shown in Figure 15.1b. This is much more reasonable.

Finally we must choose a prior over σ. In [McE20] they use $\sigma \sim \text{Unif}(0, 50)$. This ensures that σ is positive, and that the prior predictive distribution for the output is within 100cm of the average height. However, it is usually easier to specify the expected value for σ than an upper bound. To do this, we can use $\sigma \sim \text{Expon}(\lambda)$, where λ is the rate. We then set $\mathbb{E}[\sigma] = 1/\lambda$ to the value of the standard deviation that we expect. For example, we can use the empirical standard deviation of the data.

Since these priors are no longer conjugate, we cannot compute the posterior in closed form. However, we can use a variety of approximate inference methods. In this simple example, it suffices to use a quadratic (Laplace) approximation (see Section 7.4.3). The results are shown in Figure 15.1c, and look sensible.

So far, we have only considered a subset of the data, corresponding to adults over the age of 18. If

we include children, we find that the mapping from weight to height is nonlinear. This is illustrated in Figure 15.2a. We can fix this problem by using **polynomial regression**. For example, consider a quadratic expansion of the standardized features x_i:

$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 \tag{15.80}$$

If we use a log-Gaussian prior for β_2, we find that the model is too constrained, and it underfits. This is illustrated in Figure 15.2b. The reason is that we need to use an inverted quadratic with a negative coefficient, but since this is disallowed by the prior, the model ends up not using this degree of freedom (we find $\mathbb{E}[\beta_2|\mathcal{D}] \approx 0.08$). If we use a Gaussian prior on β_2, we avoid this problem, illustrated in Figure 15.2c.

This example shows that it can be useful to think about the functional form of the mapping from inputs to outputs in order to specify sensible priors.

15.2.4 Spike and slab prior

It is often useful to be able to select a subset of the input features when performing prediction, either to reduce overfitting, or to improve interpretability of the model. This can be achieved if we ensure that the weight vector $\boldsymbol{w}$ is **sparse** (i.e., has many zero elements), since if $w_d = 0$, then x_d plays no role in the inner product $\boldsymbol{w}^\mathsf{T}\boldsymbol{x}$.

The canonical way to achieve sparsity when using Bayesian inference is to use a **spike-and-slab** (**SS**) prior [MB88], which has the form of a 2 component mixture model, with one component being a "spike" at 0, and the other being a uniform "slab" between $-a$ and a:

$$p(\boldsymbol{w}) = \prod_{d=1}^{D} (1-\pi)\delta(w_d) + \pi \text{Unif}(w_d|-a, a) \tag{15.81}$$

where π is the prior probability that each coefficient is non-zero. The corresponding log prior on the coefficients is thus

$$\log p(\boldsymbol{w}) = ||\boldsymbol{w}||_0 \log(1-\pi) + (D - ||\boldsymbol{w}||_0) \log \pi = -\lambda ||\boldsymbol{w}||_0 + \text{const} \tag{15.82}$$

where $\lambda = \log \frac{\pi}{1-\pi}$ controls the sparsity of the model, and $||\boldsymbol{w}||_0 = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$ is the ℓ_0 **norm** of the weights. Thus MAP estimation with a spike and slab prior is equivalent ℓ_0 **regularization**; this penalizes the number of non-zero coefficients. Interestingly, posterior samples will also be sparse.

By contrast, consider using a a Laplace prior. The **lasso** estimator uses MAP estimation, which results in a sparse estimate. However, posterior samples are not sparse. Interestingly, [EY09] show theoretically (and [SPZ09] confirm experimentally) that using the posterior mean with a spike-and-slab prior also results in better prediction accuracy than using the posterior mode with a Laplace prior.

In practice, we often approximate the uniform slab with a broad Gaussian distribution,

$$p(\boldsymbol{w}) = \prod_{d} (1-\pi)\delta(w_d) + \pi \mathcal{N}(w_d|0, \sigma_w^2) \tag{15.83}$$

As $\sigma_w^2 \to \infty$, the second term approaches a uniform distribution over $[-\infty, +\infty]$. We can implement the mixture model by associating a binary random variable, $s_d \sim \text{Ber}(\pi)$, with each coefficient, to indicate if the coefficient is "on" or "off".