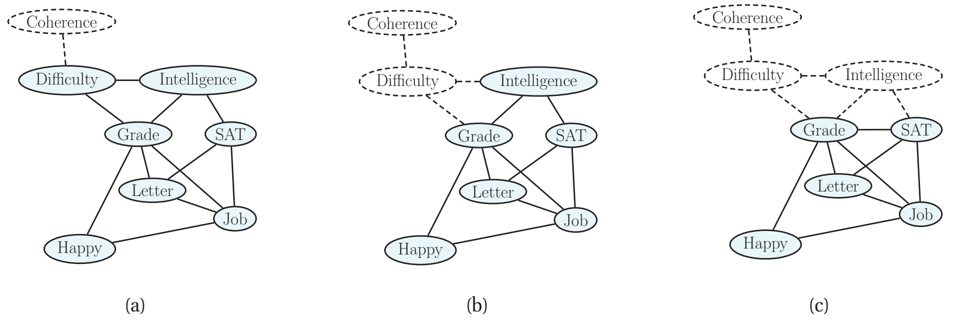


Figure 20.4 Example of the elimination process, in the order C, D, I, etc. When we eliminate I (figure c), we add a fill-in edge between G and S, since they are not connected. Based on Figure 9.10 of (Koller and Friedman 2009).

that share a factor with X_t (to reflect the new temporary factor τ'_t). The edges created by this process are called **fill-in edges**. For example, Figure 20.4 shows the fill-in edges introduced when we eliminate in the order $C, D, I, \ldots$. The first two steps do not introduce any fill-ins, but when we eliminate I, we connect G and S, since they co-occur in Equation 20.48.

Let $G(\prec)$ be the (undirected) graph induced by applying variable elimination to G using elimination ordering $\prec$. The temporary factors generated by VE correspond to maximal cliques in the graph $G(\prec)$. For example, with ordering (C, D, I, H, G, S, L), the maximal cliques are as follows:

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\} \quad (20.54)$$

It is clear that the time complexity of VE is

$$\prod_{c \in \mathcal{C}(G(\prec))} K^{|c|} \quad (20.55)$$

where $\mathcal{C}$ are the cliques that are created, $|c|$ 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 $O(K^{w(\prec)+1})$.

Obviously we would like to minimize the running time, and hence the induced width. Let us define the **treewidth** of a graph as the minimal induced width.

$$w \triangleq \min_{\prec} \max_{c \in G(\prec)} |c| - 1 \quad (20.56)$$

Then clearly the best possible running time for VE is $O(DK^{w+1})$. Unfortunately, one can show that for arbitrary graphs, finding an elimination ordering $\prec$ that minimizes $w(\prec)$ is NP-hard (Arnborg et al. 1987). In practice greedy search techniques are used to find reasonable orderings (Kjaerulff 1990), although people have tried other heuristic methods for discrete optimization, such as genetic algorithms (Larranaga et al. 1997). It is also possible to derive approximate algorithms with provable performance guarantees (Amir 2010).

In some cases, the optimal elimination ordering is clear. For example, for chains, we should work forwards or backwards in time. For trees, we should work from the leaves to the root. These orderings do not introduce any fill-in edges, so $w = 1$. Consequently, inference in chains and trees takes $O(VK^2)$ time. This is one reason why Markov chains and Markov trees are so widely used.

Unfortunately, for other graphs, the treewidth is large. For example, for an $m \times n$ 2d lattice, the treewidth is $O(\min\{m, n\})$ (Lipton and Tarjan 1979). So VE on a 100×100 Ising model would take $O(2^{100})$ time.

Of course, just because VE is slow doesn't mean that there isn't some smarter algorithm out there. We discuss this issue in Section 20.5.

20.3.3 A weakness of VE

The main disadvantage of the variable elimination algorithm (apart from its exponential dependence on treewidth) is that it is inefficient if we want to compute multiple queries conditioned on the same evidence. For example, consider computing all the marginals in a chain-structured graphical model such as an HMM. We can easily compute the final marginal $p(x_T|\mathbf{v})$ by eliminating all the nodes x_1 to x_{T-1} in order. This is equivalent to the forwards algorithm, and takes $O(K^2T)$ time. But now suppose we want to compute $p(x_{T-1}|\mathbf{v})$. We have to run VE again, at a cost 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 forwards-backwards. The difference is that FB caches the messages computed on the forwards pass, so it can reuse them later.

The same argument holds for BP on trees. For example, consider the 4-node tree in Figure 20.5. We can compute $p(x_1|\mathbf{v})$ by eliminating $\mathbf{x}_{2:4}$; this is equivalent to sending messages up to x_1 (the messages correspond to the τ factors created by VE). Similarly we can compute $p(x_2|\mathbf{v})$, $p(x_3|\mathbf{v})$ and then $p(x_4|\mathbf{v})$. 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 $O(DK^2)$ time. This is what the up-down (collect-distribute) algorithm on trees does.

The question is: how can we combine the efficiency of BP on trees with the generality of VE? The answer is given in Section 20.4.

20.4 The junction tree algorithm *

The **junction tree algorithm** or **JTA** generalizes BP from trees to arbitrary graphs. We sketch the basic idea below; for details, see e.g., (Koller and Friedman 2009).

20.4.1 Creating a junction tree

The basic idea behind the JTA is this. We first run the VE algorithm "symbolically", adding fill-in edges as we go, according to a given elimination ordering. The resulting graph will be a **chordal graph**, which means that every undirected cycle $X_1 - X_2 \cdots X_k - X_1$ of length $k \geq 4$ has a

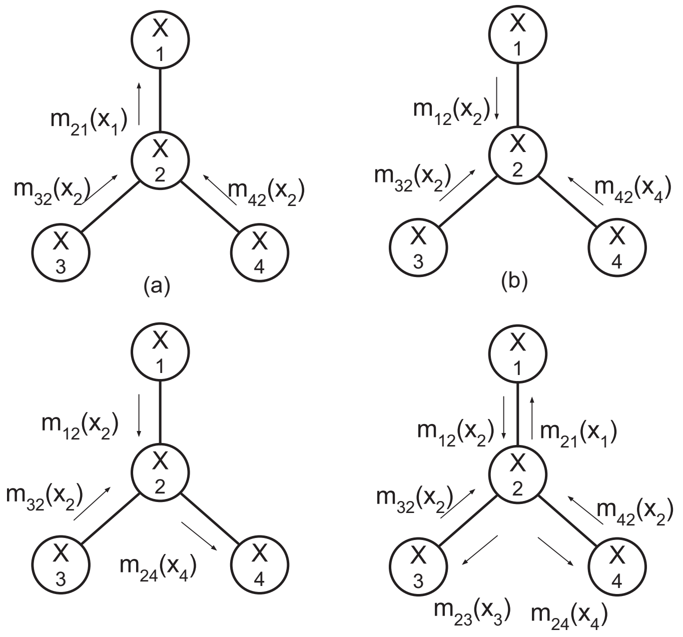


Figure 20.5 Sending multiple messages along a tree. (a) X_1 is root. (b) X_2 is root. (c) X_4 is root. (d) All of the messages needed to compute all singleton marginals. Based on Figure 4.3 of (Jordan 2007).

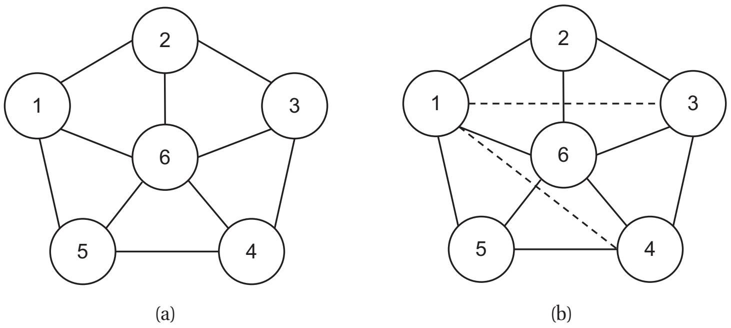


Figure 20.6 Left: this graph is not triangulated, despite appearances, since it contains a chordless 5-cycle 1-2-3-4-5-1. Right: one possible triangulation, by adding the 1-3 and 1-4 fill-in edges. Based on (Armstrong 2005, p46)

chord, i.e., an edge connects X_i, X_j for all non-adjacent nodes i,j in the cycle.[2]

Having created a chordal graph, we can extract its maximal cliques. In general, finding max cliques is computationally hard, but it turns out that it can be done efficiently from this special kind of graph. Figure 20.7(b) gives an example, where the max cliques are as follows:

$$\{C, D\}, \{G, I, D\}, \{G, S, I\}, \{G, J, S, L\}, \{H, G, J\} \tag{20.57}$$

Note that if the original graphical model was already chordal, the elimination process would not add any extra fill-in edges (assuming the optimal elimination ordering was used). We call such models **decomposable**, since they break into little pieces defined by the cliques.

It turns out that the cliques of a chordal graph can be arranged into a special kind of tree known as a **junction tree**. This enjoys the **running intersection property** (RIP), which means that any subset of nodes containing a given variable forms a connected component. Figure 20.7(c) gives an example of such a tree. We see that the node I occurs in two adjacent tree nodes, so they can share information about this variable. A similar situation holds for all the other variables.

One can show that if a tree that satisfies the running intersection property, then applying BP to this tree (as we explain below) will return the exact values of $p(\mathbf{x}_c|\mathbf{v})$ for each node c in the tree (i.e., clique in the induced graph). From this, we can easily extract the node and edge marginals, $p(x_t|\mathbf{v})$ and $p(x_s, x_t|\mathbf{v})$ from the original model, by marginalizing the clique distributions.[3]

20.4.2 Message passing on a junction tree

Having constructed a junction tree, we can use it for inference. The process is very similar to belief propagation on a tree. As in Section 20.2, there are two versions: the sum-product form, also known as the **Shafer-Shenoy** algorithm, named after (Shafer and Shenoy 1990); and the belief updating form (which involves division), also known as the **Hugin** (named after a company) or the **Lauritzen-Spiegelhalter** algorithm (named after (Lauritzen and Spiegelhalter 1988)). See (Lepar and Shenoy 1998) for a detailed comparison of these methods. Below we sketch how the Hugin algorithm works.

We assume the original model has the following form:

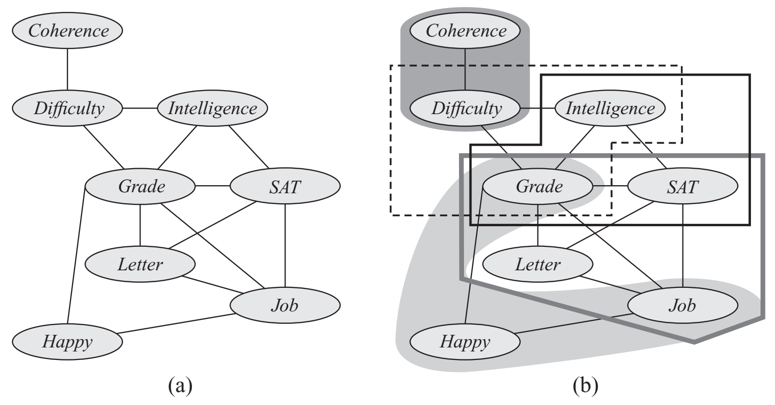
$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}(G)} \psi_c(\mathbf{x}_c) \tag{20.58}$$

where $\mathcal{C}(G)$ are the cliques of the original graph. On the other hand, the tree defines a distribution of the following form:

$$p(\mathbf{x}) = \frac{\prod_{c \in \mathcal{C}(T)} \psi_c(\mathbf{x}_c)}{\prod_{s \in \mathcal{S}(T)} \psi_s(\mathbf{x}_s)} \tag{20.59}$$

2. The largest loop in a chordal graph is length 3. Consequently chordal graphs are sometimes called **triangulated**. However, it is not enough for the graph to look like it is made of little triangles. For example, Figure 20.6(a) is not chordal, even though it is made of little triangles, since it contains the chordless 5-cycle 1-2-3-4-5-1.

3. If we want the joint distribution of some variables that are not in the same clique — a so-called **out-of-clique query** — we can adapt the technique described in Section 20.2.4.3 as follows: create a mega node containing the query variables and any other nuisance variables that lie on the path between them, multiply in messages onto the boundary of the mega node, and then marginalize out the internal nuisance variables. This internal marginalization may require the use of BP or VE. See (Koller and Friedman 2009) for details.



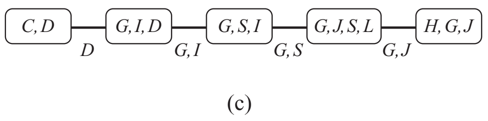


Figure 20.7 (a) The student graph with fill-in edges added. (b) The maximal cliques. (c) The junction tree. An edge between nodes s and t is labeled by the intersection of the sets on nodes s and t; this is called the **separating set**. From Figure 9.11 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

where $\mathcal{C}(T)$ are the nodes of the junction tree (which are the cliques of the chordal graph), and $\mathcal{S}(T)$ are the separators of the tree. To make these equal, we initialize by defining $\psi_s = 1$ for all separators and $\psi_c = 1$ for all cliques. Then, for each clique in the original model, $c \in \mathcal{C}(G)$, we find a clique in the tree $c' \in \mathcal{C}(T)$ which contains it, $c' \supseteq c$. We then multiply ψ_c onto $\psi_{c'}$ by computing $\psi_{c'} = \psi_{c'}\ \psi_c$. After doing this for all the cliques in the original graph, we have

$$\prod_{c \in \mathcal{C}(T)} \psi_c(\mathbf{x}_c) = \prod_{c \in \mathcal{C}(G)} \psi_c(\mathbf{x}_c) \tag{20.60}$$

As in Section 20.2.1, we now send messages from the leaves to the root and back, as sketched in Figure 20.1. In the upwards pass, also known as the **collect-to-root** phase, node i sends to its parent j the following message:

$$m_{i \to j}(S_{ij}) = \sum_{C_i \backslash S_{ij}} \psi_i(C_i) \tag{20.61}$$

That is, we marginalize out the variables that node i "knows about" which are irrelevant to j, and then we send what is left over. Once a node has received messages from all its children, it updates its belief state using

$$\psi_i(C_i) \propto \psi_i(C_i) \prod_{j \in \text{ch}_i} m_{j \to i}(S_{ij}) \tag{20.62}$$

At the root, $\psi_r(C_r)$ represents $p(\mathbf{x}_{C_r}|\mathbf{v})$, which is the posterior over the nodes in clique C_r conditioned on all the evidence. Its normalization constant is $p(\mathbf{v})/Z_0$, where Z_0 is the normalization constant for the unconditional prior, $p(\mathbf{x})$. (We have $Z_0 = 1$ if the original model was a DGM.)

In the downwards pass, also known as the **distribute-from-root** phase, node i sends to its children j the following message:

$$m_{i\to j}(S_{ij}) = \frac{\sum_{C_i \backslash S_{ij}} \psi_i(C_i)}{m_{j\to i}(S_{ij})} \tag{20.63}$$

We divide out by what j sent to i to avoid double counting the evidence. This requires that we store the messages from the upwards pass. Once a node has received a top-down message from its parent, it can compute its final belief state using

$$\psi_j(C_j) \propto \psi_j(C_j) m_{i\to j}(S_{ij}) \tag{20.64}$$

An equivalent way to present this algorithm is based on storing the messages inside the separator potentials. So on the way up, sending from i to j we compute the separator potential

$$\psi^*_{ij}(S_{ij}) = \sum_{C_i\backslash S_{ij}} \psi_i(C_i) \tag{20.65}$$

and then update the recipient potential:

$$\psi^*_j(C_j) \propto \psi_j(C_j) \frac{\psi^*_{ij}(S_{ij})}{\psi_{ij}(S_{ij})} \tag{20.66}$$

(Recall that we initialize $\psi_{ij}(S_{ij}) = 1$.) This is sometimes called **passing a flow** from i to j. On the way down, from i to j, we compute the separator potential

$$\psi^{**}_{ij}(S_{ij}) = \sum_{C_i\backslash S_{ij}} \psi^*_i(C_i) \tag{20.67}$$

and then update the recipient potential:

$$\psi^{**}_j(C_j) \propto \psi^*_j(C_j) \frac{\psi^{**}_{ij}(S_{ij})}{\psi^*_{ij}(S_{ij})} \tag{20.68}$$

This process is known as junction tree **calibration**. See Figure 20.1 for an illustration. Its correctness follows from the fact that each edge partitions the evidence into two distinct groups, plus the fact that the tree satisfies RIP, which ensures that no information is lost by only performing local computations.

20.4.2.1 Example: jtree algorithm on a chain

It is interesting to see what happens if we apply this process to a chain structured graph such as an HMM. A detailed discussion can be found in (Smyth et al. 1997), but the basic idea is this. The cliques are the edges, and the separators are the nodes, as shown in Figure 20.8. We initialize the potentials as follows: we set $\psi_s = 1$ for all the separators, we set $\psi_c(x_{t-1}, x_t) = p(x_t|x_{t-1})$ for clique $c = (X_{t-1}, X_t)$, and we set $\psi_c(x_t, y_t) = p(y_t|x_t)$ for clique $c = (X_t, Y_t)$.

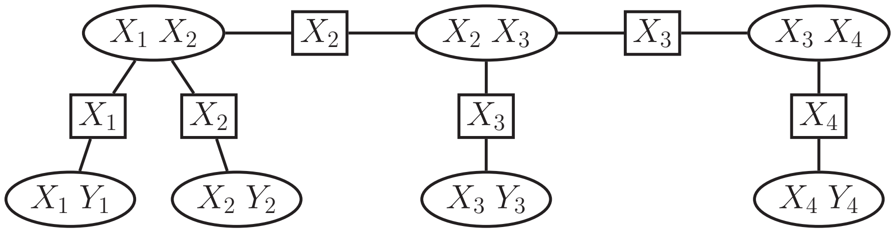


Figure 20.8 The junction tree derived from an HMM/SSM of length $T = 4$.

Next we send messages from left to right. Consider clique (X_{t-1}, X_t) with potential $p(X_t|X_{t-1})$. It receives a message from clique (X_{t-2}, X_{t-1}) via separator X_{t-1} of the form $\sum_{x_{t-2}} p(X_{t-2}, X_{t-1}|\mathbf{v}_{1:t-1}) = p(X_{t-1}|\mathbf{v}_{1:t-1})$. When combined with the clique potential, this becomes the two-slice predictive density

$$p(X_t|X_{t-1})p(X_{t-1}|\mathbf{v}_{1:t-1}) = p(X_{t-1}, X_t|\mathbf{v}_{1:t-1} \tag{20.69}$$

The clique (X_{t-1}, X_t) also receives a message from (X_t, Y_t) via separator X_t of the form $p(y_t|X_t)$, which corresponds to its local evidence. When combined with the updated clique potential, this becomes the two-slice filtered posterior

$$p(X_{t-1}, X_t|\mathbf{v}_{1:t-1})p(\mathbf{v}_t|X_t) = p(X_{t-1}, X_t|\mathbf{v}_{1:t}) \tag{20.70}$$

Thus the messages in the forwards pass are the filtered belief states α_t, and the clique potentials are the two-slice distributions. In the backwards pass, the messages are the update factors $\frac{\gamma_t}{\alpha_t}$, where $\gamma_t(k) = p(x_t = k|\mathbf{v}_{1:T})$ and $\alpha_t(k) = p(x_t = k|\mathbf{v}_{1:t})$. By multiplying by this message, we "swap out" the old α_t message and "swap in" the new γ_t message. We see that the backwards pass involves working with posterior beliefs, not conditional likelihoods. See Section 18.3.2.3 for further discussion of this difference.

20.4.3 Computational complexity of JTA

If all nodes are discrete with K states each, it is clear that the JTA takes $O(|\mathcal{C}|K^{w+1})$ time and space, where $|\mathcal{C}|$ is the number of cliques and w is the treewidth of the graph, i.e., the size of the largest clique minus 1. Unfortunately, choosing a triangulation so as to minimize the treewidth is NP-hard, as explained in Section 20.3.2.

The JTA can be modified to handle the case of Gaussian graphical models. The graph-theoretic steps remain unchanged. Only the message computation differs. We just need to define how to multiply, divide, and marginalize Gaussian potential functions. This is most easily done in information form. See e.g., (Lauritzen 1992; Murphy 1998; Cemgil 2001) for the details. The algorithm takes $O(|\mathcal{C}|w^3)$ time and $O(|\mathcal{C}|w^2)$ space. When applied to a chain structured graph, the algorithm is equivalent to the Kalman smoother in Section 18.3.2.

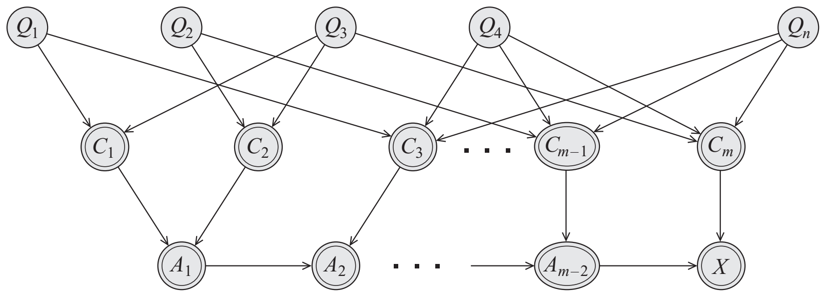


Figure 20.9 Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final x node has bounded size. The double rings denote nodes with deterministic CPDs. Source: Figure 9.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

20.4.4 JTA generalizations *

We have seen how to use the JTA algorithm to compute posterior marginals in a graphical model. There are several possible generalizations of this algorithm, some of which we mention below. All of these exploit graph decomposition in some form or other. They only differ in terms of how they define/ compute messages and "beliefs". The key requirement is that the operators which compute messages form a commutative semiring (see Section 20.3.1).

- Computing the MAP estimate. We just replace the sum-product with max-product in the collect phase, and use traceback in the distribute phase, as in the Viterbi algorithm (Section 17.4.4). See (Dawid 1992) for details.
- Computing the N-most probable configurations (Nilsson 1998).
- Computing posterior samples. The collect pass is the same as usual, but in the distribute pass, we sample variables given the values higher up in the tree, thus generalizing forwards-filtering backwards-sampling for HMMs described in Section 17.4.5. See (Dawid 1992) for details.
- Solving constraint satisfaction problems (Dechter 2003).
- Solving logical reasoning problems (Amir and McIlraith 2005).

20.5 Computational intractability of exact inference in the worst case

As we saw in Sections 20.3.2 and 20.4.3, VE and JTA take time that is exponential in the treewidth of a graph. Since the treewidth can be O(number of nodes) in the worst case, this means these algorithms can be exponential in the problem size.

Of course, just because VE and JTA are 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 is NP-hard (Dagum and Luby 1993). The proof is a simple reduction from the satisfiability prob-

Method	Restriction	Section
Forwards-backwards	Chains, D or LG	Section 17.4.3
Belief propagation	Trees, D or LG	Section 20.2
Variable elimination	Low treewidth, D or LG, single query	Section 20.3
Junction tree algorithm	Low treewidth, D or LG	Section 20.4
Loopy belief propagation	Approximate, D or LG	Section 22.2
Convex belief propagation	Approximate, D or LG	Section 22.4.2
Mean field	Approximate, C-E	Section 21.3
Gibbs sampling	Approximate	Section 24.2

Table 20.4 Summary of some methods that can be used for inference in graphical models. "D" means that all the hidden variables must be discrete. "L-G" means that all the factors must be linear-Gaussian. The term "single query" refers to the restriction that VE only computes one marginal $p(\mathbf{x}_q|\mathbf{x}_v)$ at a time. See Section 20.3.3 for a discussion of this point. "C-E" stands for "conjugate exponential"; this means that variational mean field only applies to models where the likelihood is in the exponential family, and the prior is conjugate. This includes the D and LG case, but many others as well, as we will see in Section 21.5.

lem. In particular, note that we can encode any 3-SAT problem[4] as a DGM with deterministic links, as shown in Figure 20.9. We clamp the final node, x, to be on, and we arrange the CPTs so that $p(x = 1) > 0$ iff there a satisfying assignment. Computing any posterior marginal requires evaluating the normalization constant $p(x = 1)$, which represents the probability of the evidence, so inference in this model implicitly solves the SAT problem.

In fact, exact inference is #P-hard (Roth 1996), which is even harder than NP-hard. (See e.g., (Arora and Barak 2009) for definitions of these terms.) The intuitive reason for this is that to compute the normalizing constant Z, we have to *count* how many satisfying assignments there are. By contrast, MAP estimation is provably easier for some model classes (Greig et al. 1989), since, intuitively speaking, it only requires finding one satisfying assignment, not counting all of them.

20.5.1 Approximate inference

Many popular probabilistic models support efficient exact inference, since they are based on chains, trees or low treewidth graphs. But there are many other models for which exact inference is intractable. In fact, even simple two node models of the form $\boldsymbol{\theta} \rightarrow \mathbf{x}$ may not support exact inference if the prior on $\boldsymbol{\theta}$ is not conjugate to the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$.[5]

Therefore we will need to turn to **approximate inference** methods. See Table 20.4 for a summary of coming attractions. For the most part, these methods do not come with any guarantee as to their accuracy or running time. Theoretical computer scientists would therefore describe them as **heuristics** rather than approximation algorithms. In fact, one can prove that

4. A 3-SAT problem is a logical expression of the form $(Q_1 \wedge Q_2 \wedge \neg Q_3) \vee (Q_1 \wedge \neg Q_4 \wedge Q_5) \cdots$, where the Q_i are binary variables, and each clause consists of the conjunction of three variables (or their negation). The goal is to find a satisfying assignment, which is a set of values for the Q_i variables such that the expression evaluates to true.

5. For discrete random variables, conjugacy is not a concern, since discrete distributions are always closed under conditioning and marginalization. Consequently, graph-theoretic considerations are of more importance when discussing inference in models with discrete hidden states.

it is not possible to construct polynomial time approximation schemes for inference in general discrete GMs (Dagum and Luby 1993; Roth 1996). Fortunately, we will see that for many of these heuristic methods often perform well in practice.

Exercises

Exercise 20.1 Variable elimination

Consider the MRF in Figure 10.14(b).

a. Suppose we want to compute the partition function using the elimination ordering $\prec = (1, 2, 3, 4, 5, 6)$, i.e.,

$$\sum_{x_6}\sum_{x_5}\sum_{x_4}\sum_{x_3}\sum_{x_2}\sum_{x_1} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{34}(x_3, x_4)\psi_{45}(x_4, x_5)\psi_{56}(x_5, x_6) \tag{20.71}$$

If we use the variable elimination algorithm, we will create new intermediate factors. What is the largest intermediate factor?

b. Add an edge to the original MRF between every pair of variables that end up in the same factor. (These are called fill in edges.) Draw the resulting MRF. What is the size of the largest maximal clique in this graph?

c. Now consider elimination ordering $\prec = (4, 1, 2, 3, 5, 6)$, i.e.,

$$\sum_{x_6}\sum_{x_5}\sum_{x_3}\sum_{x_2}\sum_{x_1}\sum_{x_4} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{34}(x_3, x_4)\psi_{45}(x_4, x_5)\psi_{56}(x_5, x_6) \tag{20.72}$$

If we use the variable elimination algorithm, we will create new intermediate factors. What is the largest intermediate factor?

d. Add an edge to the original MRF between every pair of variables that end up in the same factor. (These are called fill in edges.) Draw the resulting MRF. What is the size of the largest maximal clique in this graph?

Exercise 20.2 Gaussian times Gaussian is Gaussian

Prove Equation 20.17. Hint: use completing the square.

Exercise 20.3 Message passing on a tree

Consider the DGM in Figure 20.10 which represents the following fictitious biological model. Each G_i represents the genotype of a person: $G_i = 1$ if they have a healthy gene and $G_i = 2$ if they have an unhealthy gene. G_2 and G_3 may inherit the unhealthy gene from their parent G_1. $X_i \in \mathbb{R}$ is a continuous measure of blood pressure, which is low if you are healthy and high if you are unhealthy. We define the CPDs as follows

$$p(G_1) = [0.5, 0.5] \tag{20.73}$$

$$p(G_2|G_1) = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \tag{20.74}$$

$$p(G_3|G_1) = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \tag{20.75}$$

$$p(X_i|G_i = 1) = \mathcal{N}(X_i|\mu = 50, \sigma^2 = 10) \tag{20.76}$$

$$p(X_i|G_i = 2) = \mathcal{N}(X_i|\mu = 60, \sigma^2 = 10) \tag{20.77}$$

The meaning of the matrix for $p(G_2|G_1)$ is that $p(G_2 = 1|G_1 = 1) = 0.9$, $p(G_2 = 1|G_1 = 2) = 0.1$, etc.

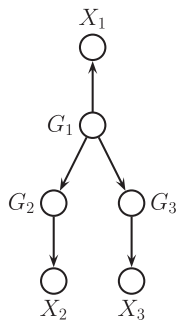


Figure 20.10 A simple DAG representing inherited diseases.

a. Suppose you observe $X_2 = 50$, and X_1 is unobserved. What is the posterior belief on G_1, i.e., $p(G_1|X_2 = 50)$?
b. Now suppose you observe $X_2 = 50$ amd $X_3 = 50$. What is $p(G_1|X_2, X_3)$? Explain your answer intuitively.
c. Now suppose $X_2 = 60$, $X_3 = 60$. What is $p(G_1|X_2, X_3)$? Explain your answer intuitively.
d. Now suppose $X_2 = 50$, $X_3 = 60$. What is $p(G_1|X_1, X_2)$? Explain your answer intuitively.

Exercise 20.4 Inference in 2D lattice MRFs

Consider an MRF with a 2D $m \times n$ lattice graph structure, so each hidden node, X_{ij}, is connected to its 4 nearest neighbors, as in an Ising model. In addition, each hidden node has its own local evidence, Y_{ij}. Assume all hidden nodes have $K > 2$ states. In general, exact inference in such models is intractable, because the maximum cliques of the corresponding triangulated graph have size $O(\max\{m, n\})$. Suppose $m \ll n$ i.e., the lattice is short and fat.

a. How can one *efficiently* perform exact inference (using a deterministic algorithm) in such models? (By exact inference, I mean computing marginal probabilities $P(X_{ij}|\vec{y})$ exactly, where $\vec{y}$ is all the evidence.) Give a *brief* description of your method.
b. What is the asymptotic complexity (running time) of your algorithm?
c. Now suppose the lattice is large and square, so $m = n$, but all hidden states are binary (ie $K = 2$). In this case, how can one efficiently exactly compute (using a deterministic algorithm) the MAP estimate $\arg\max_x P(x|y)$, where x is the joint assignment to all hidden nodes?

21 *Variational inference*

21.1 Introduction

We have now seen several algorithms for computing (functions of) a posterior distribution. For discrete graphical models, we can use the junction tree algorithm to perform exact inference, as explained in Section 20.4. However, this takes time exponential in the treewidth of the graph, rendering exact inference often impractical. For the case of Gaussian graphical models, exact inference is cubic in the treewidth. However, even this can be too slow if we have many variables. In addition, the JTA does not work for continuous random variables outside of the Gaussian case, nor for mixed discrete-continuous variables, outside of the conditionally Gaussian case.

For some simple two node graphical models, of the form $\mathbf{x} \rightarrow \mathcal{D}$, we can compute the exact posterior $p(\mathbf{x}|\mathcal{D})$ in closed form, provided the prior $p(\mathbf{x})$ is conjugate to the likelihood, $p(\mathcal{D}|\mathbf{x})$ (which means the likelihood must be in the exponential family). See Chapter 5 for some examples of this. (Note that in this chapter, $\mathbf{x}$ represent the unknown variables, whereas in Chapter 5, we used $\boldsymbol{\theta}$ to represent the unknowns.)

In more general settings, we must use approximate inference methods. In Section 8.4.1, we discussed the Gaussian approximation, which is useful for inference in two node models of the form $\mathbf{x} \rightarrow \mathcal{D}$, where the prior is not conjugate. (For example, Section 8.4.3 applied the method to logistic regression.)

The Gaussian approximation is simple. However, some posteriors are not naturally modelled using Gaussians. For example, when inferring multinomial parameters, a Dirichlet distribution is a better choice, and when inferring states in a discrete graphical model, a categorical distribution is a better choice.

In this chapter, we will study a more general class of deterministic approximate inference algorithms based on **variational inference** (Jordan et al. 1998; Jaakkola and Jordan 2000; Jaakkola 2001; Wainwright and Jordan 2008a). The basic idea is to pick an approximation $q(\mathbf{x})$ to the distribution from some tractable family, and then to try to make this approximation as close as possible to the true posterior, $p^*(\mathbf{x}) \triangleq p(\mathbf{x}|\mathcal{D})$. This reduces inference to an optimization problem. By relaxing the constraints and/or approximating the objective, we can trade accuracy for speed. The bottom line is that variational inference often gives us the speed benefits of MAP estimation but the statistical benefits of the Bayesian approach.

21.2 Variational inference

Suppose $p^*(\mathbf{x})$ is our true but intractable distribution and $q(\mathbf{x})$ is some approximation, chosen from some tractable family, such as a multivariate Gaussian or a factored distribution. We assume q has some free parameters which we want to optimize so as to make q "similar to" p^*.

An obvious cost function to try to minimize is the KL divergence:

$$\mathbb{KL}\left(p^*||q\right) = \sum_{\mathbf{x}} p^*(\mathbf{x}) \log \frac{p^*(\mathbf{x})}{q(\mathbf{x})} \tag{21.1}$$

However, this is hard to compute, since taking expectations wrt p^* is assumed to be intractable. A natural alternative is the reverse KL divergence:

$$\mathbb{KL}\left(q||p^*\right) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p^*(\mathbf{x})} \tag{21.2}$$

The main advantage of this objective is that computing expectations wrt q is tractable (by choosing a suitable form for q). We discuss the statistical differences between these two objectives in Section 21.2.2.

Unfortunately, Equation 21.2 is still not tractable as written, since even evaluating $p^*(\mathbf{x}) = p(\mathbf{x}|\mathcal{D})$ pointwise is hard, since it requires evaluating the intractable normalization constant $Z = p(\mathcal{D})$. However, usually the unnormalized distribution $\tilde{p}(\mathbf{x}) \triangleq p(\mathbf{x}, \mathcal{D}) = p^*(\mathbf{x})Z$ is tractable to compute. We therefore define our new objective function as follows:

$$J(q) \quad \triangleq \quad \mathbb{KL}\left(q||\tilde{p}\right) \tag{21.3}$$

where we are slightly abusing notation, since $\tilde{p}$ is not a normalized distribution. Plugging in the definition of KL, we get

$$\begin{aligned}
J(q) &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})} && (21.4)\\
&= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{Zp^*(\mathbf{x})} && (21.5)\\
&= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p^*(\mathbf{x})} - \log Z && (21.6)\\
&= \mathbb{KL}\left(q||p^*\right) - \log Z && (21.7)
\end{aligned}$$

Since Z is a constant, by minimizing $J(q)$, we will force q to become close to p^*.

Since KL divergence is always non-negative, we see that $J(q)$ is an upper bound on the NLL (negative log likelihood):

$$J(q) = \mathbb{KL}\left(q||p^*\right) - \log Z \geq -\log Z = -\log p(\mathcal{D}) \tag{21.8}$$

Alternatively, we can try to *maximize* the following quantity (in (Koller and Friedman 2009), this is referred to as the **energy functional**), which is a lower bound on the log likelihood of the data:

$$L(q) \triangleq -J(q) = -\mathbb{KL}\left(q||p^*\right) + \log Z \leq \log Z = \log p(\mathcal{D}) \tag{21.9}$$

Since this bound is tight when $q = p^*$, we see that variational inference is closely related to EM (see Section 11.4.7).

21.2.1 Alternative interpretations of the variational objective

There are several equivalent ways of writing this objective that provide different insights. One formulation is as follows:

$$J(q) = \mathbb{E}_q [\log q(\mathbf{x})] + \mathbb{E}_q [-\log \tilde{p}(\mathbf{x})] = -\mathbb{H}(q) + \mathbb{E}_q [E(\mathbf{x})] \tag{21.10}$$

which is the expected energy (recall $E(\mathbf{x}) = -\log \tilde{p}(\mathbf{x})$) minus the entropy of the system. In statistical physics, $J(q)$ is called the **variational free energy** or the **Helmholtz free energy**.[1]

Another formulation of the objective is as follows:

$$\begin{aligned} J(q) &= \mathbb{E}_q [\log q(\mathbf{x}) - \log p(\mathbf{x})p(\mathcal{D}|\mathbf{x})] & (21.11)\\ &= \mathbb{E}_q [\log q(\mathbf{x}) - \log p(\mathbf{x}) - \log p(\mathcal{D}|\mathbf{x})] & (21.12)\\ &= \mathbb{E}_q [-\log p(\mathcal{D}|\mathbf{x})] + \mathbb{KL}(q(\mathbf{x})||p(\mathbf{x})) & (21.13) \end{aligned}$$

This is the expected NLL, plus a penalty term that measures how far the approximate posterior is from the exact prior.

We can also interpret the variational objective from the point of view of information theory (the so-called bits-back argument). See (Hinton and Camp 1993; Honkela and Valpola 2004), for details.

21.2.2 Forward or reverse KL? *

Since the KL divergence is not symmetric in its arguments, minimizing $\mathbb{KL}(q||p)$ wrt q will give different behavior than minimizing $\mathbb{KL}(p||q)$. Below we discuss these two different methods.

First, consider the reverse KL, $\mathbb{KL}(q||p)$, also known as an **I-projection** or **information projection**. By definition, we have

$$\mathbb{KL}(q||p) = \sum_{\mathbf{x}} q(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})} \tag{21.14}$$

This is infinite if $p(\mathbf{x}) = 0$ and $q(\mathbf{x}) > 0$. Thus if $p(\mathbf{x}) = 0$ we must ensure $q(\mathbf{x}) = 0$. We say that the reverse KL is **zero forcing** for q. Hence q will typically under-estimate the support of p.

Now consider the forwards KL, also known as an **M-projection** or **moment projection**:

$$\mathbb{KL}(p||q) = \sum_{\mathbf{x}} p(\mathbf{x}) \ln \frac{p(\mathbf{x})}{q(\mathbf{x})} \tag{21.15}$$

This is infinite if $q(\mathbf{x}) = 0$ and $p(\mathbf{x}) > 0$. So if $p(\mathbf{x}) > 0$ we must ensure $q(\mathbf{x}) > 0$. We say that the forwards KL is **zero avoiding** for q. Hence q will typically over-estimate the support of p.

The difference between these methods is illustrated in Figure 21.1. We see that when the true distribution is multimodal, using the forwards KL is a bad idea (assuming q is constrained to be unimodal), since the resulting posterior mode/mean will be in a region of low density, right between the two peaks. In such contexts, the reverse KL is not only more tractable to compute, but also more sensible statistically.

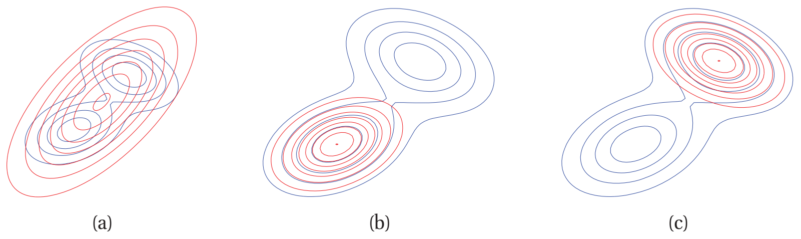


Figure 21.1 Illustrating forwards vs reverse KL on a bimodal distribution. The blue curves are the contours of the true distribution p. The red curves are the contours of the unimodal approximation q. (a) Minimizing forwards KL: q tends to "cover" p. (b-c) Minimizing reverse KL: q locks on to one of the two modes. Based on Figure 10.3 of (Bishop 2006b). Figure generated by `KLfwdReverseMixGauss`.

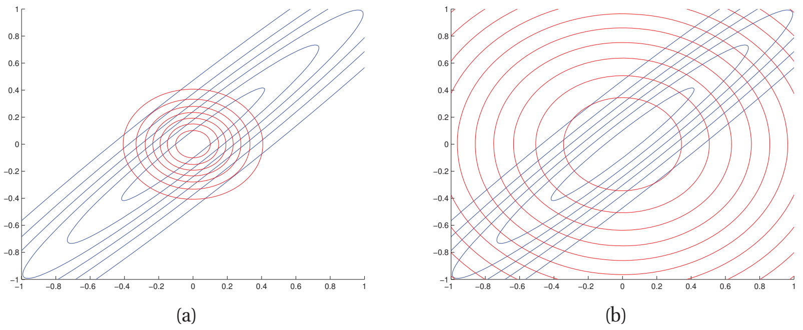


Figure 21.2 Illustrating forwards vs reverse KL on a symmetric Gaussian. The blue curves are the contours of the true distribution p. The red curves are the contours of a factorized approximation q. (a) Minimizing $\mathbb{KL}(q||p)$. (b) Minimizing $\mathbb{KL}(p||q)$. Based on Figure 10.2 of (Bishop 2006b). Figure generated by `KLpqGauss`.

Another example of the difference is shown in Figure 21.2, where the target distribution is an elongated 2d Gaussian and the approximating distribution is a product of two 1d Gaussians. That is, $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$, where

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix} \tag{21.16}$$

In Figure 21.2(a) we show the result of minimizing $\mathbb{KL}(q||p)$. In this simple example, one can show that the solution has the form

$$q(\mathbf{x}) = \mathcal{N}(x_1|m_1, \Lambda_{11}^{-1})\mathcal{N}(x_2|m_2, \Lambda_{22}^{-1}) \tag{21.17}$$

$$m_1 = \mu_1 - \Lambda_{11}^{-1}\Lambda_{12}(m_2 - \mu_2) \tag{21.18}$$

$$m_2 = \mu_2 - \Lambda_{22}^{-1}\Lambda_{21}(m_1 - \mu_1) \tag{21.19}$$

1. It is called "free" because the variables $\mathbf{x}$ are free to vary, rather than being fixed. The variational free energy is a function of the distribution q, whereas the regular energy is a function of the state vector $\mathbf{x}$.

Figure 21.2(a) shows that we have correctly captured the mean, but the approximation is too compact: its variance is controlled by the direction of smallest variance of p. In fact, it is often the case (although not always (Turner et al. 2008)) that minimizing $\mathbb{KL}(q||p)$, where q is factorized, results in an approximation that is overconfident.

In Figure 21.2(b), we show the result of minimizing $\mathbb{KL}(p||q)$. As we show in Exercise 21.7, the optimal solution when minimizing the forward KL wrt a factored approximation is to set q to be the product of marginals. Thus the solution has the form

$$q(\mathbf{x}) = \mathcal{N}(x_1|\mu_1, \Lambda_{11}^{-1})\mathcal{N}(x_2|\mu_2, \Lambda_{22}^{-1}) \tag{21.20}$$

Figure 21.2(b) shows that this is too broad, since it is an over-estimate of the support of p.

For the rest of this chapter, and for most of the next, we will focus on minimizing $\mathbb{KL}(q||p)$. In Section 22.5, when we discuss expectation proagation, we will discuss ways to locally optimize $\mathbb{KL}(p||q)$.

One can create a family of divergence measures indexed by a parameter $\alpha \in \mathbb{R}$ by defining the **alpha divergence** as follows:

$$D_\alpha(p||q) \triangleq \frac{4}{1-\alpha^2}\left(1 - \int p(x)^{(1+\alpha)/2} q(x)^{(1-\alpha)/2} dx\right) \tag{21.21}$$

This measure satisfies $D_\alpha(p||q) = 0$ iff $p = q$, but is obviously not symmetric, and hence is not a metric. $\mathbb{KL}(p||q)$ corresponds to the limit $\alpha \to 1$, whereas $\mathbb{KL}(q||p)$ corresponds to the limit $\alpha \to -1$. When $\alpha = 0$, we get a symmetric divergence measure that is linearly related to the **Hellinger distance**, defined by

$$D_H(p||q) \triangleq \int \left(p(x)^{\frac{1}{2}} - q(x)^{\frac{1}{2}}\right)^2 dx \tag{21.22}$$

Note that $\sqrt{D_H(p||q)}$ is a valid distance metric, that is, it is symmetric, non-negative and satisfies the triangle inequality. See (Minka 2005) for details.

21.3 The mean field method

One of the most popular forms of variational inference is called the **mean field** approximation (Opper and Saad 2001). In this approach, we assume the posterior is a fully factorized approximation of the form

$$q(\mathbf{x}) = \prod_i q_i(\mathbf{x}_i) \tag{21.23}$$

Our goal is to solve this optimization problem:

$$\min_{q_1,\ldots,q_D} \mathbb{KL}(q||p) \tag{21.24}$$

where we optimize over the parameters of each marginal distribution q_i. In Section 21.3.1, we derive a coordinate descent method, where at each step we make the following update:

$$\log q_j(\mathbf{x}_j) = \mathbb{E}_{-q_j}[\log \tilde{p}(\mathbf{x})] + \text{const} \tag{21.25}$$

Model	Section
Ising model	Section 21.3.2
Factorial HMM	Section 21.4.1
Univariate Gaussian	Section 21.5.1
Linear regression	Section 21.5.2
Logistic regression	Section 21.8.1.1
Mixtures of Gaussians	Section 21.6.1
Latent Dirichlet allocation	Section 27.3.6.3

Table 21.1 Some models in this book for which we provide detailed derivations of the mean field inference algorithm.

where $\tilde{p}(\mathbf{x}) = p(\mathbf{x}, \mathcal{D})$ is the unnormalized posterior and the notation $\mathbb{E}_{-q_j}[f(\mathbf{x})]$ means to take the expectation over $f(\mathbf{x})$ with respect to all the variables except for x_j. For example, if we have three variables, then

$$\mathbb{E}_{-q_2}[f(\mathbf{x})] = \sum_{x_1} \sum_{x_3} q(x_1) q_3(x_3) f(x_1, x_2, x_3) \tag{21.26}$$

where sums get replaced by integrals where necessary.

When updating q_j, we only need to reason about the variables which share a factor with x_j, i.e., the terms in j's Markov blanket (see Section 10.5.3); the other terms get absorbed into the constant term. Since we are replacing the neighboring values by their mean value, the method is known as mean field. This is very similar to Gibbs sampling (Section 24.2), except instead of sending sampled values between neighboring nodes, we send mean values between nodes. This tends to be more efficient, since the mean can be used as a proxy for a large number of samples. (On the other hand, mean field messages are dense, whereas samples are sparse; this can make sampling more scalable to very large models.)

Of course, updating one distribution at a time can be slow, since it is a form of coordinate descent. Several methods have been proposed to speed up this basic approach, including using pattern search (Honkela et al. 2003), and techniques based on parameter expansion (Qi and Jaakkola 2008). However, we will not consider these methods in this chapter.

It is important to note that the mean field method can be used to infer discrete or continuous latent quantities, using a variety of parametric forms for q_i, as we will see below. This is in contrast to some of the other variational methods we will encounter later, which are more restricted in their applicability. Table 21.1 lists some of the examples of mean field that we cover in this book.

21.3.1 Derivation of the mean field update equations

Recall that the goal of variational inference is to minimize the upper bound $J(q) \geq -\log p(\mathcal{D})$. Equivalently, we can try to maximize the lower bound

$$L(q) \triangleq -J(q) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{\tilde{p}(\mathbf{x})}{q(\mathbf{x})} \leq \log p(\mathcal{D}) \tag{21.27}$$

We will do this one term at a time.

If we write the objective singling out the terms that involve q_j, and regarding all the other terms as constants, we get

$$L(q_j) = \sum_{\mathbf{x}} \prod_i q_i(\mathbf{x}_i) \left[\log \tilde{p}(\mathbf{x}) - \sum_k \log q_k(\mathbf{x}_k) \right] \tag{21.28}$$

$$= \sum_{\mathbf{x}_j} \sum_{\mathbf{x}_{-j}} q_j(\mathbf{x}_j) \prod_{i \neq j} q_i(\mathbf{x}_i) \left[\log \tilde{p}(\mathbf{x}) - \sum_k \log q_k(\mathbf{x}_k) \right] \tag{21.29}$$

$$= \sum_{\mathbf{x}_j} q_j(\mathbf{x}_j) \sum_{\mathbf{x}_{-j}} \prod_{i \neq j} q_i(\mathbf{x}_i) \log \tilde{p}(\mathbf{x})$$

$$- \sum_{\mathbf{x}_j} q_j(\mathbf{x}_j) \sum_{\mathbf{x}_{-j}} \prod_{i \neq j} q_i(\mathbf{x}_i) \left[\sum_{k \neq j} \log q_k(\mathbf{x}_k) + q_j(\mathbf{x}_j) \right] \tag{21.30}$$

$$= \sum_{\mathbf{x}_j} q_j(\mathbf{x}_j) \log f_j(\mathbf{x}_j) - \sum_{\mathbf{x}_j} q_j(\mathbf{x}_j) \log q_j(\mathbf{x}_j) + \text{const} \tag{21.31}$$

where

$$\log f_j(\mathbf{x}_j) \triangleq \sum_{\mathbf{x}_{-j}} \prod_{i \neq j} q_i(\mathbf{x}_i) \log \tilde{p}(\mathbf{x}) = \mathbb{E}_{-q_j} \left[\log \tilde{p}(\mathbf{x}) \right] \tag{21.32}$$

So we average out all the hidden variables except for $\mathbf{x}_j$. Thus we can rewrite $L(q_j)$ as follows:

$$L(q_j) = -\mathbb{KL}\left(q_j || f_j\right) \tag{21.33}$$

We can maximize L by minimizing this KL, which we can do by setting $q_j = f_j$, as follows:

$$q_j(\mathbf{x}_j) = \frac{1}{Z_j} \exp\left(\mathbb{E}_{-q_j} \left[\log \tilde{p}(\mathbf{x}) \right] \right) \tag{21.34}$$

We can usually ignore the local normalization constant Z_j, since we know q_j must be a normalized distribution. Hence we usually work with the form

$$\log q_j(\mathbf{x}_j) = \mathbb{E}_{-q_j} \left[\log \tilde{p}(\mathbf{x}) \right] + \text{const} \tag{21.35}$$

The functional form of the q_j distributions will be determined by the type of variables $\mathbf{x}_j$, as well as the form of the model. (This is sometimes called **free-form optimization**.) If x_j is a discrete random variable, then q_j will be a discrete distribution; if $\mathbf{x}_j$ is a continuous random variable, then q_j will be some kind of pdf. We will see examples of this below.

21.3.2 Example: mean field for the Ising model

Consider the image denoising example from Section 19.4.1, where $x_i \in \{-1, +1\}$ are the hidden pixel values of the "clean" image. We have a joint model of the form

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}) p(\mathbf{y}|\mathbf{x}) \tag{21.36}$$

where the prior has the form

$$p(\mathbf{x}) = \frac{1}{Z_0}\exp(-E_0(\mathbf{x})) \tag{21.37}$$

$$E_0(\mathbf{x}) = -\sum_{i=1}^{D}\sum_{j\in\text{nbr}_i} W_{ij}x_ix_j \tag{21.38}$$

and the likelihood has the form

$$p(\mathbf{y}|\mathbf{x}) = \prod_i p(\mathbf{y}_i|x_i) = \sum_i \exp(-L_i(x_i)) \tag{21.39}$$

Therefore the posterior has the form

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{Z}\exp(-E(\mathbf{x})) \tag{21.40}$$

$$E(\mathbf{x}) = E_0(\mathbf{x}) - \sum_i L_i(x_i) \tag{21.41}$$

We will now approximate this by a fully factored approximation

$$q(\mathbf{x}) = \prod_i q(x_i, \mu_i) \tag{21.42}$$

where μ_i is the mean value of node i. To derive the update for the variational parameter μ_i, we first write out $\log\tilde{p}(\mathbf{x}) = -E(\mathbf{x})$, dropping terms that do not involve x_i:

$$\log\tilde{p}(\mathbf{x}) = x_i\sum_{j\in\text{nbr}_i} W_{ij}x_j + L_i(x_i) + \text{const} \tag{21.43}$$

This only depends on the states of the neighboring nodes. Now we take expectations of this wrt $\prod_{j\neq i} q_j(x_j)$ to get

$$q_i(x_i) \propto \exp\left(x_i\sum_{j\in\text{nbr}_i} W_{ij}\mu_j + L_i(x_i)\right) \tag{21.44}$$

Thus we replace the states of the neighbors by their average values. Let

$$m_i = \sum_{j\in\text{nbr}_i} W_{ij}\mu_j \tag{21.45}$$

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

$$q_i(x_i = 1) = \frac{e^{m_i + L_i^+}}{e^{m_i+L_i^+} + e^{-m_i+L_i^-}} = \frac{1}{1+e^{-2m_i+L_i^- - L_i^+}} = \text{sigm}(2a_i) \tag{21.46}$$

$$a_i \triangleq m_i + 0.5(L_i^+ - L_i^-) \tag{21.47}$$

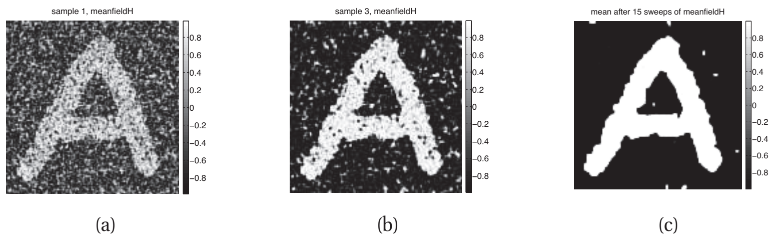


Figure 21.3 Example of image denoising using mean field (with parallel updates and a damping factor of 0.5). We use an Ising prior with $W_{ij} = 1$ and a Gaussian noise model with $\sigma = 2$. We show the results after 1, 3 and 15 iterations across the image. Compare to Figure 24.1. Figure generated by `isingImageDenoiseDemo`.

Similarly, we have $q_i(x_i = -1) = \text{sigm}(-2a_i)$. From this we can compute the new mean for site i:

$$\mu_i = \mathbb{E}_{q_i}[x_i] = q_i(x_i = +1) \cdot (+1) + q_i(x_i = -1) \cdot (-1) \tag{21.48}$$

$$= \frac{1}{1+e^{-2a_i}} - \frac{1}{1+e^{2a_i}} = \frac{e^{a_i}}{e^{a_i}+e^{-a_i}} - \frac{e^{-a_i}}{e^{-a_i}+e^{a_i}} = \tanh(a_i) \tag{21.49}$$

Hence the update equation becomes

$$\mu_i = \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j + 0.5(L_i^+ - L_i^-)\right) \tag{21.50}$$

See also Exercise 21.6 for an alternative derivation of these equations.

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

$$\mu_i^t = \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{21.51}$$

It is usually better to use **damped updates** of the form

$$\mu_i^t = (1-\lambda)\mu_i^{t-1} + \lambda \tanh\left(\sum_{j \in \text{nbr}_i} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)\right) \tag{21.52}$$

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

Figure 21.3 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.)

21.4 Structured mean field *

Assuming that all the variables are independent in the posterior is a very strong assumption that can lead to poor results. Sometimes we can exploit **tractable substructure** in our problem, so

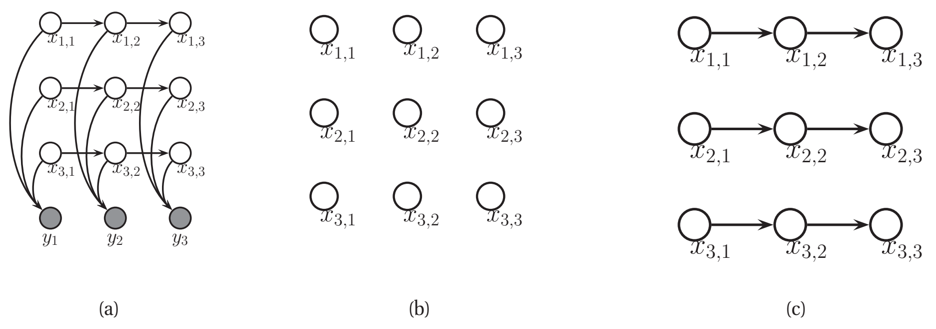


Figure 21.4 (a) A factorial HMM with 3 chains. (b) A fully factorized approximation. (c) A product-of-chains approximation. Based on Figure 2 of (Ghahramani and Jordan 1997).

that we can efficiently handle some kinds of dependencies. This is called the **structured mean field** approach (Saul and Jordan 1995). The approach is the same as before, except we group sets of variables together, and we update them simultaneously. (This follows by simply treating all the variables in the i'th group as a single "mega-variable", and then repeating the derivation in Section 21.3.1.) As long as we can perform efficient inference in each q_i, the method is tractable overall. We give an example below. See (Bouchard-Cote and Jordan 2009) for some more recent work in this area.

21.4.1 Example: factorial HMM

Consider the factorial HMM model (Ghahramani and Jordan 1997) introduced in Section 17.6.5. Suppose there are M chains, each of length T, and suppose each hidden node has K states. The model is defined as follows

$$p(\mathbf{x}, \mathbf{y}) = \prod_m \prod_t p(x_{tm}|x_{t-1,m}) p(\mathbf{y}_t|x_{tm}) \tag{21.53}$$

where $p(x_{tm} = k|x_{t-1,m} = j) = A_{mjk}$ is an entry in the transition matrix for chain m, $p(x_{1m} = k|x_{0m}) = p(x_{1m} = k) = \pi_{mk}$, is the initial state distribution for chain m, and

$$p(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{y}_t | \sum_{m=1}^{M} \mathbf{W}_m \mathbf{x}_{tm}, \mathbf{\Sigma}\right) \tag{21.54}$$

is the observation model, where $\mathbf{x}_{tm}$ is a 1-of-K encoding of x_{tm} and $\mathbf{W}_m$ is a $D \times K$ matrix (assuming $\mathbf{y}_t \in \mathbb{R}^D$). Figure 21.4(a) illustrates the model for the case where $M = 3$. Even though each chain is a priori independent, they become coupled in the posterior due to having an observed common child, $\mathbf{y}_t$. The junction tree algorithm applied to this graph takes $O(TMK^{M+1})$ time. Below we will derive a structured mean field algorithm that takes $O(TMK^2I)$ time, where I is the number of mean field iterations (typically $I \sim 10$ suffices for good performance).

We can write the exact posterior in the following form:

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{Z}\exp(-E(\mathbf{x},\mathbf{y})) \tag{21.55}$$

$$\begin{aligned} E(\mathbf{x},\mathbf{y}) &= \frac{1}{2}\sum_{t=1}^{T}\left(\mathbf{y}_t - \sum_m \mathbf{W}_m\mathbf{x}_{tm}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_t - \sum_m \mathbf{W}_m\mathbf{x}_{tm}\right) \\ &\quad - \sum_m \mathbf{x}_{1m}^T\tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^{T}\sum_m \mathbf{x}_{tm}^T\tilde{\mathbf{A}}_m\mathbf{x}_{t-1,m} \end{aligned} \tag{21.56}$$

where $\tilde{\mathbf{A}}_m \triangleq \log \mathbf{A}_m$ and $\tilde{\boldsymbol{\pi}}_m \triangleq \log \boldsymbol{\pi}_m$ (both interpreted elementwise).

We can approximate the posterior as a product of marginals, as in Figure 21.4(b), but a better approximation is to use a product of chains, as in Figure 21.4(c). Each chain can be tractably updated individually, using the forwards-backwards algorithm. More precisely, we assume

$$q(\mathbf{x}|\mathbf{y}) = \frac{1}{Z_q}\prod_{m=1}^{M} q(x_{1m}|\boldsymbol{\xi}_{1m})\prod_{t=2}^{T} q(x_{tm}|x_{t-1,m},\boldsymbol{\xi}_{tm}) \tag{21.57}$$

$$q(x_{1m}|\boldsymbol{\xi}_{1m}) = \prod_{k=1}^{K}(\xi_{1mk}\pi_{mk})^{x_{1mk}} \tag{21.58}$$

$$q(x_{tm}|x_{t-1,m},\boldsymbol{\xi}_{tm}) = \prod_{k=1}^{K}\left(\xi_{tmk}\prod_{j=1}^{K}(A_{mjk})^{x_{t-1,m,j}}\right)^{x_{tmk}} \tag{21.59}$$

We see that the ξ_{tmk} parameters play the role of an approximate local evidence, averaging out the effects of the other chains. This is contrast to the exact local evidence, which couples all the chains together.

We can rewrite the approximate posterior as $q(\mathbf{x}) = \frac{1}{Z_q}\exp(-E_q(\mathbf{x}))$, where

$$E_q(\mathbf{x}) = -\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbf{x}_{tm}^T\tilde{\boldsymbol{\xi}}_{tm} - \sum_{m=1}^{M}\mathbf{x}_{1m}^T\tilde{\boldsymbol{\pi}}_m - \sum_{t=2}^{T}\sum_{m=1}^{M}\mathbf{x}_{tm}^T\tilde{\mathbf{A}}_m\mathbf{x}_{t-1,m} \tag{21.60}$$

where $\tilde{\boldsymbol{\xi}}_{tm} = \log \boldsymbol{\xi}_{tm}$. We see that this has the same temporal factors as the exact posterior, but the local evidence term is different. The objective function is given by

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}\left[E\right] - \mathbb{E}\left[E_q\right] - \log Z_q + \log Z \tag{21.61}$$

where the expectations are taken wrt q. One can show (Exercise 21.8) that the update has the form

$$\boldsymbol{\xi}_{tm} = \exp\left(\mathbf{W}_m^T\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{y}}_{tm} - \frac{1}{2}\boldsymbol{\delta}_m\right) \tag{21.62}$$

$$\boldsymbol{\delta}_m \triangleq \text{diag}(\mathbf{W}_m^T\boldsymbol{\Sigma}^{-1}\mathbf{W}_m) \tag{21.63}$$

$$\tilde{\mathbf{y}}_{tm} \triangleq \mathbf{y}_t - \sum_{\ell \neq m}^{M}\mathbf{W}_\ell\mathbb{E}\left[\mathbf{x}_{t,\ell}\right] \tag{21.64}$$

The $\boldsymbol{\xi}_{tm}$ parameter plays the role of the local evidence, averaging over the neighboring chains. Having computed this for each chain, we can perform forwards-backwards in parallel, using these approximate local evidence terms to compute $q(\mathbf{x}_{t,m}|\mathbf{y}_{1:T})$ for each m and t.

The update cost is $O(TMK^2)$ for a full "sweep" over all the variational parameters, since we have to run forwards-backwards M times, for each chain independently. This is the same cost as a fully factorized approximation, but is much more accurate.

21.5 Variational Bayes

So far we have been concentrating on inferring latent variables $\mathbf{z}_i$ assuming the parameters $\boldsymbol{\theta}$ of the model are known. Now suppose we want to infer the parameters themselves. If we make a fully factorized (i.e., mean field) approximation, $p(\boldsymbol{\theta}|\mathcal{D}) \approx \prod_k q(\boldsymbol{\theta}_k)$, we get a method known as **variational Bayes** or **VB** (Hinton and Camp 1993; MacKay 1995a; Attias 2000; Beal and Ghahramani 2006; Smidl and Quinn 2005).[2] We give some examples of VB below, assuming that there are no latent variables. If we want to infer both latent variables and parameters, and we make an approximation of the form $p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta}) \prod_i q_i(\mathbf{z}_i)$, we get a method known as variational Bayes EM, which we described in Section 21.6.

21.5.1 Example: VB for a univariate Gaussian

Following (MacKay 2003, p429), let us consider how to apply VB to infer the posterior over the parameters for a 1d Gaussian, $p(\mu, \lambda|\mathcal{D})$, where $\lambda = 1/\sigma^2$ is the precision. For convenience, we will use a conjugate prior of the form

$$p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, (\kappa_0\lambda)^{-1})\mathrm{Ga}(\lambda|a_0, b_0) \tag{21.65}$$

However, we will use an approximate factored posterior of the form

$$q(\mu, \lambda) = q_\mu(\mu)q_\lambda(\lambda) \tag{21.66}$$

We do not need to specify the forms for the distributions q_μ and q_λ; the optimal forms will "fall out" automatically during the derivation (and conveniently, they turn out to be Gaussian and Gamma respectively).

You might wonder why we would want to do this, since we know how to compute the exact posterior for this model (Section 4.6.3.7). There are two reasons. First, it is a useful pedagogical exercise, since we can compare the quality of our approximation to the exact posterior. Second, it is simple to modify the method to handle a semi-conjugate prior of the form $p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, \tau_0)\mathrm{Ga}(\lambda|a_0, b_0)$, for which exact inference is no longer possible.

2. This method was originally called **ensemble learning** (MacKay 1995a), since we are using an ensemble of parameters (a distribution) instead of a point estimate. However, the term "ensemble learning" is also used to describe methods such as boosting, so we prefer the term VB.

21.5.1.1 Target distribution

The unnormalized log posterior has the form

$$\begin{aligned}\log \tilde{p}(\mu, \lambda) &= \log p(\mu, \lambda, \mathcal{D}) = \log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda) \quad (21.67)\\ &= \frac{N}{2}\log\lambda - \frac{\lambda}{2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{\kappa_0\lambda}{2}(\mu-\mu_0)^2 \\ &\quad + \frac{1}{2}\log(\kappa_0\lambda) + (a_0-1)\log\lambda - b_0\lambda + \text{const} \quad (21.68)\end{aligned}$$

21.5.1.2 Updating $q_\mu(\mu)$

The optimal form for $q_\mu(\mu)$ is obtained by averaging over λ:

$$\begin{aligned}\log q_\mu(\mu) &= \mathbb{E}_{q_\lambda}\left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda)\right] + \text{const} \quad (21.69)\\ &= -\frac{\mathbb{E}_{q_\lambda}[\lambda]}{2}\left\{\kappa_0(\mu-\mu_0)^2 + \sum_{i=1}^{N}(x_i-\mu)^2\right\} + \text{const} \quad (21.70)\end{aligned}$$

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

$$\mu_N = \frac{\kappa_0\mu_0 + N\overline{x}}{\kappa_0 + N}, \ \kappa_N = (\kappa_0 + N)\mathbb{E}_{q_\lambda}[\lambda] \quad (21.71)$$

At this stage we don't know what $q_\lambda(\lambda)$ is, and hence we cannot compute $\mathbb{E}[\lambda]$, but we will derive this below.

21.5.1.3 Updating $q_\lambda(\lambda)$

The optimal form for $q_\lambda(\lambda)$ is given by

$$\begin{aligned}\log q_\lambda(\lambda) &= \mathbb{E}_{q_\mu}\left[\log p(\mathcal{D}|\mu, \lambda) + \log p(\mu|\lambda) + \log p(\lambda)\right] + \text{const} \quad (21.72)\\ &= (a_0-1)\log\lambda - b_0\lambda + \frac{1}{2}\log\lambda + \frac{N}{2}\log\lambda \\ &\quad -\frac{\lambda}{2}\mathbb{E}_{q_\mu}\left[\kappa_0(\mu-\mu_0)^2 + \sum_{i=1}^{N}(x_i-\mu)^2\right] + \text{const} \quad (21.73)\end{aligned}$$

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

$$\begin{aligned}a_N &= a_0 + \frac{N+1}{2} \quad (21.74)\\ b_N &= b_0 + \frac{1}{2}\mathbb{E}_{q_\mu}\left[\kappa_0(\mu-\mu_0)^2 + \sum_{i=1}^{N}(x_i-\mu)^2\right] \quad (21.75)\end{aligned}$$

21.5.1.4 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

$$\begin{aligned} \mathbb{E}_{q(\mu)}\left[\mu\right] &= \mu_N && (21.76) \\ \mathbb{E}_{q(\mu)}\left[\mu^2\right] &= \frac{1}{\kappa_N} + \mu_N^2 && (21.77) \end{aligned}$$

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

$$\mathbb{E}_{q(\lambda)}\left[\lambda\right] = \frac{a_N}{b_N} \tag{21.78}$$

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

$$\begin{aligned} \mu_N &= \frac{\kappa_0\mu_0 + N\overline{x}}{\kappa_0 + N} && (21.79) \\ \kappa_N &= (\kappa_0 + N)\frac{a_N}{b_N} && (21.80) \end{aligned}$$

and for $q(\lambda)$ we have

$$\begin{aligned} a_N &= a_0 + \frac{N+1}{2} && (21.81) \\ b_N &= b_0 + \kappa_0(\mathbb{E}\left[\mu^2\right] + \mu_0^2 - 2\mathbb{E}\left[\mu\right]\mu_0) + \frac{1}{2}\sum_{i=1}^{N}\left(x_i^2 + \mathbb{E}\left[\mu^2\right] - 2\mathbb{E}\left[\mu\right]x_i\right) && (21.82) \end{aligned}$$

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.)

21.5.1.5 Illustration

Figure 21.5 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; See Section 21.2.2 for more discussion of this point.

21.5.1.6 Lower bound *

In VB, we are maximizing $L(q)$, which is a lower bound on the log marginal likelihood:

$$L(q) \leq \log p(\mathcal{D}) = \log \int\int p(\mathcal{D}|\mu, \lambda)p(\mu, \lambda)d\mu d\lambda \tag{21.83}$$

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

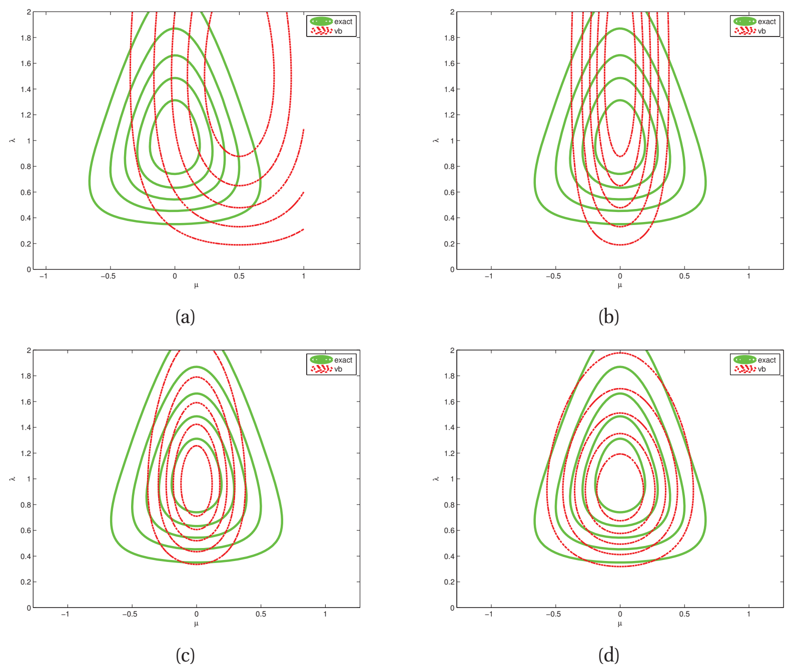


Figure 21.5 Factored variational approximation (red) to the Gaussian-Gamma distribution (green). (a) Initial guess. (b) After updating q_μ. (c) After updating q_λ. (d) At convergence (after 5 iterations). Based on 10.4 of (Bishop 2006b). Figure generated by `unigaussVbDemo`.

code: as with EM, if the bound does not increase monotonically, 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.

Unfortunately, computing this lower bound involves a fair amount of tedious algebra. We work out the details for this example, but for other models, we will just state the results without proof, or even omit discussion of the bound altogether, for brevity.

For this model, $L(q)$ can be computed as follows:

$$\begin{aligned} L(q) &= \int\int q(\mu,\lambda)\log\frac{p(\mathcal{D},\mu,\lambda)}{q(\mu,\lambda)}d\mu d\lambda && (21.84)\\ &= \mathbb{E}\left[\log p(\mathcal{D}|\mu,\lambda)\right] + \mathbb{E}\left[\log p(\mu|\lambda)\right] + \mathbb{E}\left[\log p(\lambda)\right] \\ &\quad -\mathbb{E}\left[\log q(\mu)\right] - \mathbb{E}\left[\log q(\lambda)\right] && (21.85) \end{aligned}$$

where all expectations are wrt $q(\mu, \lambda)$. We recognize the last two terms as the entropy of a Gaussian and the entropy of a Gamma distribution, which are given by

$$\begin{aligned} \mathbb{H}\left(\mathcal{N}(\mu_N, \kappa_N^{-1})\right) &= -\frac{1}{2}\log\kappa_N + \frac{1}{2}(1+\log(2\pi)) && (21.86)\\ \mathbb{H}\left(\mathrm{Ga}(a_N, b_N)\right) &= \log\Gamma(a_N) - (a_N-1)\psi(a_N) - \log(b_N) + a_N && (21.87) \end{aligned}$$

where $\psi()$ is the digamma function.

To compute the other terms, we need the following facts:

$$\begin{aligned}
\mathbb{E}\left[\log x|x \sim \text{Ga}(a,b)\right] &= \psi(a) - \log(b) && (21.88)\\
\mathbb{E}\left[x|x \sim \text{Ga}(a,b)\right] &= \frac{a}{b} && (21.89)\\
\mathbb{E}\left[x|x \sim \mathcal{N}(\mu,\sigma^2)\right] &= \mu && (21.90)\\
\mathbb{E}\left[x^2|x \sim \mathcal{N}(\mu,\sigma^2)\right] &= \mu + \sigma^2 && (21.91)
\end{aligned}$$

For the expected log likelihood, one can show that

$$\begin{aligned}
&\mathbb{E}_{q(\mu,\lambda)}\left[\log p(\mathcal{D}|\mu,\lambda)\right] && (21.92)\\
&= -\frac{N}{2}\log(2\pi) + \frac{N}{2}\mathbb{E}_{q(\lambda)}\left[\log\lambda\right] - \frac{\mathbb{E}\left[\lambda\right]_{q(\lambda)}}{2}\sum_{i=1}^{N}\mathbb{E}_{q(\mu)}\left[(x_i-\mu)^2\right]\\
&= -\frac{N}{2}\log(2\pi) + \frac{N}{2}\left(\psi(a_N) - \log b_N\right) && (21.93)\\
&\quad -\frac{Na_N}{2b_N}\left(\hat{\sigma}^2 + \overline{x}^2 - 2\mu_N\overline{x} + \mu_N^2 + \frac{1}{\kappa_N}\right) && (21.94)
\end{aligned}$$

where $\overline{x}$ and $\hat{\sigma}^2$ are the empirical mean and variance.

For the expected log prior of λ, we have

$$\begin{aligned}
\mathbb{E}_{q(\lambda)}\left[\log p(\lambda)\right] &= (a_0-1)\mathbb{E}\left[\log\lambda\right] - b_0\mathbb{E}\left[\lambda\right] + a_0\log b_0 - \log\Gamma(a_0) && (21.95)\\
&= (a_0-1)(\psi(a_N) - \log b_N) - b_0\frac{a_N}{b_N} + a_0\log b_0 - \log\Gamma(a_0) && (21.96)
\end{aligned}$$

For the expected log prior of μ, one can show that

$$\begin{aligned}
\mathbb{E}_{q(\mu,\lambda)}\left[\log p(\mu|\lambda)\right] &= \frac{1}{2}\log\frac{\kappa_0}{2\pi} + \frac{1}{2}\mathbb{E}\left[\log\lambda\right]q(\lambda) - \frac{1}{2}\mathbb{E}_{q(\mu,\lambda)}\left[(\mu-\mu_0)^2\kappa_0\lambda\right]\\
&= \frac{1}{2}\log\frac{\kappa_0}{2\pi} + \frac{1}{2}\left(\psi(a_N) - \log b_N\right)\\
&\quad -\frac{\kappa_0}{2}\frac{a_N}{b_N}\left[\frac{1}{\kappa_N} + (\mu_N-\mu_0)^2\right] && (21.97)
\end{aligned}$$

Putting it altogether, one can show that

$$L(q) = \frac{1}{2}\log\frac{1}{\kappa_N} + \log\Gamma(a_N) - a_N\log b_N + \text{const} \qquad (21.98)$$

This quantity monotonically increases after each VB update.

21.5.2 Example: VB for linear regression

In Section 7.6.4, we discussed an empirical Bayes approach to setting the hyper-parameters for ridge regression known as the evidence procedure. In particular, we assumed a likelihood of the form $p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{X}\mathbf{w},\lambda^{-1})$ and a prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0},\alpha^{-1}\mathbf{I})$. We then computed a type II estimate of α and λ. The same approach was extended in Section 13.7 to handle a prior of the form $\mathcal{N}(\mathbf{w}|\mathbf{0}, \text{diag}(\boldsymbol{\alpha})^{-1})$, which allows one hyper-parameter per feature, a technique known as automatic relevancy determination.

In this section, we derive a VB algorithm for this model. We follow the presentation of (Drugowitsch 2008).[3] Initially we will use the following prior:

$$p(\mathbf{w}, \lambda, \alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, (\lambda\alpha)^{-1}\mathbf{I})\text{Ga}(\lambda|a_0^\lambda, b_0^\lambda)\text{Ga}(\alpha|a_0^\alpha, b_0^\alpha) \tag{21.99}$$

We choose to use the following factorized approximation to the posterior:

$$q(\mathbf{w}, \alpha, \lambda) = q(\mathbf{w}, \lambda)q(\alpha) \tag{21.100}$$

Given these assumptions, one can show (see (Drugowitsch 2008)) that the optimal form for the posterior is

$$q(\mathbf{w}, \alpha, \lambda) = \mathcal{N}(\mathbf{w}|\mathbf{w}_N, \lambda^{-1}\mathbf{V}_N)\text{Ga}(\lambda|a_N^\lambda, b_N^\lambda)\text{Ga}(\alpha|a_N^\alpha, b_N^\alpha) \tag{21.101}$$

where

$$\mathbf{V}_N^{-1} = \overline{\mathbf{A}} + \mathbf{X}^{\mathbf{X}} \tag{21.102}$$

$$\mathbf{w}_N = \mathbf{V}_N\mathbf{X}^T\mathbf{y} \tag{21.103}$$

$$a_N^\lambda = a_0^\lambda + \frac{N}{2} \tag{21.104}$$

$$b_N^\lambda = b_0^\lambda + \frac{1}{2}(||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 + \mathbf{w}_N^T\overline{\mathbf{A}}\mathbf{w}_N) \tag{21.105}$$

$$a_N^\alpha = a_0^\alpha + \frac{D}{2} \tag{21.106}$$

$$b_N^\alpha = b_0^\alpha + \frac{1}{2}\left(\frac{a_N^\lambda}{b_N^\lambda}\mathbf{w}_N^T\mathbf{w}_N + \text{tr}(\mathbf{V}_N)\right) \tag{21.107}$$

$$\overline{\mathbf{A}} = \langle\alpha\rangle\mathbf{I} = \frac{a_N^\alpha}{b_N^\alpha}\mathbf{I} \tag{21.108}$$

This method can be extended to the ARD case in a straightforward way, by using the following priors:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \text{diag}(\alpha)^{-1}) \tag{21.109}$$

$$p(\boldsymbol{\alpha}) = \prod_{j=1}^{D}\text{Ga}(\alpha_j|a_0^\alpha, b_0^\alpha) \tag{21.110}$$

The posterior for $\mathbf{w}$ and λ is computed as before, except we use $\overline{\mathbf{A}} = \text{diag}(a_N^\alpha/b_{N_j}^\alpha)$ instead of

3. Note that Drugowitsch uses a_0, b_0 as the hyper-parameters for $p(\lambda)$ and c_0, d_0 as the hyper-parameters for $p(\alpha)$, whereas (Bishop 2006b, Sec 10.3) uses a_0, b_0 as the hyper-parameters for $p(\alpha)$ and treats λ as fixed. To (hopefully) avoid confusion, I use a_0^λ, b_0^λ as the hyper-parameters for $p(\lambda)$, and a_0^α, b_0^α as the hyper-parameters for $p(\alpha)$.

$a_N^\alpha/b_N^\alpha \mathbf{I}$. The posterior for $\boldsymbol{\alpha}$ has the form

$$q(\boldsymbol{\alpha}) = \prod_j \text{Ga}(\alpha_j|a_N^\alpha, b_{N_j}^\alpha) \tag{21.111}$$

$$a_N^\alpha = a_0^\alpha + \frac{1}{2} \tag{21.112}$$

$$b_{N_j}^\alpha = b_0^\alpha + \frac{1}{2}\left(\frac{a_N^\lambda}{b_N^\lambda} w_{N,j}^2 + (\mathbf{V}_N)_{jj}\right) \tag{21.113}$$

The algorithm alternates between updating $q(\mathbf{w}, \lambda)$ and $q(\boldsymbol{\alpha})$. Once $\mathbf{w}$ and λ have been inferred, the posterior predictive is a Student distribution, as shown in Equation 7.76. Specifically, for a single data case, we have

$$p(y|\mathbf{x}, \mathcal{D}) = \mathcal{T}(y|\mathbf{w}_N^T\mathbf{x}, \frac{b_N^\lambda}{a_N^\lambda}(1 + \mathbf{x}^T\mathbf{V}_N\mathbf{x}), 2a_N^\lambda) \tag{21.114}$$

The exact marginal likelihood, which can be used for model selection, is given by

$$p(\mathcal{D}) = \int\int\int p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \lambda)p(\mathbf{w}|\alpha)p(\lambda)d\mathbf{w}d\alpha d\lambda \tag{21.115}$$

We can compute a lower bound on $\log p(\mathcal{D})$ as follows:

$$\begin{aligned} L(q) = & -\frac{N}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^N\left(\frac{a_N^\lambda}{b_N^\lambda}(y_i - \mathbf{w}_N^T\mathbf{x}_i)^2 + \mathbf{x}_i^T\mathbf{V}_N\mathbf{x}_i\right) \\ & +\frac{1}{2}\log|\mathbf{V}_N| + \frac{D}{2} \\ & -\log\Gamma(a_0^\lambda) + a_0^\lambda\log b_0^\lambda - b_0^\lambda\frac{a_N^\lambda}{b_N^\lambda} + \log\Gamma(a_N^\lambda) - a_N^\lambda\log b_N^\lambda + a_N^\lambda \\ & -\log\Gamma(a_0^\alpha) + a_0^\alpha\log b_0^\alpha + \log\Gamma(a_N^\alpha) - a_N^\alpha\log b_N^\alpha \end{aligned} \tag{21.116}$$

In the ARD case, the last line becomes

$$\sum_{j=1}^D\left[-\log\Gamma(a_0^\alpha) + a_0^\alpha\log b_0^\alpha + \log\Gamma(a_N^\alpha) - a_N^\alpha\log b_{N_j}^\alpha\right] \tag{21.117}$$

Figure 21.6 compare VB and EB on a model selection problem for polynomial regression. We see that VB gives similar results to EB, but the precise behavior depends on the sample size. When $N = 5$, VB's estimate of the posterior over models is more diffuse than EB's, since VB models uncertainty in the hyper-parameters. When $N = 30$, the posterior estimate of the hyper-parameters becomes more well-determined. Indeed, if we compute $\mathbb{E}[\alpha|\mathcal{D}]$ when we have an uninformative prior, $a_0^\alpha = b_0^\alpha = 0$, we get

$$\overline{\alpha} = \frac{a_N^\alpha}{b_N^\alpha} = \frac{D/2}{\frac{1}{2}(\frac{a_N^\lambda}{b_N^\lambda}\mathbf{w}_N^T\mathbf{w}_N + \text{tr}(\mathbf{V}_N))} \tag{21.118}$$

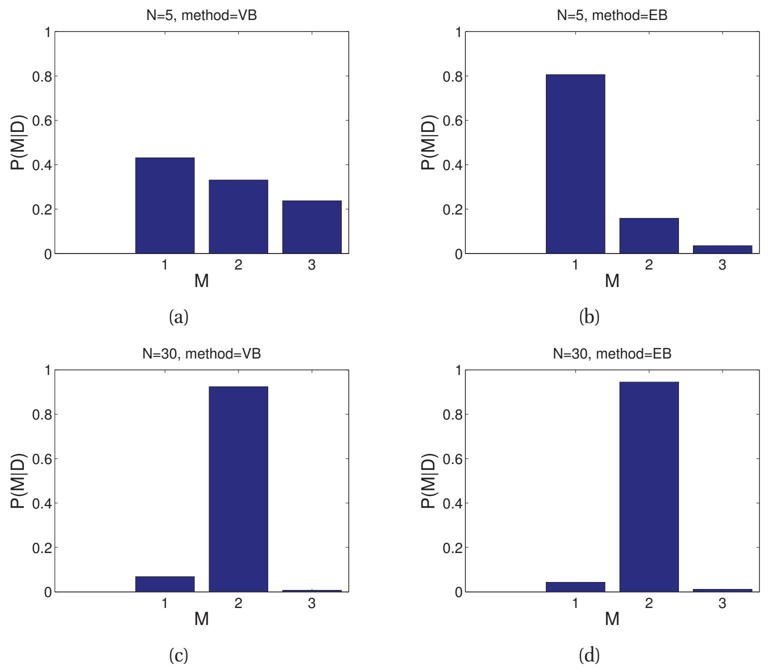


Figure 21.6 We plot the posterior over models (polynomials of degree 1, 2 and 3) assuming a uniform prior $p(m) \propto 1$. We approximate the marginal likelihood using (a,c) VB and (b,d) EB. In (a-b), we use $N = 5$ data points (shown in Figure 5.7). In (c-d), we use $N = 30$ data points (shown in Figure 5.8). Figure generated by `linregEbModelSelVsN`.

Compare this to Equation 13.167 for EB:

$$\hat{\alpha} = \frac{D}{\mathbb{E}\left[\mathbf{w}^T\mathbf{w}\right]} = \frac{D}{\mathbf{w}_N^T\mathbf{w}_N + \text{tr}(\mathbf{V}_N)} \tag{21.119}$$

Modulo the a_N^λ and b_N^λ terms, these are the same. In hindsight this is perhaps not that surprising, since EB is trying to maximize $\log p(\mathcal{D})$, and VB is trying to maximize a lower bound on $\log p(\mathcal{D})$.

21.6 Variational Bayes EM

Now consider latent variable models of the form $\mathbf{z}_i \rightarrow \mathbf{x}_i \leftarrow \boldsymbol{\theta}$. This includes mixtures models, PCA, HMMs, etc. There are now two kinds of unknowns: parameters, $\boldsymbol{\theta}$, and latent variables, $\mathbf{z}_i$. As we saw in Section 11.4, it is common to fit such models using EM, where in the E step we infer the posterior over the latent variables, $p(\mathbf{z}_i|\mathbf{x}_i, \boldsymbol{\theta})$, and in the M step, we compute a point estimate of the parameters, $\boldsymbol{\theta}$. The justification for this is two-fold. First, it results in simple algorithms. Second, the posterior uncertainty in $\boldsymbol{\theta}$ is usually less than in $\mathbf{z}_i$, since the $\boldsymbol{\theta}$ are informed by all N data cases, whereas $\mathbf{z}_i$ is only informed by $\mathbf{x}_i$; this makes a MAP estimate of

$\boldsymbol{\theta}$ more reasonable than a MAP estimate of $\mathbf{z}_i$.

However, VB provides a way to be "more Bayesian", by modeling uncertainty in the parameters $\boldsymbol{\theta}$ as well in the latent variables $\mathbf{z}_i$, at a computational cost that is essentially the same as EM. This method is known as **variational Bayes EM** or **VBEM**. The basic idea is to use mean field, where the approximate posterior has the form

$$p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})q(\mathbf{z}) = q(\boldsymbol{\theta}) \prod_i q(\mathbf{z}_i) \tag{21.120}$$

The first factorization, between $\boldsymbol{\theta}$ and $\mathbf{z}$, is a crucial assumption to make the algorithm tractable. The second factorization follows from the model, since the latent variables are iid conditional on $\boldsymbol{\theta}$.

In VBEM, we alternate between updating $q(\mathbf{z}_i|\mathcal{D})$ (the variational E step) and updating $q(\boldsymbol{\theta}|\mathcal{D})$ (the variational M step). We can recover standard EM from VBEM by approximating the parameter posterior using a delta function, $q(\boldsymbol{\theta}|\mathcal{D}) \approx \delta_{\hat{\boldsymbol{\theta}}}(\boldsymbol{\theta})$.

The variational E step is similar to a standard E step, except instead of plugging in a MAP estimate of the parameters and computing $p(\mathbf{z}_i|\mathcal{D}, \hat{\boldsymbol{\theta}})$, we need to average over the parameters. Roughly speaking, this can be computed by plugging in the posterior mean of the parameters instead of the MAP estimate, and then computing $p(\mathbf{z}_i|\mathcal{D}, \overline{\boldsymbol{\theta}})$ using standard algorithms, such as forwards-backwards. Unfortunately, things are not quite this simple, but this is the basic idea. The details depend on the form of the model; we give some examples below.

The variational M step is similar to a standard M step, except instead of computing a point estimate of the parameters, we update the hyper-parameters, using the expected sufficient statistics. This process is usually very similar to MAP estimation in regular EM. Again, the details on how to do this depend on the form of the model.

The principle advantage of VBEM over regular EM is that by marginalizing out the parameters, we can compute a lower bound on the marginal likelihood, which can be used for model selection. We will see an example of this in Section 21.6.1.6. VBEM is also "egalitarian", since it treats parameters as "first class citizens", just like any other unknown quantity, whereas EM makes an artificial distinction between parameters and latent variables.

21.6.1 Example: VBEM for mixtures of Gaussians *

Let us consider how to "fit" a mixture of Gaussians using VBEM. (We use scare quotes since we are not estimating the model parameters, but inferring a posterior over them.) We will follow the presentation of (Bishop 2006b, Sec 10.2). Unfortunately, the details are rather complicated. Fortunately, as with EM, one gets used to it after a bit of practice. (As usual with math, simply reading the equations won't help much, you should really try deriving these results yourself (or try some of the exercises) if you want to learn this stuff in depth.)

21.6.1.1 The variational posterior

The likelihood function is the usual one for Gaussian mixture models:

$$p(\mathbf{z}, \mathbf{X}|\boldsymbol{\theta}) = \prod_i \prod_k \pi_k^{z_{ik}} \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{ik}} \tag{21.121}$$

where $z_{ik} = 1$ if data point i belongs to cluster k, and $z_{ik} = 0$ otherwise.

We will assume the following factored conjugate prior

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}_0)\prod_k \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_0, (\beta_0\boldsymbol{\Lambda}_k)^{-1})\text{Wi}(\boldsymbol{\Lambda}_k|\mathbf{L}_0, \nu_0) \tag{21.122}$$

where $\boldsymbol{\Lambda}_k$ is the precision matrix for cluster k. The subscript 0 means these are parameters of the prior; we assume all the prior parameters are the same for all clusters. For the mixing weights, we usually use a symmetric prior, $\boldsymbol{\alpha}_0 = \alpha_0\mathbf{1}$.

The exact posterior $p(\mathbf{z}, \boldsymbol{\theta}|\mathcal{D})$ is a mixture of K^N distributions, corresponding to all possible labelings $\mathbf{z}$. We will try to approximate the volume around one of these modes. We will use the standard VB approximation to the posterior:

$$p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})\prod_i q(\mathbf{z}_i) \tag{21.123}$$

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 form is as follows:

$$\begin{aligned} q(\mathbf{z}, \boldsymbol{\theta}) &= q(\mathbf{z}|\boldsymbol{\theta})q(\boldsymbol{\theta}) = \left[\prod_i \text{Cat}(\mathbf{z}_i|\mathbf{r}_i)\right] && (21.124)\\ & \left[\text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha})\prod_k \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_k, (\beta_k\boldsymbol{\Lambda}_k)^{-1})\text{Wi}(\boldsymbol{\Lambda}_k|\mathbf{L}_k, \nu_k)\right] && (21.125)\end{aligned}$$

(The lack of 0 subscript means these are parameters of the posterior, not the prior.) Below we will derive the update equations for these variational parameters.

21.6.1.2 Derivation of $q(\mathbf{z})$ (variational E step)

The form for $q(\mathbf{z})$ can be obtained by looking at the complete data log joint, ignoring terms that do not involve $\mathbf{z}$, and taking expectations of what's left over wrt all the hidden variables except for $\mathbf{z}$. We have

$$\begin{aligned} \log q(\mathbf{z}) &= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta})\right] + \text{const} && (21.126)\\ &= \sum_i \sum_i z_{ik}\log\rho_{ik} + \text{const} && (21.127)\end{aligned}$$

where we define

$$\begin{aligned} \log\rho_{ik} \triangleq\ & \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \pi_k\right] + \frac{1}{2}\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log|\boldsymbol{\Lambda}_k|\right] - \frac{D}{2}\log(2\pi)\\ & -\frac{1}{2}\mathbb{E}_{q(\boldsymbol{\theta})}\left[(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\boldsymbol{\Lambda}_k(\mathbf{x}_i - \boldsymbol{\mu}_k)\right] && (21.128)\end{aligned}$$

Using the fact that $q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi})$, we have

$$\log\tilde{\pi}_k \triangleq \mathbb{E}\left[\log\pi_k\right] = \psi(\alpha_k) - \psi(\sum_{k'}\alpha_{k'}) \tag{21.129}$$

where $\psi()$ is the digamma function. (See Exercise 21.5 for the detailed derivation.) Next, we use the fact that

$$q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_k, (\beta_k \boldsymbol{\Lambda}_k)^{-1}) \mathrm{Wi}(\boldsymbol{\Lambda}_k|\mathbf{L}_k, \nu_k) \tag{21.130}$$

to get

$$\log \tilde{\Lambda}_k \triangleq \mathbb{E}\left[\log |\boldsymbol{\Lambda}_k|\right] = \sum_{j=1}^{D} \psi\left(\frac{\nu_k + 1 - j}{2}\right) + D \log 2 + \log |\boldsymbol{\Lambda}_k| \tag{21.131}$$

Finally, for the expected value of the quadratic form, we get

$$\mathbb{E}\left[(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Lambda}_k (\mathbf{x}_i - \boldsymbol{\mu}_k)\right] = D\beta_k^{-1} + \nu_k (\mathbf{x}_i - \mathbf{m}_k)^T \boldsymbol{\Lambda}_k (\mathbf{x}_i - \mathbf{m}_k) \tag{21.132}$$

Putting it altogether, we get that the posterior responsibility of cluster k for datapoint i is

$$r_{ik} \quad \propto \quad \tilde{\pi}_k \tilde{\Lambda}_k^{\frac{1}{2}} \exp\left(-\frac{D}{2\beta_k} - \frac{\nu_k}{2}(\mathbf{x}_i - \mathbf{m}_k)^T \boldsymbol{\Lambda}_k (\mathbf{x}_i - \mathbf{m}_k)\right) \tag{21.133}$$

Compare this to the expression used in regular EM:

$$r_{ik}^{EM} \quad \propto \quad \hat{\pi}_k |\hat{\boldsymbol{\Lambda}}|_k^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^T \hat{\boldsymbol{\Lambda}}_k (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)\right) \tag{21.134}$$

The significance of this difference is discussed further in Section 21.6.1.7.

21.6.1.3 Derivation of $q(\theta)$ (variational M step)

Using the mean field recipe, we have

$$\begin{aligned} \log q(\boldsymbol{\theta}) \quad = \quad & \log p(\boldsymbol{\pi}) + \sum_k \log p(\mu_k, \boldsymbol{\Lambda}_k) + \sum_i \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{z}_i|\boldsymbol{\pi})\right] \\ & + \sum_k \sum_i \mathbb{E}_{q(\mathbf{z})}\left[z_{ik}\right] \log \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) + \mathrm{const} \end{aligned} \tag{21.135}$$

We see this factorizes into the form

$$q(\boldsymbol{\theta}) \quad = \quad q(\boldsymbol{\pi}) \prod_k q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \tag{21.136}$$

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

$$\log q(\boldsymbol{\pi}) \quad = \quad (\alpha_0 - 1) \sum_k \log \pi_k + \sum_k \sum_i r_{ik} \log \pi_k + \mathrm{const} \tag{21.137}$$

Exponentiating, we recognize this as a Dirichlet distribution:

$$q(\boldsymbol{\pi}) \quad = \quad \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \tag{21.138}$$

$$\alpha_k \quad = \quad \alpha_0 + N_k \tag{21.139}$$

$$N_k \quad = \quad \sum_i r_{ik} \tag{21.140}$$

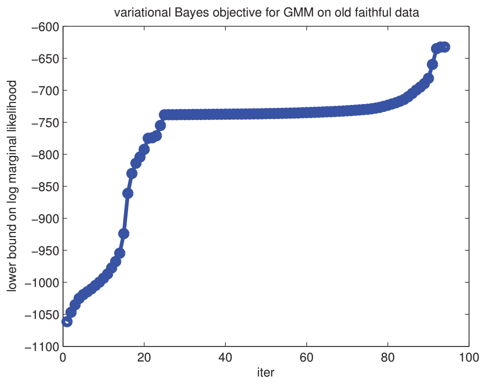


Figure 21.7 Lower bound vs iterations for the VB algorithm in Figure 21.8. The steep parts of the curve correspond to places where the algorithm figures out that it can increase the bound by "killing off" unnecessary mixture components, as described in Section 21.6.1.6. The plateaus correspond to slowly moving the clusters around. Figure generated by `mixGaussVbDemoFaithful`.

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

$$\begin{aligned}
q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) &= \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_k, (\beta_k \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k | \mathbf{L}_k, \nu_k) & (21.141)\\
\beta_k &= \beta_0 + N_k & (21.142)\\
\mathbf{m}_k &= (\beta_0 \mathbf{m}_0 + N_k \overline{\mathbf{x}}_k)/\beta_k & (21.143)\\
\mathbf{L}_k^{-1} &= \mathbf{L}_0^{-1} + N_k \mathbf{S}_k + \frac{\beta_0 N_k}{\beta_0 + N_k} (\overline{\mathbf{x}}_k - \mathbf{m}_0)(\overline{\mathbf{x}}_k - \mathbf{m}_0)^T & (21.144)\\
\nu_k &= \nu_0 + N_k + 1 & (21.145)\\
\overline{\mathbf{x}}_k &= \frac{1}{N_k} \sum_i r_{ik} \mathbf{x}_i & (21.146)\\
\mathbf{S}_k &= \frac{1}{N_k} \sum_i r_{ik} (\mathbf{x}_i - \overline{\mathbf{x}}_k)(\mathbf{x}_i - \overline{\mathbf{x}}_k)^T & (21.147)
\end{aligned}$$

This is very similar to the M step for MAP estimation discussed in Section 11.4.2.8, except here we are computing the parameters of the posterior over $\boldsymbol{\theta}$, rather than MAP estimates of $\boldsymbol{\theta}$.

21.6.1.4 Lower bound on the marginal likelihood

The algorithm is trying to maximize the following lower bound

$$\mathcal{L} = \sum_{\mathbf{z}} \int q(\mathbf{z}, \boldsymbol{\theta}) \log \frac{p(\mathbf{x}, \mathbf{z}, \boldsymbol{\theta})}{q(\mathbf{z}, \boldsymbol{\theta})} d\boldsymbol{\theta} \leq \log p(\mathcal{D}) \tag{21.148}$$

This quantity should increase monotonically with each iteration, as shown in Figure 21.7. Unfortunately, deriving the bound is a bit messy, because we need to compute expectations of the unnormalized log posterior as well as entropies of the q distribution. We leave the details (which are similar to Section 21.5.1.6) to Exercise 21.4.

21.6.1.5 Posterior predictive distribution

We showed that the approximate posterior has the form

$$q(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_k \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_k, (\beta_k \boldsymbol{\Lambda}_k)^{-1}) \text{Wi}(\boldsymbol{\Lambda}_k|\mathbf{L}_k, \nu_k) \tag{21.149}$$

Consequently the posterior predictive density can be approximated as follows, using the results from Section 4.6.3.6:

$$\begin{aligned}
p(\mathbf{x}|\mathcal{D}) &\approx \sum_z \int p(\mathbf{x}|z, \boldsymbol{\theta}) p(z|\boldsymbol{\theta}) q(\boldsymbol{\theta}) d\boldsymbol{\theta} && (21.150)\\
&= \sum_k \int \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) q(\boldsymbol{\theta}) d\boldsymbol{\theta} && (21.151)\\
&= \sum_k \frac{\alpha_k}{\sum_{k'} \alpha_{k'}} \mathcal{T}(\mathbf{x}|\mathbf{m}_k, \mathbf{M}_k, \nu_k + 1 - D) && (21.152)\\
\mathbf{M}_k &= \frac{(\nu_k + 1 - D)\beta_k}{1 + \beta_k} \mathbf{L}_k && (21.153)
\end{aligned}$$

This is just a weighted sum of Student distributions. If instead we used a plug-in approximation, we would get a weighted sum of Gaussian distributions.

21.6.1.6 Model selection using VBEM

The simplest way to select K when using VB 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 $p(K|\mathcal{D})$:

$$p(K|\mathcal{D}) = \frac{e^{\mathcal{L}(K)}}{\sum_{K'} e^{\mathcal{L}(K')}} \tag{21.154}$$

However, the lower bound needs to be modified somewhat to take into account the lack of identifiability of the parameters (Section 11.3.1). 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 we should use $\log p(\mathcal{D}|K) \approx \mathcal{L}(K) + \log(K!)$.

21.6.1.7 Automatic sparsity inducing effects of VBEM

Although VB provides a reasonable approximation to the marginal likelihood (better than BIC (Beal and Ghahramani 2006)), this method still requires fitting multiple models, one for each value of K being considered. A faster alternative is to fit a single model, where K is set large, but where α_0 is set very small, $\alpha_0 \ll 1$. From Figure 2.14(d), we see that the resulting prior for the mixing weights $\boldsymbol{\pi}$ has "spikes" near the corners of the simplex, encouraging a sparse mixing weight vector.

In regular EM, the MAP estimate of the mixing weights will have the form $\hat{\pi}_k \propto (\alpha_k - 1)$, where $\alpha_k = \alpha_0 + N_k$. Unforuntately, this can be negative if $\alpha_0 = 0$ and $N_k = 0$ (Figueiredo

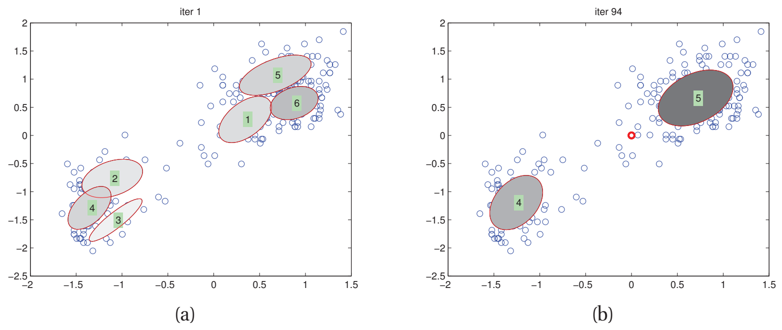


Figure 21.8 We visualize the posterior mean parameters at various stages of the VBEM algorithm applied to a mixture of Gaussians model on the Old Faithful data. Shading intensity is proportional to the mixing weight. We initialize with K-means and use $\alpha_0 = 0.001$ as the Dirichlet hyper-parameter. Based on Figure 10.6 of (Bishop 2006b). Figure generated by `mixGaussVbDemoFaithful`, based on code by Emtiyaz Khan.

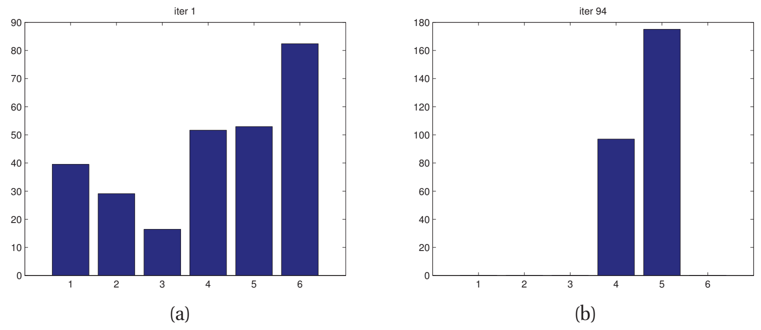


Figure 21.9 We visualize the posterior values of $\boldsymbol{\alpha}_k$ for the model in Figure 21.8. We see that unnecessary components get "killed off". Figure generated by `mixGaussVbDemoFaithful`.

and Jain 2002). However, in VBEM, we use

$$\tilde{\pi}_k = \frac{\exp[\Psi(\alpha_k)]}{\exp[\Psi(\sum_{k'} \alpha_{k'})]} \tag{21.155}$$

Now $\exp(\Psi(x)) \approx x - 0.5$ for $x > 1$. So if $\alpha_k = 0$, when we compute $\tilde{\pi}_k$, it's like we substract 0.5 from the posterior counts. This will hurt small clusters more than large clusters (like a regressive tax).[4] The effect is that clusters which have very few (weighted) members become more and more empty over successive iterations, whereas the popular clusters get more and more members. This is called the **rich get richer** phenomenon; we will encounter it again in Section 25.2, when we discuss Dirichlet process mixture models.

This automatic pruning method is demonstrated in Figure 21.8. We fit a mixture of 6 Gaussians to the Old Faithful dataset, but the data only really "needs" 2 clusters, so the rest get "killed off".

4. For more details, see (Liang et al. 2007).

In this example, we used $\alpha_0 = 0.001$; if we use a larger α_0, we do not get a sparsity effect. In Figure 21.9, we plot $q(\boldsymbol{\alpha}|\mathcal{D})$ at various iterations; we see that the unwanted components get extinguished. This provides an efficient alternative to performing a discrete search over the number of clusters.

21.7 Variational message passing and VIBES

We have seen that mean field methods, at least of the fully-factorized variety, are all very similar: just compute each node's full conditional, and average out the neighbors. This is very similar to Gibbs sampling (Section 24.2), except the derivation of the equations is usually a bit more work. Fortunately it is possible to derive a general purpose set of update equations that work for any DGM for which all CPDs are in the exponential family, and for which all parent nodes have conjugate distributions (Ghahramani and Beal 2001). (See (Wand et al. 2011) for a recent extension to handle non-conjugate priors.) One can then sweep over the graph, updating nodes one at a time, in a manner similar to Gibbs sampling. This is known as **variational message passing** or **VMP** (Winn and Bishop 2005), and has been implemented in the open-source program **VIBES**[5]. This is a VB analog to BUGS, which is a popular generic program for Gibbs sampling discussed in Section 24.2.6.

VMP/ mean field is best-suited to inference where one or more of the hidden nodes are continuous (e.g., when performing "Bayesian learning"). For models where all the hidden nodes are discrete, more accurate approximate inference algorithms can be used, as we discuss in Chapter 22.

21.8 Local variational bounds *

So far, we have been focusing on mean field inference, which is a form of variational inference based on minimizing $\mathbb{KL}(q||\tilde{p})$, where q is the approximate posterior, assumed to be factorized, and $\tilde{p}$ is the exact (but unnormalized) posterior. However, there is another kind of variational inference, where we replace a specific term in the joint distribution with a simpler function, to simplify computation of the posterior. Such an approach is sometimes called a **local variational approximation**, since we are only modifying one piece of the model, unlike mean field, which is a global approximation. In this section, we study several examples of this method.

21.8.1 Motivating applications

Before we explain how to derive local variational bounds, we give some examples of where this is useful.

21.8.1.1 Variational logistic regression

Consider the problem of how to approximate the parameter posterior for multiclass logistic regression model under a Gaussian prior. One approach is to use a Gaussian (Laplace) approximation, as discussed in Section 8.4.3. However, a variational approach can produce a more

5. Available at `http://vibes.sourceforge.net/`.

accurate approximation to the posterior, since it has tunable parameters. Another advantage is that the variational approach monotonically optimizes a lower bound on the likelihood of the data, as we will see.

To see why we need a bound, note that the likelihood can be written as follows:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{N} \exp\left[\mathbf{y}_i^T \boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i)\right] \tag{21.156}$$

where $\boldsymbol{\eta}_i = \mathbf{X}_i \mathbf{w}_i = [\mathbf{x}_i^T \mathbf{w}_1, \ldots, \mathbf{x}_i^T \mathbf{w}_M]$, where $M = C - 1$ (since we set $\mathbf{w}_C = \mathbf{0}$ for identifiability), and where we define the **log-sum-exp** or **lse** function as follows:

$$\text{lse}(\boldsymbol{\eta}_i) \triangleq \log\left(1 + \sum_{m=1}^{M} e^{\eta_{im}}\right) \tag{21.157}$$

The main problem is that this likelihood is not conjugate to the Gaussian prior. Below we discuss how to compute "Gaussian-like" lower bounds to this likelihood, which give rise to approximate Gaussian posteriors.

21.8.1.2 Multi-task learning

One important application of Bayesian inference for logistic regression is where we have multiple related classifiers we want to fit. In this case, we want to share information between the parameters for each classifier; this requires that we maintain a posterior distibution over the parameters, so we have a measure of confidence as well as an estimate of the value. We can embed the above variational method inside of a larger hierarchical model in order to perform such multi-task learning, as described in e.g., (Braun and McAuliffe 2010).

21.8.1.3 Discrete factor analysis

Another situation where variational bounds are useful arises when we fit a factor analysis model to discrete data. This model is just like multinomial logistic regression, except the input variables are hidden factors. We need to perform inference on the hidden variables as well as the regression weights. For simplicity, we might perform point estimation of the weights, and just integrate out the hidden variables. We can do this using variational EM, where we use the variational bound in the E step. See Section 12.4 for details.

21.8.1.4 Correlated topic model

A topic model is a latent variable model for text documents and other forms of discrete data; see Section 27.3 for details. Often we assume the distribution over topics has a Dirichlet prior, but a more powerful model, known as the correlated topic model, uses a Gaussian prior, which can model correlations more easily (see Section 27.4.1 for details). Unfortunately, this also involves the lse function. However, we can use our variational bounds in the context of a variational EM algorithm, as we will see later.

21.8.2 Bohning's quadratic bound to the log-sum-exp function

All of the above examples require dealing with multiplying a Gaussian prior by a multinomial likelihood; this is difficult because of the log-sum-exp (lse) term. In this section, we derive a way to derive a "Gaussian-like" lower bound on this likelihood.

Consider a Taylor series expansion of the lse function around $\boldsymbol{\psi}_i \in \mathbb{R}^M$:

$$\begin{aligned}
\text{lse}(\boldsymbol{\eta}_i) &= \text{lse}(\boldsymbol{\psi}_i) + (\boldsymbol{\eta}_i - \boldsymbol{\psi}_i)^T \mathbf{g}(\boldsymbol{\psi}_i) + \frac{1}{2}(\boldsymbol{\eta}_i - \boldsymbol{\psi}_i)^T \mathbf{H}(\boldsymbol{\psi}_i)(\boldsymbol{\eta}_i - \boldsymbol{\psi}_i) && (21.158)\\
\mathbf{g}(\boldsymbol{\psi}_i) &= \exp[\boldsymbol{\psi}_i - \text{lse}(\boldsymbol{\psi}_i)] = \mathcal{S}(\boldsymbol{\psi}_i) && (21.159)\\
\mathbf{H}(\boldsymbol{\psi}_i) &= \text{diag}(\mathbf{g}(\boldsymbol{\psi}_i)) - \mathbf{g}(\boldsymbol{\psi}_i)\mathbf{g}(\boldsymbol{\psi}_i)^T && (21.160)
\end{aligned}$$

where $\mathbf{g}$ and $\mathbf{H}$ are the gradient and Hessian of lse, and $\boldsymbol{\psi}_i \in \mathbb{R}^M$ is chosen such that equality holds. An upper bound to lse can be found by replacing the Hessian matrix $\mathbf{H}(\boldsymbol{\psi}_i)$ with a matrix $\mathbf{A}_i$ such that $\mathbf{A}_i \prec \mathbf{H}(\boldsymbol{\psi}_i)$. (Bohning 1992) showed that this can be achieved if we use the matrix $\mathbf{A}_i = \frac{1}{2}\left[\mathbf{I}_M - \frac{1}{M+1}\mathbf{1}_M\mathbf{1}_M^T\right]$. (Recall that $M+1 = C$ is the number of classes.) Note that $\mathbf{A}_i$ is independent of $\boldsymbol{\psi}_i$; however, we still write it as $\mathbf{A}_i$ (rather than dropping the i subscript), since other bounds that we consider below will have a data-dependent curvature term. The upper bound on lse therefore becomes

$$\begin{aligned}
\text{lse}(\boldsymbol{\eta}_i) &\leq \frac{1}{2}\boldsymbol{\eta}_i^T \mathbf{A}_i \boldsymbol{\eta}_i - \mathbf{b}_i^T \boldsymbol{\eta}_i + c_i && (21.161)\\
\mathbf{A}_i &= \frac{1}{2}\left[\mathbf{I}_M - \frac{1}{M+1}\mathbf{1}_M\mathbf{1}_M^T\right] && (21.162)\\
\mathbf{b}_i &= \mathbf{A}_i\boldsymbol{\psi}_i - \mathbf{g}(\boldsymbol{\psi}_i) && (21.163)\\
c_i &= \frac{1}{2}\boldsymbol{\psi}_i^T \mathbf{A}_i \boldsymbol{\psi}_i - \mathbf{g}(\boldsymbol{\psi}_i)^T\boldsymbol{\psi}_i + \text{lse}(\boldsymbol{\psi}_i) && (21.164)
\end{aligned}$$

where $\boldsymbol{\psi}_i \in \mathbb{R}^M$ is a vector of variational parameters.

We can use the above result to get the following lower bound on the softmax likelihood:

$$\log p(y_i = c|\mathbf{x}_i, \mathbf{w}) \geq \left[\mathbf{y}_i^T \mathbf{X}_i \mathbf{w} - \frac{1}{2}\mathbf{w}^T \mathbf{X}_i \mathbf{A}_i \mathbf{X}_i \mathbf{w} + \mathbf{b}_i^T \mathbf{X}_i \mathbf{w} - c_i\right]_c \tag{21.165}$$

To simplify notation, define the pseudo-measurement

$$\tilde{\mathbf{y}}_i \triangleq \mathbf{A}_i^{-1}(\mathbf{b}_i + \mathbf{y}_i) \tag{21.166}$$

Then we can get a "Gaussianized" version of the observation model:

$$p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}) \geq f(\mathbf{x}_i, \boldsymbol{\psi}_i)\, \mathcal{N}(\tilde{\mathbf{y}}_i|\mathbf{X}_i\mathbf{w}, \mathbf{A}_i^{-1}) \tag{21.167}$$

where $f(\mathbf{x}_i, \boldsymbol{\psi}_i)$ is some function that does not depend on $\mathbf{w}$. Given this, it is easy to compute the posterior $q(\mathbf{w}) = \mathcal{N}(\mathbf{m}_N, \mathbf{V}_N)$, using Bayes rule for Gaussians. Below we will explain how to update the variational parameters $\boldsymbol{\psi}_i$.

21.8.2.1 Applying Bohning's bound to multinomial logistic regression

Let us see how to apply this bound to multinomial logistic regression. From Equation 21.13, we can define the goal of variational inference as maximizing

$$\begin{aligned} L(q) &\triangleq -\mathbb{KL}\left(q(\mathbf{w})||p(\mathbf{w}|\mathcal{D})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \mathbf{w})\right] &(21.168)\\ &= -\mathbb{KL}\left(q(\mathbf{w})||p(\mathbf{w}|\mathcal{D})\right) + \mathbb{E}_q\left[\sum_{i=1}^N \mathbf{y}_i^T\boldsymbol{\eta}_i - \text{lse}(\boldsymbol{\eta}_i)\right] &(21.169)\\ &= -\mathbb{KL}\left(q(\mathbf{w})||p(\mathbf{w}|\mathcal{D})\right) + \sum_{i=1}^N \mathbf{y}_i^T\mathbb{E}_q\left[\boldsymbol{\eta}_i\right] - \sum_{i=1}^N \mathbb{E}_q\left[\text{lse}(\boldsymbol{\eta}_i)\right] &(21.170)\end{aligned}$$

where $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{V}_N)$ is the approximate posterior. The first term is just the KL divergence between two Gaussians, which is given by

$$\begin{aligned} -\mathbb{KL}\left(\mathcal{N}(\mathbf{m}_0, \mathbf{V}_0)||\mathcal{N}(\mathbf{m}_N, \mathbf{V}_N)\right) &= -\frac{1}{2}\left[\text{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}|\right.\\ &\quad \left. +(\mathbf{m}_N - \mathbf{m}_0)^T\mathbf{V}_0^{-1}(\mathbf{m}_N - \mathbf{m}_0) - DM\right] &(21.171)\end{aligned}$$

where DM is the dimensionality of the Gaussian, and we assume a prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{m}_0, \mathbf{V}_0)$, where typically $\boldsymbol{\mu}_0 = \mathbf{0}_{DM}$, and $\mathbf{V}_0$ is block diagonal. The second term is simply

$$\sum_{i=1}^N \mathbf{y}_i^T\mathbb{E}_q\left[\boldsymbol{\eta}_i\right] = \sum_{i=1}^N \mathbf{y}_i^T\tilde{\mathbf{m}}_i \tag{21.172}$$

where $\tilde{\mathbf{m}}_i \triangleq \mathbf{X}_i\mathbf{m}_N$. The final term can be lower bounded by taking expectations of our quadratic upper bound on lse as follows:

$$-\sum_{i=1}^N \mathbb{E}_q\left[\text{lse}(\boldsymbol{\eta}_i)\right] \geq -\frac{1}{2}\text{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\mathbf{m}}_i\mathbf{A}_i\tilde{\mathbf{m}}_i + \mathbf{b}_i^T\tilde{\mathbf{m}}_i - c_i \tag{21.173}$$

where $\tilde{\mathbf{V}}_i \triangleq \mathbf{X}_i\mathbf{V}_N\mathbf{X}_i^T$. Putting it altogether, we have

$$\begin{aligned} L_{QJ}(q) &\geq -\frac{1}{2}\left[\text{tr}(\mathbf{V}_N\mathbf{V}_0^{-1}) - \log|\mathbf{V}_N\mathbf{V}_0^{-1}| + (\mathbf{m}_N - \mathbf{m}_0)^T\mathbf{V}_0^{-1}(\mathbf{m}_N - \mathbf{m}_0)\right]\\ &\quad -\frac{1}{2}DM + \sum_{i=1}^N \mathbf{y}_i^T\tilde{\mathbf{m}}_i - \frac{1}{2}\text{tr}(\mathbf{A}_i\tilde{\mathbf{V}}_i) - \frac{1}{2}\tilde{\mathbf{m}}_i\mathbf{A}_i\tilde{\mathbf{m}}_i + \mathbf{b}_i^T\tilde{\mathbf{m}}_i - c_i &(21.174)\end{aligned}$$

This lower bound combines Jensen's inequality (as in mean field inference), plus the quadratic lower bound due to the lse term, so we write it as L_{QJ}.

We will use coordinate ascent to optimize this lower bound. That is, we update the variational posterior parameters $\mathbf{V}_N$ and $\mathbf{m}_N$, and then the variational likelihood parameters $\boldsymbol{\psi}_i$. We leave the detailed derivation as an exercise, and just state the results. We have

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \sum_{i=1}^{N} \mathbf{X}_i^T \mathbf{A}_i \mathbf{X}_i\right)^{-1} \tag{21.175}$$

$$\mathbf{m}_N = \mathbf{V}_n \left(\mathbf{V}_0^{-1}\mathbf{m}_0 + \sum_{i=1}^{N} \mathbf{X}_i^T(\mathbf{y}_i + \mathbf{b}_i)\right) \tag{21.176}$$

$$\boldsymbol{\psi}_i = \tilde{\mathbf{m}}_i = \mathbf{X}_i \mathbf{m}_N \tag{21.177}$$

We can exploit the fact that $\mathbf{A}_i$ is a constant matrix, plus the fact that $\mathbf{X}_i$ has block structure, to simplify the first two terms as follows:

$$\mathbf{V}_N = \left(\mathbf{V}_0 + \mathbf{A} \otimes \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right)^{-1} \tag{21.178}$$

$$\mathbf{m}_N = \mathbf{V}_n \left(\mathbf{V}_0^{-1}\mathbf{m}_0 + \sum_{i=1}^{N} (\mathbf{y}_i + \mathbf{b}_i) \otimes \mathbf{x}_i\right) \tag{21.179}$$

where $\otimes$ denotes the kronecker product. See Algorithm 15 for some pseudocode, and `http://www.cs.ubc.ca/~emtiyaz/software/catLGM.html` for some Matlab code.

Algorithm 21.1: Variational inference for multi-class logistic regression using Bohning's bound

```
1  Input: $y_i \in \{1, \ldots, C\}$, $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1 : N$, prior $\mathbf{m}_0$, $\mathbf{V}_0$ ;
2  Define $M := C - 1$; dummy encode $\mathbf{y}_i \in \{0,1\}^M$; define $\mathbf{X}_i = \text{blockdiag}(\mathbf{x}_i^T)$ ;
3  Define $\mathbf{y} := [\mathbf{y}_1; \ldots; \mathbf{y}_N]$, $\mathbf{X} := [\mathbf{X}_1; \ldots; \mathbf{X}_N]$ and $\mathbf{A} := \frac{1}{2}\left[\mathbf{I}_M - \frac{1}{M+1}\mathbf{1}_M\mathbf{1}_M^T\right]$;
4  $\mathbf{V}_N := \left(\mathbf{V}_0^{-1} + \sum_{i=1}^{n} \mathbf{X}_i^T \mathbf{A} \mathbf{X}_i\right)^{-1}$;
5  Initialize $\mathbf{m}_N := \mathbf{m}_0$;
6  repeat
7      $\boldsymbol{\psi} := \mathbf{X}\mathbf{m}_N$;
8      $\boldsymbol{\Psi} := \text{reshape}(\mathbf{m}, M, N)$;
9      $\mathbf{G} := \exp(\boldsymbol{\Psi} - \text{lse}(\boldsymbol{\Psi}))$;
10     $\mathbf{B} := \mathbf{A}\boldsymbol{\Psi} - \mathbf{G}$;
11     $\mathbf{b} := (\mathbf{B})$ ;
12     $\mathbf{m}_N := \mathbf{V}_N \left(\mathbf{V}_0^{-1}\mathbf{m}_0 + \mathbf{X}^T(\mathbf{y} + \mathbf{b})\right)$;
13     Compute the lower bound $L_{QJ}$ using Equation 21.174;
14 until converged;
15 Return $\mathbf{m}_N$ and $\mathbf{V}_N$;
```

21.8.3 Bounds for the sigmoid function

In many models, we just have binary data. In this case, we have $y_i \in \{0, 1\}$, $M = 1$ and $\eta_i = \mathbf{w}^T\mathbf{x}_i$ where $\mathbf{w} \in \mathbb{R}^D$ is a weight vector (not matrix). In this case, the Bohning bound

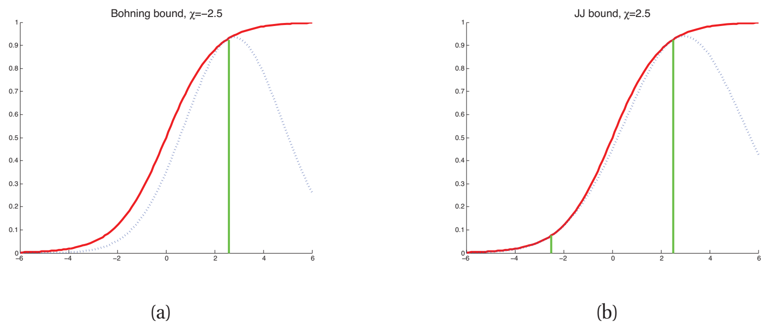


Figure 21.10 Quadratic lower bounds on the sigmoid (logistic) function. In solid red, we plot $\text{sigm}(x)$ vs x. In dotted blue, we plot the lower bound $L(x, \xi)$ vs x for $\xi = 2.5$. (a) Bohning bound. This is tight at $-\xi = 2.5$. (b) JJ bound. This is tight at $\xi = \pm 2.5$. Figure generated by `sigmoidLowerBounds`.

becomes

$$\log(1+e^\eta) \leq \frac{1}{2}a\eta^2 - b\eta + c \tag{21.180}$$
$$a = \frac{1}{4} \tag{21.181}$$
$$b = A\psi - (1+e^{-\psi})^{-1} \tag{21.182}$$
$$c = \frac{1}{2}A\psi^2 - (1+e^{-\psi})^{-1}\psi + \log(1+e^\psi) \tag{21.183}$$

It is possible to derive an alternative quadratic bound for this case, as shown in (Jaakkola and Jordan 1996b, 2000). This has the following form

$$\log(1+e^\eta) \leq \lambda(\xi)(\eta^2 - \xi^2) + \frac{1}{2}(\eta - \xi) + \log(1+e^\xi) \tag{21.184}$$
$$\lambda(\xi) \triangleq \frac{1}{4\xi}\tanh(\xi/2) = \frac{1}{2\xi}\left[\text{sigm}(\xi) - \frac{1}{2}\right] \tag{21.185}$$

We shall refer to this as the **JJ bound**, after its inventors, (Jaakkola and Jordan 1996b, 2000).

To facilitate comparison with Bohning's bound, let us rewrite the JJ bound as a quadratic form as follows

$$\log(1+e^\eta) \leq \frac{1}{2}a(\xi)\eta^2 - b(\xi)\eta + c(\xi) \tag{21.186}$$
$$a(\xi) = 2\lambda(\xi) \tag{21.187}$$
$$b(\xi) = -\frac{1}{2} \tag{21.188}$$
$$c(\xi) = -\lambda(\xi)\xi^2 - \frac{1}{2}\xi + \log(1+e^\xi) \tag{21.189}$$

The JJ bound has an adaptive curvature term, since a depends on ξ. In addition, it is tight at two points, as is evident from Figure 21.10(b). By contrast, the Bohning bound is a constant curvature bound, and is only tight at one point, as is evident from Figure 21.10(a).

If we wish to use the JJ bound for binary logistic regression, we can make some small modifications to Algorithm 15. First, we use the new definitions for a_i, b_i and c_i. The fact that a_i is not constant when using the JJ bound, unlike when using the Bohning bound, means we cannot compute $\mathbf{V}_N$ outside of the main loop, making the method a constant factor slower. Next we note that $\mathbf{X}_i = \mathbf{x}_i^T$, so the updates for the posterior become

$$\mathbf{V}_N^{-1} = \mathbf{V}_0^{-1} + 2\sum_{i=1}^N \lambda(\xi_i)\mathbf{x}_i\mathbf{x}_i^T \tag{21.190}$$

$$\mathbf{m}_N = \mathbf{V}_N \left(\mathbf{V}_0^{-1}\mathbf{m}_0 + \sum_{i=1}^N (y_i - \frac{1}{2})\mathbf{x}_i \right) \tag{21.191}$$

Finally, to compute the update for ξ_i, we isolate the terms in L_{QJ} that depend on ξ_i to get

$$L(\boldsymbol{\xi}) = \sum_{i=1}^N \left\{ \ln \text{sigm}(\xi_i) - \xi_i/2 - \lambda(\xi_i)(\mathbf{x}_i^T \mathbb{E}_q \left[\mathbf{w}\mathbf{w}^T\right] \mathbf{x}_i - \xi_i^2) \right\} + \text{const} \tag{21.192}$$

Optimizing this wrt ξ_i gives the equation

$$0 = \lambda'(\xi_i)(\mathbf{x}_i^T \mathbb{E}_q \left[\mathbf{w}\mathbf{w}^T\right] \mathbf{x}_i - \xi_i^2) \tag{21.193}$$

Now $\lambda'(\xi_i)$ is monotonic for $\xi_i \geq 0$, and we do not need to consider negative values of ξ_i by symmetry of the bound around $\xi_i = 0$ (see Figure 21.10). Hence the only way to make the above expression 0 is if we have $(\mathbf{x}_i^T \mathbb{E}\left[\mathbf{w}\mathbf{w}^T\right] \mathbf{x}_i - \xi_i^2) = 0$. Hence the update becomes

$$(\xi_i^{new})^2 = \mathbf{x}_i^T(\mathbf{V}_N + \mathbf{m}_N\mathbf{m}_N^T)\mathbf{x}_i \tag{21.194}$$

Although the JJ bound is tighter than the Bohning bound, sometimes it is not tight enough in order to estimate the posterior covariance accurately. A more accurate approach, which uses a piecewise quadratic upper bound to lse, is described in (Marlin et al. 2011). By increasing the number of pieces, the bound can be made arbitrarily tight.

21.8.4 Other bounds and approximations to the log-sum-exp function *

There are several other bounds and approximations to the multiclass lse function which we can use, which we briefly summarize below. Note, however, that all of these require numerical optimization methods to compute $\mathbf{m}_N$ and $\mathbf{V}_N$, making them more complicated to implement.

21.8.4.1 Product of sigmoids

The approach in (Bouchard 2007) exploits the fact that

$$\log\left(\sum_{k=1}^K e^{\eta_k}\right) \leq \alpha + \sum_{k=1}^K \log(1 + e^{\eta_k - \alpha}) \tag{21.195}$$

It then applies the JJ bound to the term on the right.

21.8.4.2 Jensen's inequality

The approach in (Blei and Lafferty 2006a, 2007) uses Jensen's inequality as follows:

$$\begin{aligned}\mathbb{E}_q\left[\text{lse}(\boldsymbol{\eta}_i)\right] &= \mathbb{E}_q\left[\log\left(1+\sum_{c=1}^{M}\exp(\mathbf{x}_i^T\mathbf{w}_c)\right)\right] &(21.196)\\ &\le \log\left(1+\sum_{c=1}^{M}\mathbb{E}_q\left[\exp(\mathbf{x}_i^T\mathbf{w}_c)\right]\right) &(21.197)\\ &\le \log\left(1+\sum_{c=1}^{M}\exp(\mathbf{x}_i^T\mathbf{m}_{N,c}+\frac{1}{2}\mathbf{x}_i^T\mathbf{V}_{N,cc}\mathbf{x}_i)\right) &(21.198)\end{aligned}$$

where the last term follows from the mean of a log-normal distribution, which is $e^{\mu+\sigma^2/2}$.

21.8.4.3 Multivariate delta method

The approach in (Ahmed and Xing 2007; Braun and McAuliffe 2010) uses the **multivariate delta method**, which is a way to approximate moments of a function using a Taylor series expansion. In more detail, let $f(\mathbf{w})$ be the function of interest. Using a second-order approximation around $\mathbf{m}$ we have

$$f(\mathbf{w}) \approx f(\mathbf{m})+(\mathbf{w}-\mathbf{m})^T\mathbf{g}(\mathbf{w}-\mathbf{m})+\frac{1}{2}(\mathbf{w}-\mathbf{m})^T\mathbf{H}(\mathbf{w}-\mathbf{m}) \tag{21.199}$$

where $\mathbf{g}$ and $\mathbf{H}$ are the gradient and Hessian evaluated at $\mathbf{m}$. If $q(\mathbf{w})=\mathcal{N}(\mathbf{w}|\mathbf{m},\mathbf{V})$, we have

$$\mathbb{E}_q\left[f(\mathbf{w})\right] \approx f(\mathbf{m})+\frac{1}{2}\text{tr}[\mathbf{HV}] \tag{21.200}$$

If we use $f(\mathbf{w})=\text{lse}(\mathbf{X}_i\mathbf{w})$, we get

$$\mathbb{E}_q\left[\text{lse}(\mathbf{X}_i\mathbf{w})\right] \approx \text{lse}(\mathbf{X}_i\mathbf{m})+\frac{1}{2}\text{tr}[\mathbf{X}_i\mathbf{H}\mathbf{X}_i^T\mathbf{V}] \tag{21.201}$$

where $\mathbf{g}$ and $\mathbf{H}$ for the lse function are defined in Equations 21.159 and 21.160.

21.8.5 Variational inference based on upper bounds

So far, we have been concentrating on lower bounds. However, sometimes we need to use an upper bound. For example, (Saul et al. 1996) derives a mean field algorithm for sigmoid belief nets, which are DGMs in which each CPD is a logistic regression function (Neal 1992). Unlike the case of Ising models, the resulting MRF is not pairwise, but contains higher order interactions. This makes the standard mean field updates intractable. In particular, they turn out to involve computing an expression which requires evaluating

$$\mathbb{E}\left[\log(1+e^{-\sum_{j\in\text{pa}_i}w_{ij}x_j})\right] = \mathbb{E}\left[-\log\text{sigm}(\mathbf{w}_i^T\mathbf{x}_{\text{pa}(i)})\right] \tag{21.202}$$

(Notice the minus sign in front.) (Saul et al. 1996) show how to derive an upper bound on the sigmoid function so as to make this update tractable, resulting in a monotonically convergent inference procedure.

Exercises

Exercise 21.1 Laplace approximation to $p(\mu, \log\sigma|\mathcal{D})$ for a univariate Gaussian

Compute a Laplace approximation of $p(\mu, \log\sigma|\mathcal{D})$ for a Gaussian, using an uninformative prior $p(\mu, \log\sigma) \propto 1$.

Exercise 21.2 Laplace approximation to normal-gamma

Consider estimating μ and $\ell = \log\sigma$ for a Gaussian using an uniformative normal-Gamma prior. The log posterior is

$$\log p(\mu, \ell|\mathcal{D}) = -n\log\sigma - \frac{1}{2\sigma^2}[ns^2 + n(\overline{y}-\mu)^2] \tag{21.203}$$

a. Show that the first derivatives are

$$\begin{aligned}\frac{\partial}{\partial\mu}\log p(\mu, \ell|\mathcal{D}) &= \frac{n(\overline{y}-\mu)}{\sigma^2} && (21.204)\\ \frac{\partial}{\partial\ell}\log p(\mu, \ell|\mathcal{D}) &= -n + \frac{ns^2 + n(\overline{y}-\mu)^2}{\sigma^2} && (21.205)\end{aligned}$$

b. Show that the Hessian matrix is given by

$$\begin{aligned}\mathbf{H} &= \begin{pmatrix}\frac{\partial^2}{\partial\mu^2}\log p(\mu, \ell|\mathcal{D}) & \frac{\partial^2}{\partial\mu\partial\ell}\log p(\mu, \ell|\mathcal{D})\\ \frac{\partial^2}{\partial\ell^2}\log p(\mu, \ell|\mathcal{D}) & \frac{\partial^2}{\partial\ell^2}\log p(\mu, \ell|\mathcal{D})\end{pmatrix} && (21.206)\\ &= \begin{pmatrix}-\frac{n}{\sigma^2} & -2n\frac{\overline{y}-\mu}{\sigma^2}\\ -2n\frac{\overline{y}-\mu}{\sigma^2} & -\frac{2}{\sigma^2}(ns^2 + n(\overline{y}-\mu)^2)\end{pmatrix} && (21.207)\end{aligned}$$

c. Use this to derive a Laplace approximation to the posterior $p(\mu, \ell|\mathcal{D})$.

Exercise 21.3 Variational lower bound for VB for univariate Gaussian

Fill in the details of the derivation in Section 21.5.1.6.

Exercise 21.4 Variational lower bound for VB for GMMs

Consider VBEM for GMMs as in Section 21.6.1.4. Show that the lower bound has the following form

$$\begin{aligned}\mathcal{L} = {}& \mathbb{E}[\ln p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] + \mathbb{E}[\ln p(\mathbf{z}|\boldsymbol{\pi})] + \mathbb{E}[\ln p(\boldsymbol{\pi})] + \mathbb{E}[\ln p(\boldsymbol{\mu}, \boldsymbol{\Lambda})]\\ & -\mathbb{E}[\ln q(\mathbf{z})] - \mathbb{E}[\ln q(\boldsymbol{\pi})] - \mathbb{E}[\ln q(\boldsymbol{\mu}, \boldsymbol{\Lambda})] \end{aligned} \tag{21.208}$$

where

$$\mathbb{E}\left[\ln p(\mathbf{x}|\mathbf{z},\boldsymbol{\mu},\boldsymbol{\Lambda})\right] = \frac{1}{2}\sum_k N_k \Big\{\ln\tilde{\Lambda}_k - D\beta_k^{-1} - \nu_k \text{tr}(\mathbf{S}_k\mathbf{L}_k) \\ -\nu_k(\overline{\mathbf{x}}_k - \mathbf{m}_k)^T\mathbf{L}_k(\overline{\mathbf{x}}_k - \mathbf{m}_k) - D\ln(2\pi)\Big\} \tag{21.209}$$

$$\mathbb{E}\left[\ln p(\mathbf{z}|\boldsymbol{\pi})\right] = \sum_i\sum_k r_{ik}\ln\tilde{\pi}_k \tag{21.210}$$

$$\mathbb{E}\left[\ln p(\boldsymbol{\pi})\right] = \ln C_{dir}(\boldsymbol{\alpha}_0) + (\alpha_0 - 1)\sum_k \ln\tilde{\pi}_k \tag{21.211}$$

$$\mathbb{E}\left[\ln p(\boldsymbol{\mu},\boldsymbol{\Lambda})\right] = \frac{1}{2}\sum_k\Big\{D\ln(\beta_0/2\pi) + \ln\tilde{\Lambda}_k - \frac{D\beta_0}{\beta_k} \\ -\beta_0\nu_k(\mathbf{m}_k - \mathbf{m}_0)^T\mathbf{L}_k(\mathbf{m}_k - \mathbf{m}_0) \\ + \ln C_{Wi}(\mathbf{L}_0,\nu_0) + \frac{\nu_0 - D - 1}{2}\ln\tilde{\Lambda}_k - \frac{1}{2}\nu_k\text{tr}(\mathbf{L}_0^{-1}\mathbf{L}_k)\Big\} \tag{21.212}$$

$$\mathbb{E}\left[\ln q(\mathbf{z})\right] = \sum_i\sum_k r_{ik}\ln r_{ik} \tag{21.213}$$

$$\mathbb{E}\left[\ln q(\boldsymbol{\pi})\right] = \sum_k(\alpha_k - 1)\ln\tilde{\pi}_k + \ln C_{dir}(\boldsymbol{\alpha}) \tag{21.214}$$

$$\mathbb{E}\left[\ln q(\boldsymbol{\mu},\boldsymbol{\Lambda})\right] = \sum_k\left\{\frac{1}{2}\ln\tilde{\Lambda}_k + \frac{D}{2}\ln\left(\frac{\beta_k}{2\pi}\right) - \frac{D}{2} - \mathbb{H}\left(q(\boldsymbol{\Lambda}_k)\right)\right\} \tag{21.215}$$

where the normalization constant for the Dirichlet and Wishart is given by

$$C_{dir}(\boldsymbol{\alpha}) \triangleq \frac{\Gamma(\sum_k\alpha_k)}{\prod_k\Gamma(\alpha_k)} \tag{21.216}$$

$$C_{Wi}(\mathbf{L},\nu) \triangleq |\mathbf{L}|^{-\nu/2}\left(2^{\nu D/2}\Gamma_D(\nu/2)\right)^{-1} \tag{21.217}$$

$$\Gamma_D(\alpha) \triangleq \pi^{D(D-1)/4}\prod_{j=1}^{D}\Gamma\left(\alpha + (1-j)/2\right) \tag{21.218}$$

where $\Gamma_D(\nu)$ is the multivariate Gamma function. Finally, the entropy of the Wishart is given by

$$\mathbb{H}\left(\text{Wi}(\mathbf{L},\nu)\right) = -\ln C_{Wi}(\mathbf{L},\nu) - \frac{\nu - D - 1}{2}\mathbb{E}\left[\ln|\boldsymbol{\Lambda}|\right] + \frac{\nu D}{2} \tag{21.219}$$

where $\mathbb{E}\left[\ln|\boldsymbol{\Lambda}|\right]$ is given in Equation 21.131.

Exercise 21.5 Derivation of $\mathbb{E}\left[\log\pi_k\right]$ under a Dirichlet distribution
Show that

$$\exp(\mathbb{E}\left[\log\pi_k\right]) = \frac{\exp(\Psi(\alpha_k))}{\exp(\Psi(\sum_{k'}\alpha_{k'}))} \tag{21.220}$$

where $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$.

Exercise 21.6 Alternative derivation of the mean field updates for the Ising model
Derive Equation 21.50 by directly optimizing the variational free energy one term at a time.

Exercise 21.7 Forwards vs reverse KL divergence

(Source: Exercise 33.7 of (MacKay 2003).) Consider a factored approximation $q(x, y) = q(x)q(y)$ to a joint distribution $p(x, y)$. Show that to minimize the forwards KL $\mathbb{KL}\left(p||q\right)$ we should set $q(x) = p(x)$ and $q(y) = p(y)$, i.e., the optimal approximation is a product of marginals

Now consider the following joint distribution, where the rows represent y and the columns x.

	x			
	1	2	3	4
1	1/8	1/8	0	0
2	1/8	1/8	0	0
3	0	0	1/4	0
4	0	0	0	1/4

Show that the reverse KL $\mathbb{KL}\left(q||p\right)$ for this p has three distinct minima. Identify those minima and evaluate $\mathbb{KL}\left(q||p\right)$ at each of them. What is the value of $\mathbb{KL}\left(q||p\right)$ if we set $q(x, y) = p(x)p(y)$?

Exercise 21.8 Derivation of the structured mean field updates for FHMM

Derive the updates in Section 21.4.1.

Exercise 21.9 Variational EM for binary FA with sigmoid link

Consider the binary FA model:

$$\begin{aligned} p(\mathbf{x}_i|\mathbf{z}_i, \boldsymbol{\theta}) &= \prod_{j=1}^{D} \text{Ber}(x_{ij}|\text{sigm}(\mathbf{w}_j^T\mathbf{z}_i + \beta_j)) = \prod_{j=1}^{D} \text{Ber}(x_{ij}|\text{sigm}(\eta_{ij})) && (21.221) \\ \boldsymbol{\eta}_i &= \tilde{\mathbf{W}}\tilde{\mathbf{z}}_i && (21.222) \\ \tilde{\mathbf{z}}_i &\triangleq (\mathbf{z}_i; 1) && (21.223) \\ \tilde{\mathbf{W}} &\triangleq (\mathbf{W}, \boldsymbol{\beta}) && (21.224) \\ p(\mathbf{z}_i) &= \mathcal{N}(\mathbf{0}, \mathbf{I}) && (21.225) \end{aligned}$$

Derive an EM algorithm to fit this model, using the Jaakkola-Jordan bound. Hint: the answer is in (Tipping 1998), but the exercise asks you to derive these equations.

Exercise 21.10 VB for binary FA with probit link

In Section 11.4.6, we showed how to use EM to fit probit regression, using a model of the form $p(y_i = 1|z_i) = \mathbb{I}(z_i > 0)$, where $z_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, 1)$ is latent. Now consider the case where the inputs $\mathbf{x}_i$ are also unknown, as in binary factor analysis. Show how to fit this model using variational Bayes, making an approximation to the posterior of the form $q(\mathbf{x}, \mathbf{z}, \mathbf{W}) = \prod_{i=1}^{N} q(\mathbf{x}_i)q(z_i) \prod_{l=1}^{L} q(\mathbf{w}_l)$. Hint: $q(\mathbf{x}_i)$ and $q(\mathbf{w}_i)$ will be Gaussian, and $q(z_i)$ will be a truncated univariate Gaussian.

22 *More variational inference*

22.1 Introduction

In Chapter 21, we discussed mean field inference, which approximates the posterior by a product of marginal distributions. This allows us to use different parametric forms for each variable, which is particularly useful when performing Bayesian inference for the parameters of statistical models (such as the mean and variance of a Gaussian or GMM, or the regression weights in a GLM), as we saw when we discussed variational Bayes and VB-EM.

In this chapter, we discuss a slightly different kind of variational inference. The basic idea is to minimize $J(q) = \mathbb{KL}\left(q||\tilde{p}\right)$, where $\tilde{p}$ is the exact but unnormalized posterior as before, but where we no longer require q to be factorized. In fact, we do not even require q to be a globally valid joint distribution. Instead, we only require that q is locally consistent, meaning that the joint distribution of two adjacent nodes agrees with the corresponding marginals (we will define this more precisely below).

In addition to this new kind of inference, we will discuss approximate methods for MAP state estimation in discrete graphical models. It turns out that algorithms for solving the MAP problem are very similar to some approximate methods for computing marginals, as we will see.

22.2 Loopy belief propagation: algorithmic issues

There is a very simple approximate inference algorithm for discrete (or Gaussian) graphical models known as **loopy belief propagation** or **LBP**. The basic idea is extremely simple: we apply the belief propagation algorithm of Section 20.2 to the graph, even if it has loops (i.e., even if it is not a tree). This method is simple and efficient, and often works well in practice, outperforming mean field (Weiss 2001). In this section, we discuss the algorithm in more detail. In the next section, we analyse this algorithm in terms of variational inference.

22.2.1 A brief history

When applied to loopy graphs, BP is not guaranteed to give correct results, and may not even converge. 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 — (Pearl 1988, p.195)

Despite these reservations, Pearl advocated the use of belief propagation in loopy networks as an approximation scheme (J. Pearl, personal communication) and exercise 4.7 in (Pearl 1988) investigates the quality of the approximation when it is applied to a particular loopy belief network.

However, the main impetus behind the interest in BP arose when McEliece et al. (1998) showed that a popular algorithm for error correcting codes known as turbo codes (Berrou et al. 1993) could be viewed as an instance of BP applied to a certain kind of graph. This was an important observation since turbo codes have gotten very close to the theoretical lower bound on coding efficiency proved by Shannon. (Another approach, known as low density parity check or LDPC codes, has achieved comparable performance; it also uses LBP for decoding — see Figure 22.1 for an example.) In (Murphy et al. 1999), LBP was experimentally shown to also work well for inference in other kinds of graphical models beyond the error-correcting code context, and since then, the method has been widely used in many different applications.

22.2.2 LBP on pairwise models

We now discuss how to apply LBP to an undirected graphical model with pairwise factors (we discuss the directed case, which can involve higher order factors, in the next section). The method is simple: just continually apply Equations 20.11 and 20.10 until convergence. See Algorithm 8 for the pseudocode, and `beliefPropagation` for some Matlab code. We will discuss issues such as convergence and accuracy of this method shortly.

Algorithm 22.1: Loopy belief propagation for a pairwise MRF

```
1 Input: node potentials $\psi_s(x_s)$, edge potentials $\psi_{st}(x_s, x_t)$;
2 Initialize messages $m_{s\to t}(x_t) = 1$ for all edges $s - t$;
3 Initialize beliefs $\text{bel}_s(x_s) = 1$ for all nodes $s$;
4 repeat
5     Send message on each edge
      $m_{s\to t}(x_t) = \sum_{x_s} \left(\psi_s(x_s)\psi_{st}(x_s, x_t) \prod_{u\in \text{nbr}_s \setminus t} m_{u\to s}(x_s)\right)$;
6     Update belief of each node $\text{bel}_s(x_s) \propto \psi_s(x_s) \prod_{t\in \text{nbr}_s} m_{t\to s}(x_s)$;
7 until beliefs don't change significantly;
8 Return marginal beliefs $\text{bel}_s(x_s)$;
```

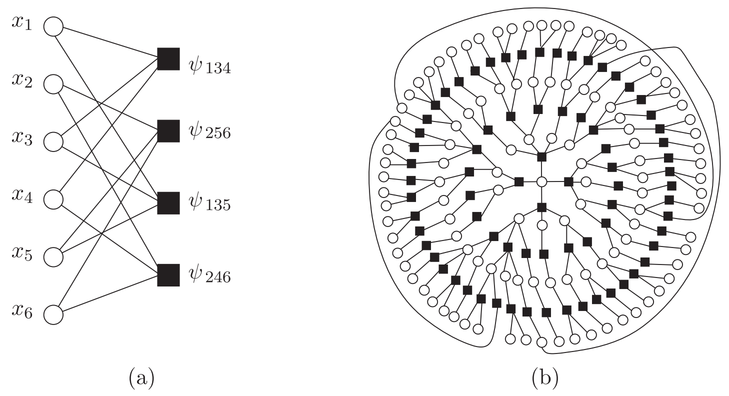


Figure 22.1 (a) A simple factor graph representation of a (2,3) low-density parity check code (factor graphs are defined in Section 22.2.3.1). Each message bit (hollow round circle) is connected to two parity factors (solid black squares), and each parity factor is connected to three bits. Each parity 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.) Source: Figure 2.9 from (Wainwright and Jordan 2008b). Used with kind permission of Martin Wainwright.

22.2.3 LBP on a factor graph

To handle models with higher-order clique potentials (which includes directed models where some nodes have more than one parent), it is useful to use a representation known as a factor graph. We explain this representation below, and then describe how to apply LBP to such models.

22.2.3.1 Factor graphs

A **factor graph** (Kschischang et al. 2001; Frey 2003) is a graphical representation that unifies directed and undirected models, and which simplifies certain message passing algorithms. More precisely, a factor graph is an undirected bipartite graph with two kinds of nodes. Round nodes represent variables, square nodes represent factors, and there is an edge from each variable to every factor that mentions it. For example, consider the MRF in Figure 22.2(a). If we assume one potential per maximal clique, we get the factor graph in Figure 22.2(b), which represents the function

$$f(x_1, x_2, x_3, x_4) = f_{124}(x_1, x_2, x_4) f_{234}(x_2, x_3, x_4) \tag{22.1}$$

If we assume one potential per edge. we get the factor graph in Figure 22.2(c), which represents the function

$$f(x_1, x_2, x_3, x_4) = f_{14}(x_1, x_4) f_{12}(x_1, x_2) f_{34}(x_3, x_4) f_{23}(x_2, x_3) f_{24}(x_2, x_4) \tag{22.2}$$

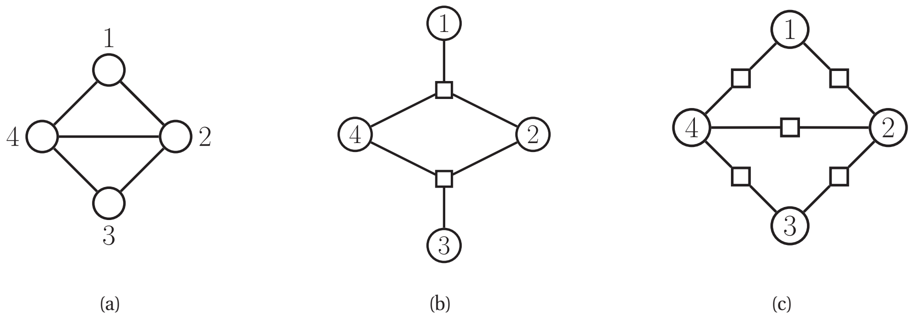


Figure 22.2 (a) A simple UGM. (b) A factor graph representation assuming one potential per maximal clique. (c) A factor graph representation assuming one potential per edge.

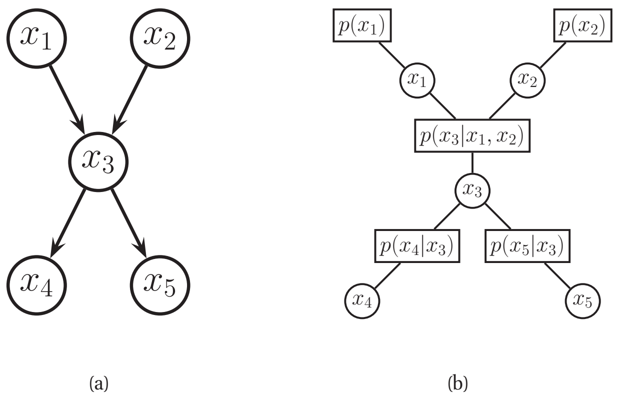


Figure 22.3 (a) A simple DGM. (b) Its corresponding factor graph. Based on Figure 5 of (Yedidia et al. 2001)..

We can also convert a DGM to a factor graph: just create one factor per CPD, and connect that factor to all the variables that use that CPD. For example, Figure 22.3 represents the following factorization:

$$f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1)f_2(x_2)f_{123}(x_1, x_2, x_3)f_{34}(x_3, x_4)f_{35}(x_3, x_5) \tag{22.3}$$

where we define $f_{123}(x_1, x_2, x_3) = p(x_3|x_1, x_2)$, etc. If each node has at most one parent (and hence the graph is a chain or simple tree), then there will be one factor per edge (root nodes can have their prior CPDs absorvbed into their children's factors). Such models are equivalent to pairwise MRFs.

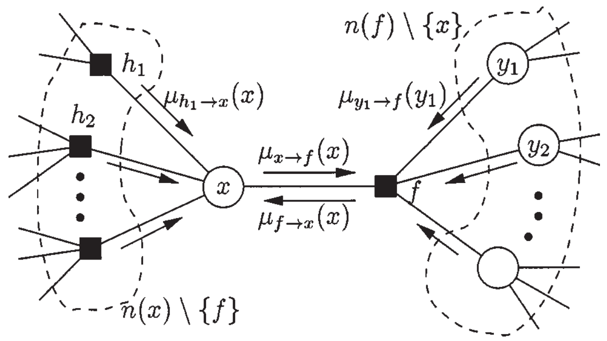


Figure 22.4 Message passing on a bipartite factor graph. Square nodes represent factors, and circles represent variables. Source: Figure 6 of (Kschischang et al. 2001). Used with kind permission of Brendan Frey.

22.2.3.2 BP on a factor graph

We now derive a version of BP that sends messages on a factor graph, as proposed in (Kschischang et al. 2001). Specifically, we now have two kinds of messages: variables to factors

$$m_{x \to f}(x) = \prod_{h \in \text{nbr}(x) \setminus \{f\}} m_{h \to x}(x) \tag{22.4}$$

and factors to variables:

$$m_{f \to x}(x) = \sum_{\mathbf{y}} f(x, \mathbf{y}) \prod_{y \in \text{nbr}(f) \setminus \{x\}} m_{y \to f}(y) \tag{22.5}$$

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

$$\text{bel}(x) \propto \prod_{f \in \text{nbr}(x)} m_{f \to x}(x) \tag{22.6}$$

In the following sections, we will focus on LBP for pairwise models, rather than for factor graphs, but this is just for notational simplicity.

22.2.4 Convergence

LBP does not always converge, and even when it does, it may converge to the wrong answers. This raises several questions: how can we predict when convergence will occur? what can we do to increase the probability of convergence? what can we do to increase the rate of convergence? We briefly discuss these issues below. We then discuss the issue of accuracy of the results at convergence.

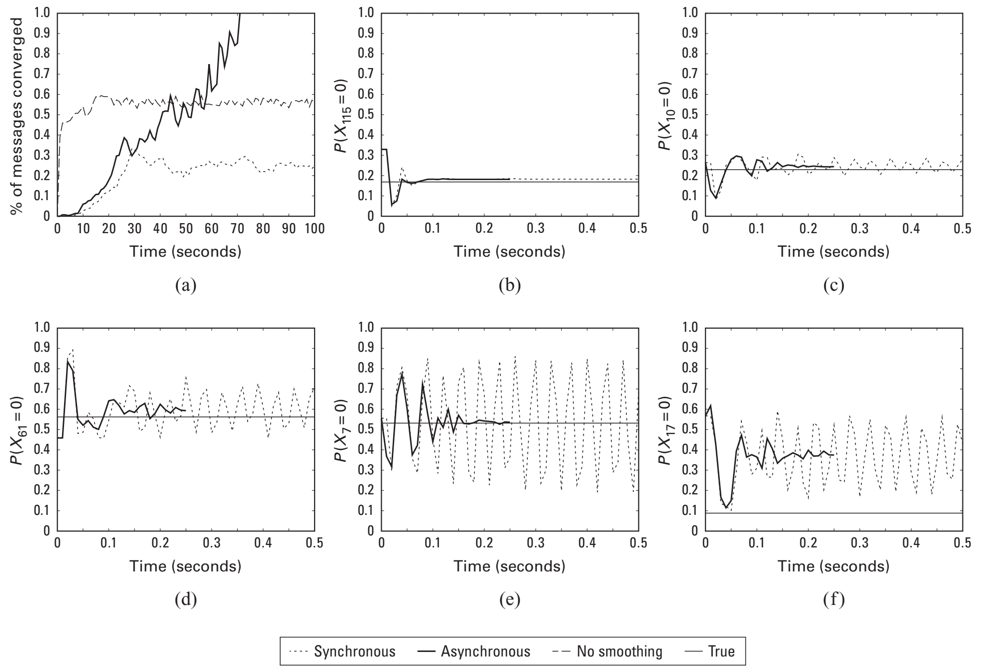


Figure 22.5 Illustration of the behavior of loopy belief propagation on an 11×11 Ising grid with random potentials, $w_{ij} \sim \text{Unif}(-C, C)$, where $C = 11$. For larger C, inference becomes harder. (a) Percentage of messasges that have converged vs time for 3 different update schedules: Dotted = damped sychronous (few nodes converge), dashed = undamped asychnronous (half the nodes converge), solid = damped asychnronous (all nodes converge). (b-f) Marginal beliefs of certain nodes vs time. Solid straight line = truth, dashed = sychronous, solid = damped asychronous. Source: Figure 11.C.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

22.2.4.1 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 22.6 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 (Wainwright and Jordan 2008b) and references therein for more information.

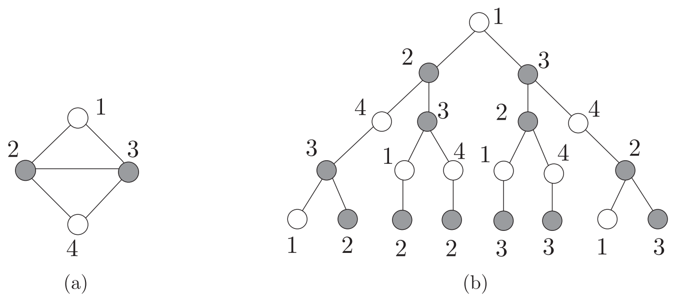


Figure 22.6 (a) A simple loopy graph. (b) The computation tree, rooted at node 1, after 4 rounds of message passing. Nodes 2 and 3 occur more often in the tree because they have higher degree than nodes 1 and 2. Source: Figure 8.2 of (Wainwright and Jordan 2008b). Used with kind permission of Martin Wainwright.

22.2.4.2 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 reduce the chance of oscillation is to use **damping**. That is, instead of sending the message M_{ts}^k, we send a damped message of the form

$$\tilde{M}_{ts}^k(x_s) = \lambda M_{ts}(x_s) + (1-\lambda)\tilde{M}_{ts}^{k-1}(x_s) \tag{22.7}$$

where $0 \leq \lambda \leq 1$ 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 22.5, where we see that damped updating results in convergence much more often than undamped updating.

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 (Yuille 2001; Welling and Teh 2001). 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. In Section 22.4.2, we will see a different convergent version of BP that is widely used.

22.2.4.3 Increasing the convergence rate: message scheduling

Even if LBP converges, it may take a long time. 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

$$\mathbf{m}^{k+1} = (f_1(\mathbf{m}^k), \ldots, f_E(\mathbf{m}^k)) \tag{22.8}$$

where E is the number of edges, and $f_{st}(\mathbf{m})$ is the function that computes the message for edge $s \to t$ given all the old messages. This is analogous to the Jacobi method for solving linear systems of equations. It is well known (Bertsekas 1997) 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

$$\mathbf{m}_i^{k+1} = f_i\left(\{\mathbf{m}_j^{k+1} : j < i\}, \{\mathbf{m}_j^k : j > i\}\right) \tag{22.9}$$

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 22.5, where we see that (damped) asynchronous updating results in convergence much more often than synchronous updating.

A smarter approach is to pick a set of spanning trees, and then to perform an up-down sweep on one tree at a time, keeping all the other messages fixed. This is known as **tree reparameterization** (TRP) (Wainwright et al. 2001), which should not be confused with the more sophisticated tree-reweighted BP (often abbreviated to TRW) to be discussed in Section 22.4.2.1.

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. (Elidan et al. 2006) 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_{st} at iteration k to be

$$r(s,t,k) = ||\log m_{st} - \log m_{st}^k||_\infty = \max_i |\log \frac{m_{st}(i)}{m_{st}^k(i)}| \tag{22.10}$$

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_{st} (i.e., messages of the form m_{tu} 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 (Elidan et al. 2006), they showed (experimentally) that this method converges more often, and much faster, than using sychronous updating, asynchronous updating with a fixed order, and the TRP approach.

A refinement of residual BP was presented in (Sutton and McCallum 2007). 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.

22.2.5 Accuracy of LBP

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 (Weiss 2000). For more general graphs, one can bound the error in the approximate marginals computed by LBP, as shown in (Wainwright et al. 2003; Vinyals et al. 2010). Much stronger results are available in the case of Gaussian models (Weiss and Freeman 2001a; Johnson et al. 2006; Bickson 2009). In particular, in the Gaussian case, if the method converges, the means are exact, although the variances are not (typically the beliefs are over confident).

22.2.6 Other speedup tricks for LBP *

There are several tricks one can use to make BP run faster. We discuss some of them below.

22.2.6.1 Fast message computation for large state spaces

The cost of computing each message in BP (whether in a tree or a loopy graph) is $O(K^f)$, where K is the number of states, and f is the size of the largest factor ($f = 2$ for pairwise UGMs). In many vision problems (e.g., image denoising), K is quite large (say 256), because it represents the discretization of some underlying continuous space, so $O(K^2)$ per message is too expensive. Fortunately, for certain kinds of pairwise potential functions of the form $\psi_{st}(x_s, x_t) = \psi(x_s - x_t)$, one can compute the sum-product messages in $O(K \log K)$ time using the fast Fourier transform or FFT, as explained in (Felzenszwalb and Huttenlocher 2006). The key insight is that message computation is just convolution:

$$M_{st}^k(x_t) = \sum_{x_s} \psi(x_s - x_t) h(x_s) \tag{22.11}$$

where $h(x_s) = \psi_s(x_s) \prod_{v \in \text{nbr}(s) \setminus t} M_{vs}^{k-1}(x_s)$. If the potential function $\psi(z)$ is a Gaussian-like potential, we can compute the convolution in $O(K)$ time by sequentially convolving with a small number of box filters (Felzenszwalb and Huttenlocher 2006).

For the max-product case, a technique called the **distance transform** can be used to compute messages in $O(K)$ time. However, this only works if $\psi(z) = \exp(-E(z))$ and where $E(z)$ has one the following forms: quadratic, $E(z) = z^2$; truncated linear, $E(z) = \min(c_1|z|, c_2)$; or Potts model, $E(z) = c\, \mathbb{I}(z \neq 0)$. See (Felzenszwalb and Huttenlocher 2006) for details.

22.2.6.2 Multi-scale methods

A method which is specific to 2d lattice structures, which commonly arise in computer vision, is based on multi-grid techniques. Such methods are widely used in numerical linear algebra, where one of the core problems is the fast solution of linear systems of equations; this is equivalent to MAP estimation in a Gaussian MRF. In the computer vision context, (Felzenszwalb and Huttenlocher 2006) suggested using the following heuristic to significantly speedup BP: construct a coarse-to-fine grid, compute messages at the coarse level, and use this to initialize messages at the level below; when we reach the bottom level, just a few iterations of standard BP are required, since long-range communication has already been achieved via the initialization process.

The beliefs at the coarse level are computed over a small number of large blocks. The local evidence is computed from the average log-probability each possible block label assigns to all the pixels in the block. The pairwise potential is based on the discrepancy between labels of neighboring blocks, taking into account their size. We can then run LBP at the coarse level, and then use this to initialize the messages one level down. Note that the *model* is still a flat grid; however, the *initialization process* exploits the multi-scale nature of the problem. See (Felzenszwalb and Huttenlocher 2006) for details.

22.2.6.3 Cascades

Another trick for handling high-dimensional state-spaces, that can also be used with exact inference (e.g., for chain-structured CRFs), is to prune out improbable states based on a computationally cheap filtering step. In fact, one can create a hierarchy of models which tradeoff speed and accuracy. This is called a computational **cascade**. In the case of chains, one can guarantee that the cascade will never filter out the true MAP solution (Weiss et al. 2010).

22.3 Loopy belief propagation: theoretical issues *

We now attempt to understand the LBP algorithm from a variational point of view. Our presentation is closely based on an excellent 300-page review article (Wainwright and Jordan 2008a). This paper is sometimes called "the monster" (by its own authors!) in view of its length and technical difficulty. This section just sketches some of the main results.

To simplify the presentation, we focus on the special case of pairwise UGMs with discrete variables and tabular potentials. Many of the results generalize to UGMs with higher-order clique potentials (which includes DGMs), but this makes the notation more complex (see (Koller and Friedman 2009) for details of the general case).

22.3.1 UGMs represented in exponential family form

We assume the distribution has the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}, G) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\} \tag{22.12}$$

where graph G has nodes $\mathcal{V}$ and edges $\mathcal{E}$. (Henceforth we will drop the explicit conditioning on $\boldsymbol{\theta}$ and G for brevity, since we assume both are known and fixed.) We can rewrite this in exponential family form as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-E(\mathbf{x})) \tag{22.13}$$

$$E(\mathbf{x}) \triangleq -\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) \tag{22.14}$$

where $\boldsymbol{\theta} = (\{\theta_{s;j}\}, \{\theta_{s,t;j,k}\})$ are all the node and edge parameters (the canonical parameters), and $\boldsymbol{\phi}(\mathbf{x}) = (\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(x_s = j, x_t = k)\})$ are all the node and edge indicator functions (the sufficient statistics). Note: we use $s, t \in \mathcal{V}$ to index nodes and $j, k \in \mathcal{X}$ to index states.

The mean of the sufficient statistics are known as the mean parameters of the model, and are given by

$$\boldsymbol{\mu} = \mathbb{E}\left[\boldsymbol{\phi}(\mathbf{x})\right] = (\{p(x_s = j)\}_s, \{p(x_s = j, x_t = k)\}_{s \neq t}) = (\{\mu_{s;j}\}_s, \{\mu_{st;jk}\}_{s \neq t}) \tag{22.15}$$

This is a vector of length $d = |\mathcal{X}||V| + |\mathcal{X}|^2|E|$, containing the node and edge marginals. It completely characterizes the distribution $p(\mathbf{x}|\boldsymbol{\theta})$, so we sometimes treat $\boldsymbol{\mu}$ as a distribution itself.

Equation 22.12 is called the **standard overcomplete representation**. It is called "overcomplete" because it ignores the sum-to-one constraints. In some cases, it is convenient to remove this redundancy. For example, consider an Ising model where $X_s \in \{0, 1\}$. The model can be written as

$$p(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{ \sum_{s \in \mathcal{V}} \theta_s x_s + \sum_{(s,t) \in \mathcal{E}} \theta_{st} x_s x_t \right\} \tag{22.16}$$

Hence we can use the following minimal parameterization

$$\boldsymbol{\phi}(\mathbf{x}) = (x_s, s \in V; x_s x_t, (s,t) \in E) \in \mathbb{R}^d \tag{22.17}$$

where $d = |V| + |E|$. The corresponding mean parameters are $\mu_s = p(x_s = 1)$ and $\mu_{st} = p(x_s = 1, x_t = 1)$.

22.3.2 The marginal polytope

The space of allowable $\boldsymbol{\mu}$ vectors is called the **marginal polytope**, and is denoted $\mathbb{M}(G)$, where G is the structure of the graph defining the UGM. This is defined to be the set of all mean parameters for the given model that can be generated from a valid probability distribution:

$$\mathbb{M}(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \exists p \quad \text{s.t.} \quad \boldsymbol{\mu} = \sum_{\mathbf{x}} \boldsymbol{\phi}(\mathbf{x}) p(\mathbf{x}) \text{ for some } p(\mathbf{x}) \geq 0, \sum_{\mathbf{x}} p(\mathbf{x}) = 1\} \tag{22.18}$$

For example, consider an Ising model. If we have just two nodes connected as $X_1 - X_2$, one can show that we have the following minimal set of constraints: $0 \leq \mu_{12}$, $0 \leq \mu_{12} \leq \mu_1$, $0 \leq \mu_{12} \leq \mu_2$, and $1 + \mu_{12} - \mu_1 - \mu_2 \geq 0$. We can write these in matrix-vector form as

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_{12} \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \tag{22.19}$$

These four constraints define a series of half-planes, whose intersection defines a polytope, as shown in Figure 22.7(a).

Since $\mathbb{M}(G)$ is obtained by taking a convex combination of the $\boldsymbol{\phi}(\mathbf{x})$ vectors, it can also be written as the **convex hull** of the feature set:

$$\mathbb{M}(G) = \text{conv}\{\phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})\} \tag{22.20}$$

For example, for a 2 node MRF $X_1 - X_2$ with binary states, we have

$$\mathbb{M}(G) = \text{conv}\{(0,0,0), (1,0,0), (0,1,0), (1,1,1)\} \tag{22.21}$$

These are the four black dots in Figure 22.7(a). We see that the convex hull defines the same volume as the intersection of half-spaces.

The marginal polytope will play a crucial role in the approximate inference algorithms we discuss in the rest of this chapter.

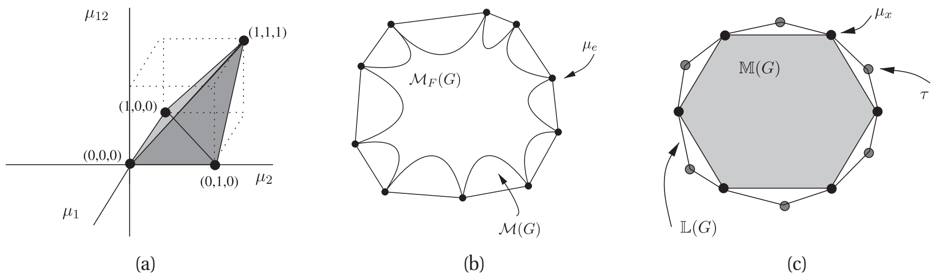


Figure 22.7 (a) Illustration of the marginal polytope for an Ising model with two variables. (b) Cartoon illustration of the set $\mathbb{M}_F(G)$, which is a nonconvex inner bound on the marginal polytope $\mathbb{M}(G)$. $\mathbb{M}_F(G)$ is used by mean field. (c) Cartoon illustration of the relationship between $\mathbb{M}(G)$ and $\mathbb{L}(G)$, which is used by loopy BP. The set $\mathbb{L}(G)$ is always an outer bound on $\mathbb{M}(G)$, and the inclusion $\mathbb{M}(G) \subset \mathbb{L}(G)$ is strict whenever G has loops. Both sets are polytopes, which can be defined as an intersection of half-planes (defined by facets), or as the convex hull of the vertices. $\mathbb{L}(G)$ actually has fewer facets than $\mathbb{M}(G)$, despite the picture. In fact, $\mathbb{L}(G)$ has $O(|\mathcal{X}||V| + |\mathcal{X}|^2|E|)$ facets, where $|\mathcal{X}|$ is the number of states per variable, $|V|$ is the number of variables, and $|E|$ is the number of edges. By contrast, $\mathbb{M}(G)$ has $O(|\mathcal{X}|^{|V|})$ facets. On the other hand, $\mathbb{L}(G)$ has more vertices than $\mathbb{M}(G)$, despite the picture, since $\mathbb{L}(G)$ contains all the binary vector extreme points $\boldsymbol{\mu} \in \mathbb{M}(G)$, plus additional fractional extreme points. Source: Figures 3.6, 5.4 and 4.2 of (Wainwright and Jordan 2008a). Used with kind permission of Martin Wainwright.

22.3.3 Exact inference as a variational optimization problem

Recall from Section 21.2 that the goal of variational inference is to find the distribution q that maximizes the **energy functional**

$$L(q) = -\mathbb{KL}\left(q||p\right) + \log Z = \mathbb{E}_q\left[\log \tilde{p}(\mathbf{x})\right] + \mathbb{H}\left(q\right) \leq \log Z \tag{22.22}$$

where $\tilde{p}(\mathbf{x}) = Zp(\mathbf{x})$ is the unnormalized posterior. If we write $\log \tilde{p}(\mathbf{x}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x})$, and we let $q = p$, then the exact energy functional becomes

$$\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}\left(\boldsymbol{\mu}\right) \tag{22.23}$$

where $\boldsymbol{\mu} = \mathbb{E}_p\left[\boldsymbol{\phi}(\mathbf{x})\right]$ is a joint distribution over all state configurations $\mathbf{x}$ (so it is valid to write $\mathbb{H}\left(\boldsymbol{\mu}\right)$). Since the KL divergence is zero when $p = q$, we know that

$$\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}\left(\boldsymbol{\mu}\right) = \log Z(\boldsymbol{\theta}) \tag{22.24}$$

This is a way to cast exact inference as a variational optimization problem.

Equation 22.24 seems easy to optimize: the objective is concave, since it is the sum of a linear function and a concave function (see Figure 2.21 to see why entropy is concave); furthermore, we are maximizing this over a convex set. However, the marginal polytope $\mathbb{M}(G)$ has exponentially many facets. In some cases, there is structure to this polytope that can be exploited by dynamic programming (as we saw in Chapter 20), but in general, exact inference takes exponential time. Most of the existing deterministic approximate inference schemes that have been proposed in the literature can be seen as different approximations to the marginal polytope, as we explain below.

22.3.4 Mean field as a variational optimization problem

We discussed mean field at length in Chapter 21. Let us re-interpret mean field inference in our new more abstract framework. This will help us compare it to other approximate methods which we discuss below.

First, let F be an edge subgraph of the original graph G, and let $\mathcal{I}(F) \subseteq \mathcal{I}$ be the subset of sufficient statistics associated with the cliques of F. Let Ω be the set of canonical parameters for the full model, and define the canonical parameter space for the submodel as follows:

$$\Omega(F) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_\alpha = 0 \; \forall \alpha \in \mathcal{I} \setminus \mathcal{I}(F)\} \tag{22.25}$$

In other words, we require that the natural parameters associated with the sufficient statistics α outside of our chosen class to be zero. For example, in the case of a fully factorized approximation, F_0, we remove all edges from the graph, giving

$$\Omega(F_0) \triangleq \{\boldsymbol{\theta} \in \Omega : \boldsymbol{\theta}_{st} = 0 \; \forall (s,t) \in E\} \tag{22.26}$$

In the case of structured mean field (Section 21.4), we set $\theta_{st} = 0$ for edges which are not in our tractable subgraph.

Next, we define the mean parameter space of the restricted model as follows:

$$\mathbb{M}_F(G) \triangleq \{\boldsymbol{\mu} \in \mathbb{R}^d : \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}(\mathbf{x})] \quad \text{for some } \boldsymbol{\theta} \in \Omega(F)\} \tag{22.27}$$

This is called an **inner approximation** to the marginal polytope, since $\mathbb{M}_F(G) \subseteq \mathbb{M}(G)$. See Figure 22.7(b) for a sketch. Note that $\mathbb{M}_F(G)$ is a non-convex polytope, which results in multiple local optima. By contrast, some of the approximations we will consider later will be convex.

We define the entropy of our approximation $\mathbb{H}(\boldsymbol{\mu}(F))$ as the entropy of the distribution $\boldsymbol{\mu}$ defined on submodel F. Then we define the **mean field energy functional** optimization problem as follows:

$$\max_{\boldsymbol{\mu} \in \mathbb{M}_F(G)} \boldsymbol{\theta}^T \boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) \leq \log Z(\boldsymbol{\theta}) \tag{22.28}$$

In the case of the fully factorized mean field approximation for pairwise UGMs, we can write this objective as follows:

$$\max_{\boldsymbol{\mu} \in \mathcal{P}^d} \sum_{s \in V} \sum_{x_s} \theta_s(x_s)\mu_s(x_s) + \sum_{(s,t) \in E} \sum_{x_s, x_t} \theta_{st}(x_s, x_t)\mu_s(x_s)\mu_t(x_t) + \sum_{s \in V} \mathbb{H}(\boldsymbol{\mu}_s) \tag{22.29}$$

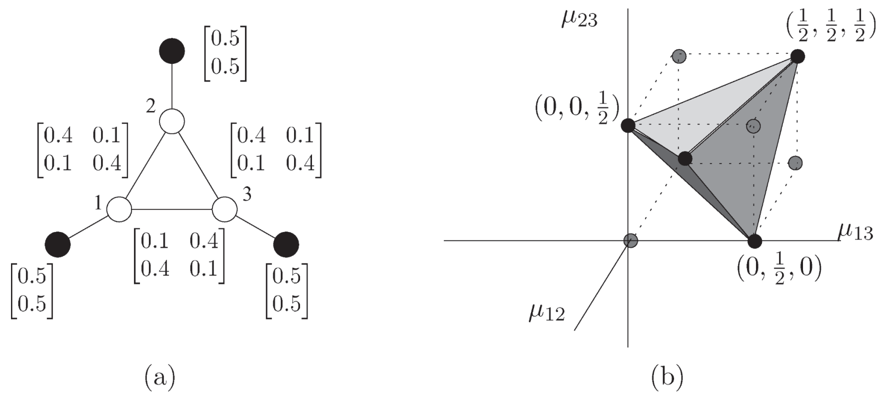
where $\boldsymbol{\mu}_s \in \mathcal{P}$, and $\mathcal{P}$ is the probability simplex over $\mathcal{X}$.

Mean field involves a concave objective being maximized over a non-convex set. It is typically optimized using coordinate ascent, since it is easy to optimize a scalar concave function over $\mathcal{P}$ for each μ_s. For example, for a pairwise UGM we get

$$\mu_s(x_s) \propto \exp(\theta_s(x_s)) \exp\left(\sum_{t \in \text{nbr}(s)} \sum_{x_t} \mu_t(x_t) \theta_{st}(x_s, x_t) \right) \tag{22.30}$$

22.3.5 LBP as a variational optimization problem

In this section, we explain how LBP can be viewed as a variational inference problem.



$\begin{bmatrix}0.5\\0.5\end{bmatrix}$
2
$\begin{bmatrix}0.4 & 0.1\\0.1 & 0.4\end{bmatrix}$
$\begin{bmatrix}0.4 & 0.1\\0.1 & 0.4\end{bmatrix}$
1
3
$\begin{bmatrix}0.1 & 0.4\\0.4 & 0.1\end{bmatrix}$
$\begin{bmatrix}0.5\\0.5\end{bmatrix}$
$\begin{bmatrix}0.5\\0.5\end{bmatrix}$
(a)
μ_{23}
$(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
$(0, 0, \frac{1}{2})$
μ_{13}
$(0, \frac{1}{2}, 0)$
μ_{12}
(b)

Figure 22.8 (a) Illustration of pairwise UGM on binary nodes, together with a set of pseudo marginals that are not globally consistent. (b) A slice of the marginal polytope illustrating the set of feasible edge marginals, assuming the node marginals are clamped at $\mu_1 = \mu_2 = \mu_3 = 0.5$. Source: Figure 4.1 of (Wainwright and Jordan 2008a). Used with kind permission of Martin Wainwright.

22.3.5.1 An outer approximation to the marginal polytope

If we want to consider all possible probability distributions which are Markov wrt our model, we need to consider all vectors $\boldsymbol{\mu} \in \mathbb{M}(G)$. Since the set $\mathbb{M}(G)$ is exponentially large, it is usually infeasible to optimize over. A standard strategy in combinatorial optimization is to relax the constraints. In this case, instead of requiring probability vector $\boldsymbol{\mu}$ to live in $\mathbb{M}(G)$, we consider a vector $\boldsymbol{\tau}$ that only satisfies the following **local consistency** constraints:

$$\sum_{x_s} \tau_s(x_s) = 1 \tag{22.31}$$

$$\sum_{x_t} \tau_{st}(x_s, x_t) = \tau_s(x_s) \tag{22.32}$$

The first constraint is called the normalization constraint, and the second is called the marginalization constraint. We then define the set

$$\mathbb{L}(G) \triangleq \{\boldsymbol{\tau} \geq 0 : (22.31) \text{ holds } \forall s \in V \text{ and } (22.32) \text{ holds } \forall (s,t) \in E\} \tag{22.33}$$

The set $\mathbb{L}(G)$ is also a polytope, but it only has $O(|V| + |E|)$ constraints. It is a convex **outer approximation** on $\mathbb{M}(G)$, as shown in Figure 22.7(c).

We call the terms $\tau_s, \tau_{st} \in \mathbb{L}(G)$ **pseudo marginals**, since they may not correspond to marginals of any valid probability distribution. As an example of this, consider Figure 22.8(a). The picture shows a set of pseudo node and edge marginals, which satisfy the local consistency requirements. However, they are not globally consistent. To see why, note that τ_{12} implies $p(X_1 = X_2) = 0.8$, τ_{23} implies $p(X_2 = X_3) = 0.8$, but τ_{13} implies $p(X_1 = X_3) = 0.2$, which is not possible (see (Wainwright and Jordan 2008b, p81) for a formal proof). Indeed, Figure 22.8(b) shows that $\mathbb{L}(G)$ contains points that are not in $\mathbb{M}(G)$.

We claim that $\mathbb{M}(G) \subseteq \mathbb{L}(G)$, with equality iff G is a tree. To see this, first consider an element $\boldsymbol{\mu} \in \mathbb{M}(G)$. Any such vector must satisfy the normalization and marginalization constraints, hence $\mathbb{M}(G) \subseteq \mathbb{L}(G)$.

Now consider the converse. Suppose T is a tree, and let $\boldsymbol{\mu} \in \mathbb{L}(T)$. By definition, this satisfies the normalization and marginalization constraints. However, any tree can be represented in the form

$$p_{\boldsymbol{\mu}}(\mathbf{x}) = \prod_{s \in V} \mu_s(x_s) \prod_{(s,t) \in E} \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} \tag{22.34}$$

Hence satsifying normalization and local consistency is enough to define a valid distribution for any tree. Hence $\boldsymbol{\mu} \in \mathbb{M}(T)$ as well.

In contrast, if the graph has loops, we have that $\mathbb{M}(G) \neq \mathbb{L}(G)$. See Figure 22.8(b) for an example of this fact.

22.3.5.2 The entropy approximation

From Equation 22.34, we can write the exact entropy of any tree structured distribution $\boldsymbol{\mu} \in \mathbb{M}(T)$ as follows:

$$\begin{aligned} \mathbb{H}(\boldsymbol{\mu}) &= \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} I_{st}(\mu_{st}) && (22.35) \\ H_s(\mu_s) &= - \sum_{x_s \in \mathcal{X}_s} \mu_s(x_s) \log \mu_s(x_s) && (22.36) \\ I_{st}(\mu_{st}) &= \sum_{(x_s, x_t) \in \mathcal{X}_s \times \mathcal{X}_t} \mu_{st}(x_s, x_t) \log \frac{\mu_{st}(x_s, x_t)}{\mu_s(x_s)\mu_t(x_t)} && (22.37) \end{aligned}$$

Note that we can rewrite the mutual information term in the form $I_{st}(\mu_{st}) = H_s(\mu_s) + H_t(\mu_t) - H_{st}(\mu_{st})$, and hence we get the following alternative but equivalent expression:

$$\mathbb{H}(\boldsymbol{\mu}) = - \sum_{s \in V} (d_s - 1) H_s(\mu_s) + \sum_{(s,t) \in E} H_{st}(\mu_{st}) \tag{22.38}$$

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

The **Bethe**[1] approximation to the entropy is simply the use of Equation 22.35 even when we don't have a tree:

$$\mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) = \sum_{s \in V} H_s(\tau_s) - \sum_{(s,t) \in E} I_{st}(\tau_{st}) \tag{22.39}$$

We define the **Bethe free energy** as

$$F_{\text{Bethe}}(\boldsymbol{\tau}) \triangleq - \left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \right] \tag{22.40}$$

We define the **Bethe energy functional** as the negative of the Bethe free energy.

1. Hans Bethe was a German-American physicist, 1906–2005.

22.3.5.3 The LBP objective

Combining the outer approximation $\mathbb{L}(G)$ with the Bethe approximation to the entropy, we get the following Bethe variational problem (BVP):

$$\min_{\boldsymbol{\tau}\in\mathbb{L}(G)} F_{\text{Bethe}}(\boldsymbol{\tau}) = \max_{\boldsymbol{\tau}\in\mathbb{L}(G)} \boldsymbol{\theta}^T\boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) \tag{22.41}$$

The space we are optimizing over is a convex set, but the objective itself is not concave (since $\mathbb{H}_{\text{Bethe}}$ is not concave). Thus there can be multiple local optima of the BVP.

The value obtained by the BVP is an approximation to $\log Z(\boldsymbol{\theta})$. In the case of trees, the approximation is exact, and in the case of models with attractive potentials, the approximation turns out to be an upper bound (Sudderth et al. 2008).

22.3.5.4 Message passing and Lagrange multipliers

In this subsection, we will show that any fixed point of the LBP algorithm defines a stationary point of the above constrained objective. Let us define the normalization constraint at $C_{ss}(\boldsymbol{\tau}) \triangleq 1 - \sum_{x_s} \tau_s(x_s)$, and the marginalization constraint as $C_{ts}(x_s;\boldsymbol{\tau}) \triangleq \tau_s(x_s) - \sum_{x_t} \tau_{st}(x_s, x_t)$ for each edge $t \to s$. We can now write the Lagrangian as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\tau},\boldsymbol{\lambda};\boldsymbol{\theta}) \quad &\triangleq \quad \boldsymbol{\theta}^T\boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau}) + \sum_s \lambda_{ss} C_{ss}(\boldsymbol{\tau}) \\ &\quad + \sum_{s,t}\left[\sum_{x_s} \lambda_{ts}(x_s) C_{ts}(x_s;\boldsymbol{\tau}) + \sum_{x_t} \lambda_{st}(x_t) C_{st}(x_t;\boldsymbol{\tau})\right]\end{aligned} \tag{22.42}$$

(The constraint that $\boldsymbol{\tau} \geq 0$ is not explicitly enforced, but one can show that it will hold at the optimum since $\boldsymbol{\theta} > 0$.) Some simple algebra then shows that $\nabla_{\boldsymbol{\tau}}\mathcal{L} = \mathbf{0}$ yields

$$\log \tau_s(x_s) = \lambda_{ss} + \theta_s(x_s) + \sum_{t\in\text{nbr}(s)} \lambda_{ts}(x_s) \tag{22.43}$$

$$\log \frac{\tau_{st}(x_s, x_t)}{\tilde{\tau}_s(x_s)\tilde{\tau}_t(x_t)} = \theta_{st}(x_s, x_t) - \lambda_{ts}(x_s) - \lambda_{st}(x_t) \tag{22.44}$$

where we have defined $\tilde{\tau}_s(x_s) \triangleq \sum_{x_t} \tau(x_s, x_t)$. Using the fact that the marginalization constraint implies $\tilde{\tau}_s(x_s) = \tau_s(x_s)$, we get

$$\begin{aligned}\log \tau_{st}(x_s, x_t) \quad &= \quad \lambda_{ss} + \lambda_{tt} + \theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t) \\ &\quad + \sum_{u\in\text{nbr}(s)\backslash t} \lambda_{us}(x_s) + \sum_{u\in\text{nbr}(t)\backslash s} \lambda_{ut}(x_t)\end{aligned} \tag{22.45}$$

To make the connection to message passing, define $M_{ts}(x_s) = \exp(\lambda_{ts}(x_s))$. With this notation, we can rewrite the above equations (after taking exponents of both sides) as follows:

$$\tau_s(x_s) \propto \exp(\theta_s(x_s)) \prod_{t\in\text{nbr}(s)} M_{ts}(x_s) \tag{22.46}$$

$$\begin{aligned}\tau_{st}(x_s, x_t) \quad &\propto \quad \exp\left(\theta_{st}(x_s, x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \\ &\quad \times \prod_{u\in\text{nbr}(s)\backslash t} M_{us}(x_s) \prod_{u\in\text{nbr}(t)\backslash s} M_{ut}(x_t)\end{aligned} \tag{22.47}$$

where the λ terms are absorbed into the constant of proportionality. We see that this is equivalent to the usual expression for the node and edge marginals in LBP.

To derive an equation for the messages in terms of other messages (rather than in terms of λ_{ts}), we enforce the marginalization condition $\sum_{x_t} \tau_{st}(x_s, x_t) = \tau_s(x_s)$. Then one can show that

$$M_{ts}(x_s) \propto \sum_{x_t} \left[\exp\left\{\theta_{st}(x_s, x_t) + \theta_t(x_t)\right\} \prod_{u \in \text{nbr}(t) \setminus s} M_{ut}(x_t) \right] \tag{22.48}$$

We see that this is equivalent to the usual expression for the messages in LBP.

22.3.6 Loopy BP vs mean field

It is interesting to compare the naive mean field (MF) and LBP approximations. There are several obvious differences. First, LBP is exact for trees whereas MF is not, suggesting LBP will in general be more accurate (see (Wainwright et al. 2003) for an analysis). Second, LBP optimizes over node and edge marginals, whereas MF only optimizes over node marginals, again suggesting LBP will be more accurate. Third, in the case that the true edge marginals factorize, so $\boldsymbol{\mu}_{st} = \boldsymbol{\mu}_s \boldsymbol{\mu}_t$, the free energy approximations will be the same in both cases.

What is less obvious, but which nevertheless seems to be true, is that the MF objective has many more local optima than the LBP objective, so optimizing the MF objective seems to be harder. In particular, (Weiss 2001), shows empirically that optimizing MF starting from uniform or random initial conditions often leads to poor results, whereas optimizing BP from uniform initial messages often leads to good results. Furthermore, initializing MF with the BP marginals also leads to good results (although MF tends to be more overconfident than BP), indicating that the problem is caused not by the inaccuracy of the MF approximation, but rather by the severe non-convexity of the MF objective, and by the weakness of the standard coordinate descent optimization method used by MF.[2] However, the advantage of MF is that it gives a lower bound on the partition function, unlike BP, which is useful when using it as a subroutine inside a learning algorithm. Also, MF is easier to extend to other distributions besides discrete and Gaussian, as we saw in Chapter 21. Intuitively, this is because MF only works with marginal distributions, which have a single type, rather than needing to define pairwise distributions, which may need to have two different types.

22.4 Extensions of belief propagation *

In this section, we discuss various extensions of LBP.

22.4.1 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**. The result is a hyper-graph, which is a graph

2. (Honkela et al. 2003) discusses the use of the pattern search algorithm to speedup mean field inference in the case of continuous random variables. It is possible that similar ideas could be adapted to the discrete case, although there may be no reason to do this, given that LBP already works well in the discrete case.

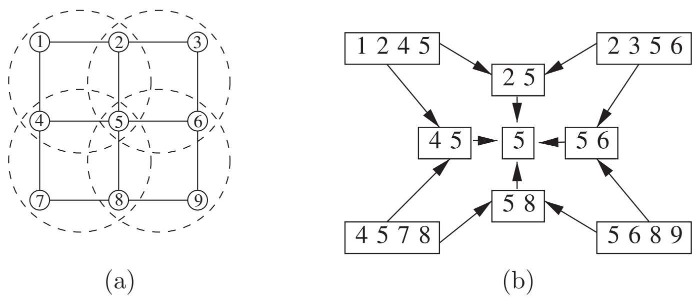


Figure 22.9 (a) Kikuchi clusters superimposed on a 3×3 lattice graph. (b) Corresponding hyper-graph. Source: Figure 4.5 of (Wainwright and Jordan 2008b). Used with kind permission of Martin Wainwright.

where there are hyper-edges between sets of vertices instead of between single vertices. Note that a junction tree (Section 20.4.1) is a kind of hyper-graph. We can represent hyper-graph using a poset (partially ordered set) diagram, where each node represents a hyper-edge, and there is an arrow $e_1 \rightarrow e_2$ if $e_2 \subset e_1$. See Figure 22.9 for an example.

Let t be the size of the largest hyper-edge in the hyper-graph. If we allow t 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.

Define $\mathbb{L}_t(G)$ to be the set of all pseudo-marginals such that normalization and marginalization constraints hold on a hyper-graph whose largest hyper-edge is of size $t+1$. For example, in Figure 22.9, we impose constraints of the form

$$\sum_{x_1,x_2} \tau_{1245}(x_1,x_2,x_4,x_5) = \tau_{45}(x_4,x_5), \quad \sum_{x_6} \tau_{56}(x_5,x_6) = \tau_5(x_5), \ldots \tag{22.49}$$

Furthermore, we approximate the entropy as follows:

$$\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq \sum_{g \in E} c(g) H_g(\tau_g) \tag{22.50}$$

where $H_g(\tau_g)$ is the entropy of the joint (pseudo) distribution on the vertices in set g, and $c(g)$ is called the **overcounting number** of set g. These are related to **Mobious numbers** in set theory. Rather than giving a precise definition, we just give a simple example. For the graph in Figure 22.9, we have

$$\begin{aligned}\mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) &= [H_{1245} + H_{2356} + H_{4578} + H_{5689}] \\ &\quad -[H_{25} + H_{45} + H_{56} + H_{58}] + H_5\end{aligned} \tag{22.51}$$

Putting these two approximations together, we can define the **Kikuchi free energy**[3] as follows:

$$F_{\text{Kikuchi}}(\boldsymbol{\tau}) \triangleq -\left[\boldsymbol{\theta}^T \boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau})\right] \tag{22.52}$$

3. Ryoichi Kikuchi is a Japanese physicist.

Our variational problem becomes

$$\min_{\boldsymbol{\tau}\in\mathbb{L}_t(G)} F_{\text{Kikuchi}}(\boldsymbol{\tau}) = \max_{\boldsymbol{\tau}\in\mathbb{L}_t(G)} \boldsymbol{\theta}^T\boldsymbol{\tau} + \mathbb{H}_{\text{Kikuchi}}(\boldsymbol{\tau}) \tag{22.53}$$

Just as with the Bethe free energy, this is not a concave objective. There are several possible algorithms for finding a local optimum of this objective, including a message passing algorithm known as **generalized belief propagation**. However, the details are beyond the scope of this chapter. See e.g., (Wainwright and Jordan 2008b, Sec 4.2) or (Koller and Friedman 2009, Sec 11.3.2) for more information. Suffice it to say that the method gives more accurate results than LBP, but at increased computational cost (because of the need to handle clusters of nodes). This cost, plus the complexity of the approach, have precluded it from widespread use.

22.4.2 Convex belief propagation

The mean field energy functional is concave, but it is maximized over a non-convex inner approximation to the marginal polytope. The Bethe and Kikuchi energy functionals are not concave, but they are maximized over a convex outer approximation to the marginal polytope. Consequently, for both MF and LBP, the optimization problem has multiple optima, so the methods are sensitive to the initial conditions. Given that the exact formulation (Equation 22.24) a concave objective maximized over a convex set, it is natural to try to come up with an appproximation which involves a concave objective being maximized over a convex set.

We now describe one method, known as **convex belief propagation**. This involves working with a set of tractable submodels, $\mathcal{F}$, such as trees or planar graphs. For each model $F \subset G$, the entropy is higher, $\mathbb{H}(\boldsymbol{\mu}(F)) \geq \mathbb{H}(\boldsymbol{\mu}(G))$, since F has fewer constraints. Consequently, any convex combination of such subgraphs will have higher entropy, too:

$$\mathbb{H}(\boldsymbol{\mu}(G)) \leq \sum_{F\in\mathcal{F}} \rho(F)\mathbb{H}(\boldsymbol{\mu}(F)) \triangleq \mathbb{H}(\boldsymbol{\mu},\rho) \tag{22.54}$$

where $\rho(F) \geq 0$ and $\sum_F \rho(F) = 1$. Furthermore, $\mathbb{H}(\boldsymbol{\mu},\rho)$ is a concave function of $\boldsymbol{\mu}$. We now define the convex free energy as

$$F_{\text{Convex}}(\boldsymbol{\mu},\rho) \triangleq -\left[\boldsymbol{\mu}^T\boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\mu},\rho)\right] \tag{22.55}$$

We define the concave energy functional as the negative of the convex free energy. We discuss how to optimize ρ below.

Having defined an upper bound on the entropy, we now consider a convex outerbound on the marginal polytope of mean parameters. We want to ensure we can evaluate the entropy of any vector $\boldsymbol{\tau}$ in this set, so we restrict it so that the projection of $\boldsymbol{\tau}$ onto the subgraph G lives in the projection of $\mathbb{M}$ onto F:

$$\mathbb{L}(G;\mathcal{F}) \triangleq \{\boldsymbol{\tau}\in\mathbb{R}^d : \boldsymbol{\tau}(F)\in\mathbb{M}(F)\ \forall F\in\mathcal{F}\} \tag{22.56}$$

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

$$\min_{\boldsymbol{\tau}\in\mathbb{L}(G;\mathcal{F})} F_{\text{Convex}}(\boldsymbol{\tau},\rho) = \max_{\boldsymbol{\tau}\in\mathbb{L}(G;\mathcal{F})} \boldsymbol{\tau}^T\boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\tau},\rho) \tag{22.57}$$

This is a concave objective being maximized over a convex set, and hence has a unique maximum. We give a specific example below.

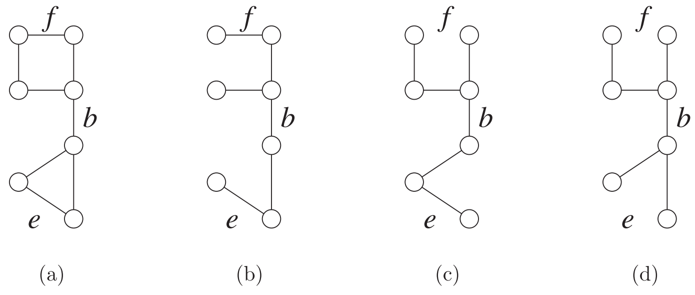


Figure 22.10 (a) A graph. (b-d) Some of its spanning trees. Source: Figure 7.1 of (Wainwright and Jordan 2008b). Used with kind permission of Martin Wainwright.

22.4.2.1 Tree-reweighted belief propagation

Consider the specific case where $\mathcal{F}$ is all spanning trees of a graph. For any given tree, the entropy is given by Equation 22.35. To compute the upper bound, obtained by averaging over all trees, note that the terms $\sum_F \rho(F)H(\mu(F)_s)$ for single nodes will just be H_s, since node s appears in every tree, and $\sum_F \rho(F) = 1$. But the mutual information term I_{st} receives weight $\rho_{st} = \mathbb{E}_\rho [\mathbb{I}((s,t) \in E(T))]$, known as the **edge appearance probability**. Hence we have the following upper bound on the entropy:

$$\mathbb{H}(\boldsymbol{\mu}) \leq \sum_{s \in V} H_s(\mu_s) - \sum_{(s,t) \in E} \rho_{st} I_{st}(\mu_{st}) \tag{22.58}$$

The edge appearance probabilities live in a space called the **spanning tree polytope**. This is because they are constrained to arise from a distribution over trees. Figure 22.10 gives an example of a graph and three of its spanning trees. Suppose each tree has equal weight under ρ. The edge f occurs in 1 of the 3 trees, so $\rho_f = 1/3$. The edge e occurs in 2 of the 3 trees, so $\rho_e = 2/3$. The edge b appears in all of the trees, so $\rho_b = 1$. And so on. Ideally we can find a distribution ρ, or equivalently edge probabilities in the spanning tree polytope, that make the above bound as tight as possible. An algorithm to do this is described in (Wainwright et al. 2005). (A simpler approach is to generate spanning trees of G at random until all edges are covered, or use all single edges with weight $\rho_e = 1/E$.)

What about the set we are optimizing over? We require $\boldsymbol{\mu}(T) \in \mathbb{M}(T)$ for each tree T, which means enforcing normalization and local consistency. Since we have to do this for every tree, we are enforcing normalization and local consistency on every edge. Hence $\mathbb{L}(G; \mathcal{F}) = \mathbb{L}(G)$. So our final optimization problem is as follows:

$$\max_{\boldsymbol{\tau} \in \mathbb{L}(G)} \left\{ \boldsymbol{\tau}^T \boldsymbol{\theta} + \sum_{s \in V} H_s(\tau_s) - \sum_{(s,t) \in E(G)} \rho_{st} I_{st}(\tau_{st}) \right\} \tag{22.59}$$

which is the same as the LBP objective except for the crucial ρ_{st} weights. So long as $\rho_{st} > 0$ for all edges (s,t), this problem is strictly concave with a unique maximum.

How can we find this global optimum? As for LBP, there are several algorithms, but perhaps the simplest is a modification of belief propagation known as **tree reweighted belief propagation**, also called **TRW** or **TRBP** for short. The message from t to s is now a function of all messages sent from other neighbors v to t, as before, but now it is also a function of the message sent from s to t. Specifically

$$M_{ts}(x_s) \quad \propto \quad \sum_{x_t} \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_t(x_t)\right) \frac{\prod_{v \in \text{nbr}(t)\backslash s}[M_{vt}(x_t)]^{\rho_{vt}}}{[M_{st}(x_t)]^{1-\rho_{ts}}} \tag{22.60}$$

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

$$\tau_s(x_s) \quad \propto \quad \exp(\theta_s(x_s)) \prod_{v\in\text{nbr}(s)} [M_{vs}(x_s)]^{\rho_{vs}} \tag{22.61}$$

$$\tau_{st}(x_s,x_t) \quad \propto \quad \varphi_{st}(x_s,x_t) \frac{\prod_{v\in\text{nbr}(s)\backslash t}[M_{vs}(x_s)]^{\rho_{vs}}}{[M_{ts}(x_s)]^{1-\rho_{st}}} \frac{\prod_{v\in\text{nbr}(t)\backslash s}[M_{vt}(x_t)]^{\rho_{vt}}}{[M_{st}(x_t)]^{1-\rho_{ts}}} \tag{22.62}$$

$$\varphi_{st}(x_s,x_t) \quad \triangleq \quad \exp\left(\frac{1}{\rho_{st}}\theta_{st}(x_s,x_t) + \theta_s(x_s) + \theta_t(x_t)\right) \tag{22.63}$$

This algorithm can be derived using a method similar to that described in Section 22.3.5.4.

If $\rho_{st} = 1$ for all edges $(s,t) \in E$, the algorithm reduces to the standard LBP algorithm. However, the condition $\rho_{st} = 1$ implies every edge is present in every spanning tree with probability 1, which is only possible if the original graph is a tree. Hence the method is only equivalent to standard LBP on trees, when the method is of course exact.

In general, this message passing scheme is not guaranteed to converge to the unique global optimum. One can devise double-loop methods that are guaranteed to converge (Hazan and Shashua 2008), but in practice, using damped updates as in Equation 22.7 is often sufficient to ensure convergence.

It is also possible to produce a convex version of the Kikuchi free energy, which one can optimize with a modified version of generalized belief propagation. See (Wainwright and Jordan 2008b, Sec 7.2.2) for details.

From Equation 22.59, and using the fact that the TRBP entropy approximation is an upper bound on the true entropy, wee see that the TRBP objective is an upper bound on $\log Z$. Using the fact that $I_{st} = H_s + H_t - H_{st}$, we can rewrite the upper bound as follows:

$$\log \hat{Z}(\boldsymbol{\theta}) \triangleq \boldsymbol{\tau}^T\boldsymbol{\theta} + \sum_{st} \rho_{st} H_{st}(\tau_{st}) + \sum_s c_s H_s(\tau_s) \le \log Z(\boldsymbol{\theta}) \tag{22.64}$$

where $c_s \triangleq 1 - \sum_t \rho_{st}$.

22.5 Expectation propagation

Expectation propagation (EP) (Minka 2001c) is a form of belief propagation where the messages are approximated. It is a generalization of the assumed density filtering (ADF) algorithm, discussed in Section 18.5.3. In that method, we approximated the posterior at each step using an assumed functional form, such as a Gaussian. This posterior can be computed using moment matching, which locally optimizes $\mathbb{KL}(p||q)$ for a single term. From this, we derived the message to send to the next time step.

ADF works well for sequential Bayesian updating, but the answer it gives depends on the order in which the data is seen. EP essentially corrects this flaw by making multiple passes over the data (thus EP is an offline or batch inference algorithm).

22.5.1 EP as a variational inference problem

We now explain how to view EP in terms of variational inference. We follow the presentation of (Wainwright and Jordan 2008b, Sec 4.3), which should be consulted for further details.

Suppose the joint distribution can be written in exponential family form as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) \propto f_0(\mathbf{x}) \exp(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) \prod_{i=1}^{d_I} \exp(\tilde{\boldsymbol{\theta}}_i^T \boldsymbol{\Phi}_i(\mathbf{x})) \tag{22.65}$$

where we have partitioned the parameters and the sufficient statistics into a tractable term $\boldsymbol{\theta}$ of size d_T and d_I intractable terms $\tilde{\boldsymbol{\theta}}_i$, each of size b.

For example, consider the problem of inferring an unknown vector $\mathbf{x}$, when the observation model is a mixture of two Gaussians, one centered at $\mathbf{x}$ and one centered at $\mathbf{0}$. (This can be used to represent outliers, for example.) Minka (who invented EP) calls this the **clutter problem**. More formally, we assume an observation model of the form

$$p(\mathbf{y}|\mathbf{x}) = (1-w)\mathcal{N}(\mathbf{y}|\mathbf{x}, \mathbf{I}) + w\mathcal{N}(\mathbf{y}|\mathbf{0}, a\mathbf{I}) \tag{22.66}$$

where $0 < w < 1$ is the known mixing weight (fraction of outliers), and $a > 0$ is the variance of the background distribution. Assuming a fixed prior of the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{0}, \boldsymbol{\Sigma})$, we can write our model in the required form as follows:

$$p(\mathbf{x}|\mathbf{y}_{1:N}) \propto \mathcal{N}(\mathbf{x}|\mathbf{0}, \boldsymbol{\Sigma}) \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}) \tag{22.67}$$

$$= \exp\left(-\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}\right) \exp\left(\sum_{i=1}^{N} \log p(\mathbf{y}_i|\mathbf{x})\right) \tag{22.68}$$

This matches our canonical form where $f_0(\mathbf{x}) \exp(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$ corresponds to $\exp\left(-\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}^{-1}\mathbf{x}\right)$, using $\boldsymbol{\phi}(\mathbf{x}) = (\mathbf{x}, \mathbf{x}\mathbf{x}^T)$, and we set $\boldsymbol{\Phi}_i(\mathbf{x}) = \log p(\mathbf{y}_i|\mathbf{x})$, $\tilde{\boldsymbol{\theta}}_i = 1$, and $d_I = N$.

The exact inference problem corresponds to

$$\max_{(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) \in \mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi})} \boldsymbol{\tau}^T \boldsymbol{\theta} + \tilde{\boldsymbol{\tau}}^T \tilde{\boldsymbol{\theta}} + \mathbb{H}\left((\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}})\right) \tag{22.69}$$

where $\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi})$ is the set of mean parameters realizable by any probability distribution as seen through the eyes of the sufficient statistics:

$$\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi}) = \{(\boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}) \in \mathbb{R}^{d_T} \times \mathbb{R}^{d_I b} : (\boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}) = \mathbb{E}\left[(\boldsymbol{\phi}(\mathbf{X}), \boldsymbol{\Phi}_1(\mathbf{X}), \ldots, \boldsymbol{\Phi}_{d_I}(\mathbf{X}))\right]\} \tag{22.70}$$

As it stands, it is intractable to perform inference in this distribution. For example, in our clutter example, the posterior contains 2^N modes. But suppose we incorporate just one of the intractable terms, say the i'th one; we will call this the $\boldsymbol{\Phi}_i$-augmented distribution:

$$p(\mathbf{x}|\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}_i) \propto f_0(\mathbf{x}) \exp(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) \exp(\tilde{\boldsymbol{\theta}}_i^T \boldsymbol{\Phi}_i(\mathbf{x})) \tag{22.71}$$

In our clutter example, this becomes

$$p(\mathbf{x}|\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}_i) = \exp\left(-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}\right)[w\mathcal{N}(\mathbf{y}_i|\mathbf{0}, a\mathbf{I}) + (1-w)\mathcal{N}(\mathbf{y}_i|\mathbf{x}, \mathbf{I})] \tag{22.72}$$

This *is* tractable to compute, since it is just a mixture of 2 Gaussians.

The key idea behind EP is to work with these the $\boldsymbol{\Phi}_i$-augmented distributions in an iterative fashion. First, we approximate the convex set $\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi})$ with another, larger convex set:

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\Phi}) \triangleq \{(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) : \boldsymbol{\tau} \in \mathcal{M}(\boldsymbol{\phi}), (\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}_i) \in \mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi}_i)\} \tag{22.73}$$

where $\mathcal{M}(\boldsymbol{\phi}) = \{\boldsymbol{\mu} \in \mathbb{R}^{d_T} : \boldsymbol{\mu} = \mathbb{E}[\boldsymbol{\phi}(\mathbf{X})]\}$ and $\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi}_i) = \{(\boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}_i) \in \mathbb{R}^{d_T} \times \mathbb{R}^b : (\boldsymbol{\mu}, \tilde{\boldsymbol{\mu}}_i) = \mathbb{E}[(\boldsymbol{\phi}(\mathbf{X}), \boldsymbol{\Phi}_i(\mathbf{X}))]$. Next we approximate the entropy by the following term-by-term approximation:

$$\mathbb{H}_{\text{ep}}(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) \triangleq \mathbb{H}(\boldsymbol{\tau}) + \sum_{i=1}^{d_I}[\mathbb{H}(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}_i) - \mathbb{H}(\boldsymbol{\tau})] \tag{22.74}$$

Then the EP problem becomes

$$\max_{(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) \in \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\Phi})} \boldsymbol{\tau}^T\boldsymbol{\theta} + \tilde{\boldsymbol{\tau}}^T\tilde{\boldsymbol{\theta}} + \mathbb{H}_{\text{ep}}(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) \tag{22.75}$$

22.5.2 Optimizing the EP objective using moment matching

We now discuss how to maximize the EP objective in Equation 22.75. Let us duplicate $\boldsymbol{\tau}$ d_I times to yield $\boldsymbol{\eta}_i = \boldsymbol{\tau}$. The augmented set of parameters we need to optimize is now

$$(\boldsymbol{\tau}, (\boldsymbol{\eta}_i, \tilde{\boldsymbol{\tau}}_i)_{i=1}^{d_I}) \in \mathbb{R}^{d_T} \times (\mathbb{R}^{d_T} \times \mathbb{R}^b)^{d_I} \tag{22.76}$$

subject to the constraints that $\boldsymbol{\eta}_i = \boldsymbol{\tau}$ and $(\boldsymbol{\eta}_i, \tilde{\boldsymbol{\tau}}_i) \in \mathcal{M}(\boldsymbol{\phi}; \boldsymbol{\Phi}_i)$. Let us associate a vector of Lagrange multipliers $\boldsymbol{\lambda}_i \in \mathbb{R}^{d_T}$ with the first set of constraints. Then the partial Lagrangian becomes

$$L(\boldsymbol{\tau}; \boldsymbol{\lambda}) = \boldsymbol{\tau}^T\boldsymbol{\theta} + \mathbb{H}(\boldsymbol{\tau}) + \sum_{i=1}^{d_i}\left[\tilde{\boldsymbol{\tau}}_i^T\tilde{\boldsymbol{\theta}}_i + \mathbb{H}((\boldsymbol{\eta}_i, \tilde{\boldsymbol{\tau}}_i)) - \mathbb{H}(\boldsymbol{\eta}_i) + \boldsymbol{\lambda}_i^T(\boldsymbol{\tau} - \boldsymbol{\eta}_i)\right] \tag{22.77}$$

By solving $\nabla_{\boldsymbol{\tau}} L(\boldsymbol{\tau}; \boldsymbol{\lambda}) = \mathbf{0}$, we can show that the corresponding distribution in $\mathcal{M}(\boldsymbol{\phi})$ has the form

$$q(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\lambda}) \propto f_0(\mathbf{x})\exp\{(\boldsymbol{\theta} + \sum_{i=1}^{d_I}\boldsymbol{\lambda}_i)^T\boldsymbol{\phi}(\mathbf{x})\} \tag{22.78}$$

The $\boldsymbol{\lambda}_i^T\boldsymbol{\phi}(\mathbf{x})$ terms represents an approximation to the i'th intractable term using the sufficient statistics from the base distribution, as we will see below. Similarly, by solving $\nabla_{(\boldsymbol{\eta}_i, \tilde{\boldsymbol{\tau}}_i)} L(\boldsymbol{\tau}; \boldsymbol{\lambda}) = \mathbf{0}$, we find that the corresponding distribution in $\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi}_i)$ has the form

$$q_i(\mathbf{x}|\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}_i, \boldsymbol{\lambda}) \propto f_0(\mathbf{x})\exp\{(\boldsymbol{\theta} + \sum_{j \neq i}\boldsymbol{\lambda}_j)^T\boldsymbol{\phi}(\mathbf{x}) + \tilde{\boldsymbol{\theta}}_i^T\boldsymbol{\Phi}_i(\mathbf{x})\} \tag{22.79}$$

This corresponds to removing the approximation to the i'th term, $\boldsymbol{\lambda}_i$, from the base distribution, and adding in the correct i'th term, $\boldsymbol{\Phi}_i$. Finally, $\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{\tau}; \boldsymbol{\lambda}) = \mathbf{0}$ just enforces the constraints that $\boldsymbol{\tau} = \mathbb{E}_q [\boldsymbol{\phi}(\mathbf{X})]$ and $\boldsymbol{\eta}_i = \mathbb{E}_{q_i} [\boldsymbol{\phi}(\mathbf{X})]$ are equal. In other words, we get the following moment matching constraints:

$$\int q(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\lambda})\boldsymbol{\phi}(\mathbf{x})d\mathbf{x} = \int q_i(\mathbf{x}|\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}_i, \boldsymbol{\lambda})\boldsymbol{\phi}(\mathbf{x})d\mathbf{x} \tag{22.80}$$

Thus the overall algorithm is as follows. First we initialize the $\boldsymbol{\lambda}_i$. Then we iterate the following to convergence: pick a term i; compute q_i (corresponding to removing the old approximation to $\boldsymbol{\Phi}_i$ and adding in the new one); then update the $\boldsymbol{\lambda}_i$ term in q by solving the moment matching equation $\mathbb{E}_{q_i} [\boldsymbol{\phi}(\mathbf{X})] = \mathbb{E}_q [\boldsymbol{\phi}(\mathbf{X})]$. (Note that this particular optimization scheme is not guaranteed to converge to a fixed point.)

An equivalent way of stating the algorithm is as follows. Let us assume the true distribution is given by

$$p(\mathbf{x}|\mathcal{D}) = \frac{1}{Z}\prod_i f_i(\mathbf{x}) \tag{22.81}$$

We approximate each f_i by $\tilde{f}_i$ and set

$$q(\mathbf{x}) = \frac{1}{Z}\prod_i \tilde{f}_i(\mathbf{x}) \tag{22.82}$$

Now we repeat the following until convergence:

1. Choose a factor $\tilde{f}_i$ to refine.
2. Remove $\tilde{f}_i$ from the posterior by dividing it out:

   $$q_{-i}(\mathbf{x}) = \frac{q(\mathbf{x})}{\tilde{f}_i(\mathbf{x})} \tag{22.83}$$

   This can be implemented by substracting off the natural parameters of $\tilde{f}_i$ from q.
3. Compute the new posterior $q^{new}(\mathbf{x})$ by solving

   $$\min_{q^{new}(\mathbf{x})} \mathbb{KL}\left(\frac{1}{Z_i} f_i(\mathbf{x})q_{-i}(\mathbf{x})||q^{new}(\mathbf{x})\right) \tag{22.84}$$

   This can be done by equating the moments of $q^{new}(\mathbf{x})$ with those of $q_i(\mathbf{x}) \propto q_{-i}(\mathbf{x})f_i(\mathbf{x})$. The corresponding normalization constant has the form

   $$Z_i = \int q_{-i}(\mathbf{x})f_i(\mathbf{x})d\mathbf{x} \tag{22.85}$$

4. Compute the new factor (message) that was implicitly used (so it can be later removed):

   $$\tilde{f}_i(\mathbf{x}) = Z_i \frac{q^{new}(\mathbf{x})}{q_{-i}(\mathbf{x})} \tag{22.86}$$

After convergence, we can approximate the marginal likelihood using

$$p(\mathcal{D}) \approx \int \prod_i \tilde{f}_i(\mathbf{x}) d\mathbf{x} \tag{22.87}$$

We will give some examples of this below which will make things clearer.

22.5.3 EP for the clutter problem

Let us return to considering the clutter problem. Our presentation is based on (Bishop 2006b).[4] For simplicity, we will assume that the prior is a spherical Gaussian, $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, b\mathbf{I})$. Also, we choose to approximate the posterior by a spherical Gaussian, $q(\mathbf{x}) = \mathcal{N}(\mathbf{m}, v\mathbf{I})$. We set $f_0(\mathbf{x})$ to be the prior; this can be held fixed. The factor approximations will be "Gaussian like" terms of the form

$$\tilde{f}_i(\mathbf{x}) = s_i \mathcal{N}(\mathbf{x}|\mathbf{m}_i, v_i\mathbf{I}) \tag{22.88}$$

Note, however, that in the EP updates, the variances may be negative! Thus these terms should be interpreted as functions, but not necessarily probability distributions. (If the variance is negative, it means the that $\tilde{f}_i$ curves upwards instead of downwards.)

First we remove $\tilde{f}_i(\mathbf{x})$ from $q(\mathbf{x})$ by division, which yields $q_{-i}(\mathbf{x}) = \mathcal{N}(\mathbf{m}_{-i}, v_{-i}\mathbf{I})$, where

$$v_{-i}^{-1} = v^{-1} - v_i^{-1} \tag{22.89}$$

$$\mathbf{m}_{-i} = \mathbf{m} + v_{-i}v_i^{-1}(\mathbf{m} - \mathbf{m}_i) \tag{22.90}$$

The normalization constant is given by

$$Z_i = (1-w)\mathcal{N}(\mathbf{y}_i|\mathbf{m}_{-i}, (v_{-i}+1)\mathbf{I}) + w\mathcal{N}(\mathbf{y}_i|\mathbf{0}, a\mathbf{I}) \tag{22.91}$$

Next we compute $q^{new}(\mathbf{x})$ by computing the mean and variance of $q{-}i(\mathbf{x})f_i(\mathbf{x})$ as follows:

$$\mathbf{m} = \mathbf{m}_{-i} + \rho_i \frac{v_{-i}}{v_{-i}+1}(\mathbf{y}_i - \mathbf{m}_{-i}) \tag{22.92}$$

$$v = v_{-i} - \rho_i \frac{v_{-i}^2}{v_{-i}+1} + \rho_i(1-\rho_i)\frac{v_{-i}^2||\mathbf{y}_i - \mathbf{m}_i||^2}{D(v_{-i}+1)^2} \tag{22.93}$$

$$\rho_i = 1 - \frac{w}{Z_i}\mathcal{N}(\mathbf{y}_i|\mathbf{0}, a\mathbf{I}) \tag{22.94}$$

where D is the dimensionality of $\mathbf{x}$ and ρ_i can be interpreted as the probability that $\mathbf{y}_i$ is not clutter.

Finally, we compute the new factor $\tilde{f}_i$ whose parameters are given by

$$v_i^{-1} = v^{-1} - v_{-i}^{-1} \tag{22.95}$$

$$\mathbf{m}_i = \mathbf{m}_{-i} + (v_i + v_{-i})v_{-i}^{-1}(\mathbf{m} - \mathbf{m}_{-i}) \tag{22.96}$$

$$s_i = \frac{Z_i}{(2\pi v_i)^{D/2}\mathcal{N}(\mathbf{m}_i|\mathbf{m}_{-i}, (v_i+v_{-i})\mathbf{I})} \tag{22.97}$$

4. For a handy "crib sheet", containing many of the standard equations needed for deriving Gaussian EP algorithms, see `http://research.microsoft.com/en-us/um/people/minka/papers/ep/minka-ep-quickref.pdf`.

At convergence, we can approximate the marginal likelihood as follows:

$$p(\mathcal{D}) \approx (2\pi v)^{D/2} \exp(c/2) \prod_{i=1}^{N} s_i (2\pi v_i)^{-D/2} \tag{22.98}$$

$$c \triangleq \frac{\mathbf{m}^T\mathbf{m}}{v} - \sum_{i=1}^{N} \frac{\mathbf{m}_i^T \mathbf{m}_i}{v_i} \tag{22.99}$$

In (Minka 2001d), it is shown that, at least on this example, EP gives better accuracy per unit of CPU time than VB and MCMC.

22.5.4 LBP is a special case of EP

We now show that loopy belief propagation is a special case of EP, where the base distribution contains the node marginals and the "intractable" terms correspond to the edge potentials. We assume the model has the pairwise form shown in Equation 22.12. If there are m nodes, the base distribution takes the form

$$p(\mathbf{x}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m, \mathbf{0}) \propto \prod_{s \in V} \exp(\theta_s(x_s)) \tag{22.100}$$

The entropy of this distribution is simply

$$\mathbb{H}(\boldsymbol{\tau}_{1:m}) = \sum_s \mathbb{H}(\boldsymbol{\tau}_s) \tag{22.101}$$

If we add in the $u - v$ edge, the $\boldsymbol{\Phi}_{uv}$ augmented distribution has the form

$$p(\mathbf{x}|\boldsymbol{\theta}_{1:m}, \boldsymbol{\theta}_{uv}) \propto \left[\prod_{s \in V} \exp(\theta_s(x_s))\right] \exp(\theta_{uv}(x_u, x_v)) \tag{22.102}$$

Since this graph is a tree, the exact entropy of this distribution is given by

$$\mathbb{H}(\boldsymbol{\tau}_{1:m}, \tilde{\boldsymbol{\tau}}_{uv}) = \sum_s \mathbb{H}(\boldsymbol{\tau}_s) - I(\tilde{\boldsymbol{\tau}}_{uv}) \tag{22.103}$$

where $I(\boldsymbol{\tau}_{uv}) = \mathbb{H}(\boldsymbol{\tau}_u) + \mathbb{H}(\boldsymbol{\tau}_v) - \mathbb{H}(\boldsymbol{\tau}_{uv})$ is the mutual information. Thus the EP approximation to the entropy of the full distribution is given by

$$\begin{aligned}
\mathbb{H}_{\text{ep}}(\boldsymbol{\tau}, \tilde{\boldsymbol{\tau}}) &= \mathbb{H}(\boldsymbol{\tau}) + \sum_{(u,v)\in E} \left[\mathbb{H}(\boldsymbol{\tau}_{1:m}, \tilde{\boldsymbol{\tau}}_{uv}) - \mathbb{H}(\boldsymbol{\tau})\right] && (22.104)\\
&= \sum_s \mathbb{H}(\boldsymbol{\tau}_s) + \sum_{(u,v)\in E} \left[\sum_s \mathbb{H}(\boldsymbol{\tau}_s) - I(\tilde{\boldsymbol{\tau}}_{uv}) - \sum_s \mathbb{H}(\boldsymbol{\tau}_s)\right] && (22.105)\\
&= \sum_s \mathbb{H}(\boldsymbol{\tau}_s) - \sum_{(u,v)\in E} I(\tilde{\boldsymbol{\tau}}_{uv}) && (22.106)
\end{aligned}$$

which is precisely the Bethe approximation to the entropy.

We now show that the convex set that EP is optimizing over, $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\Phi})$ given by Equation 22.73, is the same as the one that LBP is optimizing over, $\mathbb{L}(G)$ given in Equation 22.33. First, let us consider the set $\mathcal{M}(\boldsymbol{\phi})$. This consists of all marginal distributions $(\boldsymbol{\tau}_s, s \in V)$, realizable by a factored distribution. This is therefore equivalent to the set of all distributions which satisfy non-negativity $\tau_s(x_s) \geq 0$ and the local normalization constraint $\sum_{x_s} \tau(x_s) = 1$. Now consider the set $\mathcal{M}(\boldsymbol{\phi}, \boldsymbol{\Phi}_{uv})$ for a single $u-v$ edge. This is equivalent to the marginal polytope $\mathbb{M}(G_{uv})$, where G_{uv} is the graph with the single $u - v$ edge added. Since this graph corresponds to a tree, this set also satisfies the marginalization conditions

$$\sum_{x_v} \tau_{uv}(x_u, x_v) = \tau_u(x_u), \; \sum_{x_u} \tau_{uv}(x_u, x_v) = \tau_v(x_v) \tag{22.107}$$

Since $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\Phi})$ is the union of such sets, as we sweep over all edges in the graph, we recover the same set as $\mathbb{L}(G)$.

We have shown that the Bethe approximation is equivalent to the EP approximation. We now show how the EP algorithm reduces to LBP. Associated with each intractable term $i = (u, v)$ will be a pair of Lagrange multipliers, $(\lambda_{uv}(x_v), \lambda_{vu}(x_u))$. Recalling that $\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) = [\theta_s(x_s)]_s$, the base distribution in Equation 22.78 has the form

$$q(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\lambda}) \propto \prod_s \exp(\theta_s(x_s)) \prod_{(u,v)\in E} \exp(\lambda_{uv}(x_v) + \lambda_{vu}(x_u)) \tag{22.108}$$

$$= \prod_s \exp\left(\theta_s(x_s) + \sum_{t \in \mathcal{N}(s)} \lambda_{ts}(x_s)\right) \tag{22.109}$$

Similarly, the augmented distribution in Equation 22.79 has the form

$$q_{uv}(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\lambda}) \propto q(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\lambda}) \exp\left(\theta_{uv}(x_u, x_v) - \lambda_{uv}(x_v) - \lambda_{vu}(x_u)\right) \tag{22.110}$$

We now need to update $\tau_u(x_u)$ and $\tau_v(x_v)$ to enforce the moment matching constraints:

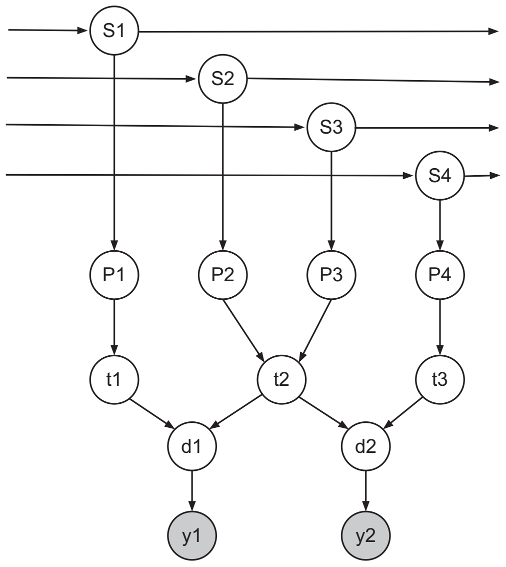
$$\left(\mathbb{E}_q[x_s], \mathbb{E}_q[x_t]\right) = \left(\mathbb{E}_{q_{uv}}[x_s], \mathbb{E}_{q_{uv}}[x_t]\right) \tag{22.111}$$

It can be shown that this can be done by performing the usual sum-product message passing step along the $u - v$ edge (in both directions), where the messages are given by $M_{uv}(x_v) = \exp(\lambda_{uv}(x_v))$, and $M_{vu}(x_u) = \exp(\lambda_{vu}(x_u))$. Once we have updated q, we can derive the corresponding messages λ_{uv} and λ_{vu}.

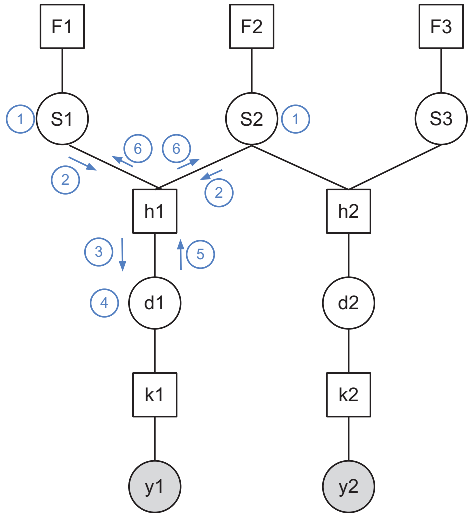
The above analysis suggests a natural extension, where we make the base distribution be a tree structure instead of a fully factored distribution. We then add in one edge at a time, absorb its effect, and approximate the resulting distribution by a new tree. This is known as **tree EP** (Minka and Qi 2003), and is more accurate than LBP, and sometimes faster. By considering other kinds of structured base distributions, we can derive algorothms that outperform generalization belief propagation (Welling et al. 2005).

22.5.5 Ranking players using TrueSkill

We now present an interesting application of EP to the problem of ranking players who compete in games. Microsoft uses this method — known as **TrueSkill** (Herbrich et al. 2007) — to rank



(a)



(b)

Figure 22.11 (a) A DGM representing the TrueSkill model for 4 players and 3 teams, where team 1 is player 1, team 2 is players 2 and 3, and team 3 is player 4. We assume there are two games, team 1 vs team 2, and team 2 vs team 3. Nodes with double circles are deterministic. (b) A factor graph representation of the model where we assume there are 3 players (and no teams). There are 2 games, player 1 vs player 2, and player 2 vs player 3. The numbers inside circles represent steps in the message passing algorithm.

players who use the Xbox 360 Live online gaming system; this system process over 10^5 games per day, making this one of the largest application of Bayesian statistics to date.[5] The same method can also be applied to other games, such as tennis or chess.[6]

The basic idea is shown in Figure 22.11(a). We assume each player i has a latent or true underlying skill level $s_i \in \mathbb{R}$. These skill levels can evolve over time according to a simple dynamical model, $p(s_i^t|s_i^{t-1}) = \mathcal{N}(s_i^t|s_i^{t-1}, \gamma^2)$. In any given game, we define the performance of player i to be p_i, which has the conditional distribution $p(p_i|s_i) = \mathcal{N}(p_i|s_i, \beta^2)$. We then define the performance of a team to be the sum of the performance of its constituent players. For example, in Figure 22.11(a), we assume team 2 is composed of players 2 and 3, so we define $t_2 = p_2 + p_3$. Finally, we assume that the outcome of a game depends on the difference in performance levels of the two teams. For example, in Figure 22.11(a), we assume $y_1 = \text{sign}(d_1)$, where $d_1 = t_1 - t_2$, and where $y_1 = +1$ means team 1 won, and $y_1 = -1$ means team 2 won. Thus the prior probability that team 1 wins is

$$p(y_1 = +1|\mathbf{s}) = \int p(d_1 > 0|t_1, t_2)p(t_1|s_1)p(t_2|s_2)dt_1 dt_2 \tag{22.112}$$

where $t_1 \sim \mathcal{N}(s_1, \beta^2)$ and $t_2 \sim \mathcal{N}(s_2 + s_3, \beta^2)$.[7]

To simplify the presentation of the algorithm, we will ignore the dynamical model and assume a common static factored Gaussian prior, $\mathcal{N}(\mu_0, \sigma_0^2)$, on the skills. Also, we will assume that each team consists of 1 player, so $t_i = p_i$, and that there can be no ties. Finally, we will integrate out the performance variables p_i, and assume $\beta^2 = 1$, leading to a final model of the form

$$\begin{aligned} p(\mathbf{s}) &= \prod_i \mathcal{N}(s_i|\mu_0, \sigma^2) && (22.113) \\ p(d_g|\mathbf{s}) &= \mathcal{N}(d_g|s_{i_g} - s_{j_g}, 1) && (22.114) \\ p(y_g|d_g) &= \mathbb{I}(y_g = \text{sign}(d_g)) && (22.115) \end{aligned}$$

where i_g is the first player of game g, and j_g is the second player. This is represented in factor graph form in in Figure 22.11(b). We have 3 kinds of factors: the prior factor, $f_i(s_i) = \mathcal{N}(s_i|\mu_0, \sigma_0^2)$, the game factor, $h_g(s_{i_g}, s_{j_g}, d_g) = \mathcal{N}(d_g|s_{i_g} - s_{j_g}, 1)$, and the outcome factor, $k_g(d_g, y_g) = \mathbb{I}(y_g = \text{sign}(d_g))$.

Since the likelihood term $(y_g|d_g)$ is not conjugate to the Gaussian priors, we will have to perform approximate inference. Thus even when the graph is a tree, we will need to iterate. (If there were an additional game, say between player 1 and player 3, then the graph would no longer be a tree.) We will represent all messages and marginal beliefs by 1d Gaussians. We will use the notation μ and v for the mean and variance (the moment parameters), and $\lambda = 1/v$ and $\eta = \lambda\mu$ for the precision and precision-adjusted mean (the natural parameters).

5. Naive Bayes classifiers, which are widely used in spam filters, are often described as the most common application of Bayesian methods. However, the parameters of such models are usually fit using non-Bayesian methods, such as penalized maximum likelihood.

6. Our presentation of this algorithm is based in part on lecture notes by Carl Rasmussen Joaquin Quinonero-Candela, available at `http://mlg.eng.cam.ac.uk/teaching/4f13/1112/lect13.pdf`.

7. Note that this is very similar to probit regression, discussed in Section 9.4, except the inputs are (the differences of) latent 1 dimensional factors. If we assume a logistic noise model instead of a Gaussian noise model, we recover the Bradley Terry model of ranking.

We initialize by assuming that at iteration 0, the initial upward messages from factors h_g to variables s_i are uniform, i.e.,

$$m^0_{h_g \to s_{i_g}}(s_{i_g}) = 1,\ \lambda^0_{h_g \to s_{i_g}} = 0,\ \eta^0_{h_g \to s_{i_g}} = 0 \tag{22.116}$$

and similarly $m^0_{h_g \to s_{j_g}}(s_{j_g}) = 1$. The messages passing algorithm consists of 6 steps per game, as illustrated in Figure 22.11(b). We give the details of these steps below.

1. Compute the posterior over the skills variables:

$$q^t(s_i) = f(s_i) \prod_g m^{t-1}_{h_g \to s_i}(s_i) = \mathcal{N}_c(s_i|\eta^t_i, \lambda^t_i) \tag{22.117}$$

$$\lambda^t_i = \lambda_0 + \sum_g \lambda^{t-1}_{h_g \to s_i},\quad \eta^t_i = \eta_0 + \sum_g \eta^{t-1}_{h_g \to s_i} \tag{22.118}$$

2. Compute the message from the skills variables down to the game factor h_g:

$$m^t_{s_{i_g} \to h_g}(s_{i_g}) = \frac{q^t(s_{i_g})}{m^t_{h_g \to s_{i_g}}(s_{i_g})},\quad m^t_{s_{j_g} \to h_g}(s_{j_g}) = \frac{q^t(s_{j_g})}{m^t_{h_g \to s_{j_g}}(s_{j_g})} \tag{22.119}$$

where the division is implemented by subtracting the natural parameters as follows:

$$\lambda^t_{s_{i_g} \to h_g} = \lambda^t_{s_{i_g}} - \lambda^t_{h_g \to s_{i_g}},\quad \eta^t_{s_{i_g} \to h_g} = \eta^t_{s_{i_g}} - \eta^t_{h_g \to s_{i_g}} \tag{22.120}$$

and similarly for s_{j_g}.

3. Compute the message from the game factor h_g down to the difference variable d_g:

$$m^t_{h_g \to d_g}(d_g) = \int\int h_g(d_g, s_{i_g}, s_{j_g}) m^t_{s_{i_g} \to h_g}(s_{i_g}) m^t_{s_{j_g} \to h_g}(s_{j_g}) ds_{i_g} ds_{j_g} \tag{22.121}$$

$$= \int\int \mathcal{N}(d_g|s_{i_g} - s_{j_g}, 1)\mathcal{N}(s_{i_g}|\mu^t_{s_{i_g} \to h_g}, v^t_{s_{i_g} \to h_g}) \tag{22.122}$$

$$\mathcal{N}(s_{j_g}|\mu^t_{s_{j_g} \to h_g}, v^t_{s_{j_g} \to h_g}) ds_{i_g} ds_{j_g} \tag{22.123}$$

$$= \mathcal{N}(d_g|\mu^t_{h_g \to d_g}, v^t_{h_g \to d_g}) \tag{22.124}$$

$$v^t_{h_g \to d_g} = 1 + v^t_{s_{i_g} \to h_g} + v^t_{s_{j_g} \to h_g} \tag{22.125}$$

$$\mu^t_{h_g \to d_g} = \mu^t_{s_{i_g} \to h_g} - \mu^t_{s_{j_g} \to h_g} \tag{22.126}$$

4. Compute the posterior over the difference variables:

$$q^t(d_g) \propto m^t_{h_g \to d_g}(d_g) m_{k_g \to d_g}(d_g) \tag{22.127}$$

$$= \mathcal{N}(d_g|\mu^t_{h_g \to d_g}, v^t_{h_g \to d_g}) \mathbb{I}(y_g = \text{sign}(d_g)) \tag{22.128}$$

$$\approx \mathcal{N}(d_g|\mu^t_g, v^t_g) \tag{22.129}$$

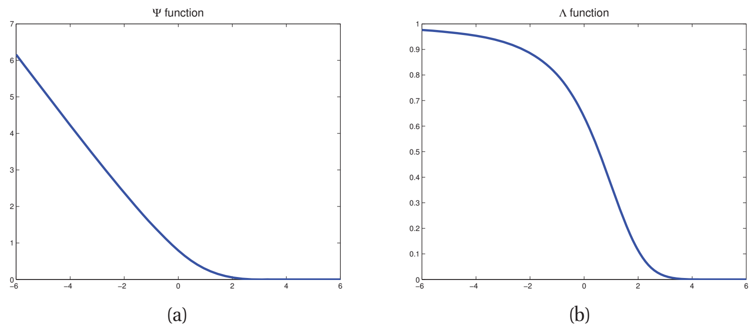


Figure 22.12 (a) Ψ function. (b) Λ function. Based on Figure 2 of (Herbrich et al. 2007). Figure generated by `trueskillPlot`.

(Note that the upward message from the k_g factor is constant.) We can find these parameters by moment matching as follows:

$$\mu_g^t = y_g \mu_{h_g \to d_g}^t + \sigma_{h_g \to d_g}^t \Psi\left(\frac{y_g \mu_{h_g \to d_g}^t}{\sigma_{h_g \to d_g}^t}\right) \tag{22.130}$$

$$v_g^t = v_{h_g \to d_g}^t \left[1 - \Lambda\left(\frac{y_g \mu_{h_g \to d_g}^t}{\sigma_{h_g \to d_g}^t}\right)\right] \tag{22.131}$$

$$\Psi(x) \triangleq \frac{\mathcal{N}(x|0,1)}{\Phi(x)} \tag{22.132}$$

$$\Lambda(x) \triangleq \Psi(x)(\Psi(x) + x) \tag{22.133}$$

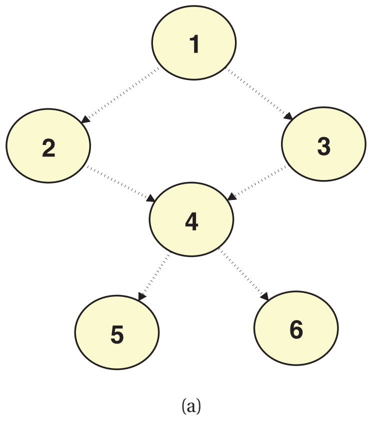
(The derivation of these equations is left as a modification to Exercise 11.15.) These functions are plotted in Figure 22.12. Let us try to understand these equations. Suppose $\mu_{h_g \to d_g}^t$ is a large positive number. That means we expect, based on the current estimate of the skills, that d_g will be large and positive. Consequently, if we observe $y_g = +1$, we will not be surprised that i_g is the winner, which is reflected in the fact that the update factor for the mean is small, $\Psi(y_g \mu_{h_g \to d_g}^t) \approx 0$. Similarly, the update factor for the variance is small, $\Lambda(y_g \mu_{h_g \to d_g}^t) \approx 0$. However, if we observe $y_g = -1$, then the update factor for the mean and variance becomes quite large.

5. Compute the upward message from the difference variable to the game factor h_g:

$$m_{d_g \to h_g}^t(d_g) = \frac{q^t(d_g)}{m_{d_g \to h_g}^t(d_g)} \tag{22.134}$$

$$\lambda_{d_g \to h_h}^t = \lambda_g^t - \lambda_{h_g \to d_g}^t, \quad \eta_{d_g \to h_h}^t = \eta_g^t - \eta_{h_g \to d_g}^t \tag{22.135}$$

6. Compute the upward messages from the game factor to the skill variables. Let us assume



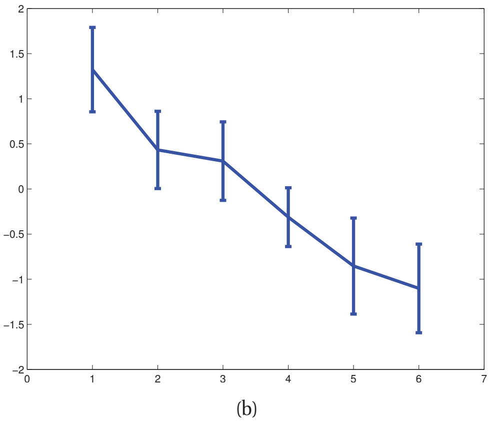


Figure 22.13 (a) A DAG representing a partial ordering of players. (b) Posterior mean plus/minus 1 standard deviation for the latent skills of each player based on 26 games. Figure generated by `trueskillDemo`.

that i_g is the winner, and j_g is the loser. Then we have

$$m^t_{h_g \to s_{i_g}}(s_{i_g}) = \int\int h_g(d_g, s_{i_g}, s_{j_g}) m^t_{d_g \to h_g}(d_g) m^t_{s_{j_g} \to h_g}(s_{j_g}) dd_g ds_{j_g} \tag{22.136}$$

$$= \mathcal{N}(s_{i_g} | \mu^t_{h_g \to s_{i_g}}, v^t_{h_g \to s_{i_g}}) \tag{22.137}$$

$$v^t_{h_g \to s_{i_g}} = 1 + v^t_{d_g \to h_g} + v^t_{s_{j_g} \to h_g} \tag{22.138}$$

$$\mu^t_{h_g \to s_{i_g}} = \mu^t_{d_g \to h_g} + \mu^t_{s_{j_g} \to h_g} \tag{22.139}$$

And similarly

$$m^t_{h_g \to s_{j_g}}(s_{j_g}) = \int\int h_g(d_g, s_{i_g}, s_{j_g}) m^t_{d_g \to h_g}(d_g) m^t_{s_{i_g} \to h_g}(s_{i_g}) dd_g ds_{i_g} \tag{22.140}$$

$$= \mathcal{N}(s_{j_g} | \mu^t_{h_g \to s_{j_g}}, v^t_{h_g \to s_{j_g}}) \tag{22.141}$$

$$v^t_{h_g \to s_{j_g}} = 1 + v^t_{d_g \to h_g} + v^t_{s_{i_g} \to h_g} \tag{22.142}$$

$$\mu^t_{h_g \to s_{j_g}} = \mu^t_{d_g \to h_g} - \mu^t_{s_{i_g} \to h_g} \tag{22.143}$$

When we compute $q^{t+1}(s_{i_g})$ at the next iteration, by combining $m^t_{h_g \to s_{i_g}}(s_{i_g})$ with the prior factor, we will see that the posterior mean of s_{i_g} goes up. Similarly, the posterior mean of s_{j_g} goes down.

It is straightforward to combine EP with ADF to perform online inference, which is necessary for most practical applications.

Let us consider a simple example of this method. We create a partial ordering of 5 players as shown in Figure 22.13(a). We then sample some game outcomes from this graph, where a parent always beats a child. We pass this data into (5 iterations of) the EP algorithm and infer the posterior mean and variance for each player's skill level. The results are shown in Figure 22.13(b). We see that the method has correctly inferred the rank ordering of the players.

22.5.6 Other applications of EP

The TrueSkill model was developed by researchers at Microsoft. They and others have extended the model to a variety of other interesting applications, including personalized ad recommendation (Stern et al. 2009), predicting click-through-rate on ads in the Bing search engine (Graepel et al. 2010), etc. They have also developed a general purpose Bayesian inference toolbox based on EP called **infer.net** (Minka et al. 2010).

EP has also been used for a variety of other models, such as Gaussian process classification (Nickisch and Rasmussen 2008). See `http://research.microsoft.com/en-us/um/people/minka/papers/ep/roadmap.html` for a list of other EP applications.

22.6 MAP state estimation

In this section, we consider the problem of finding the most probable configuration of variables in a discrete-state graphical model, i.e., our goal is to find a MAP assignment of the following form:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}^m} p(\mathbf{x}|\boldsymbol{\theta}) = \arg\max_{\mathbf{x}\in\mathcal{X}^m} \sum_{i\in V}\theta_i(x_i) + \sum_{f\in F}\theta_f(\mathbf{x}_f) = \arg\max_{\mathbf{x}\in\mathcal{X}^m} \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x}) \quad (22.144)$$

where θ_i are the singleton node potentials, and θ_f are the factor potentials. (In this section, we follow the notation of (Sontag et al. 2011), which considers the case of general potentials, not just pairwise ones.) Note that the partition function $Z(\boldsymbol{\theta})$ plays no role in MAP estimation.

If the treewidth is low, we can solve this problem with the junction tree algorithm (Section 20.4), but in general this problem is intractable. In this section, we discuss various approximations, building on the material from Section 22.3.

22.6.1 Linear programming relaxation

We can rewrite the objective in terms of the variational parameters as follows:

$$\arg\max_{\mathbf{x}\in\mathcal{X}^m} \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x}) = \arg\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} \quad (22.145)$$

where $\boldsymbol{\phi}(\mathbf{x}) = [\{\mathbb{I}(x_s = j)\}, \{\mathbb{I}(\mathbf{x}_f = k)\}]$ and $\boldsymbol{\mu}$ is a probability vector in the marginal polytope. To see why this equation is true, note that we can just set $\boldsymbol{\mu}$ to be a degenerate distribution with $\mu(x_s) = \mathbb{I}(x_s = x_s^*)$, where x_s^* is the optimal assigment of node s. So instead of optimizing over discrete assignments, we now optimize over probability distributions $\boldsymbol{\mu}$.

It seems like we have an easy problem to solve, since the objective in Equation 22.145 is linear in $\boldsymbol{\mu}$, and the constraint set $\mathbb{M}(G)$ is convex. The trouble is, $\mathbb{M}(G)$ in general has a number of facets that is exponential in the number of nodes.

A standard strategy in combinatorial optimization is to relax the constraints. In this case, instead of requiring probability vector $\boldsymbol{\mu}$ to live in the marginal polytope $\mathbb{M}(G)$, we allow it to live inside a convex outer bound $\mathbb{L}(G)$. Having defined this relaxed constraint set, we have

$$\max_{\mathbf{x}\in\mathcal{X}^m} \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x}) = \max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} \le \max_{\boldsymbol{\tau}\in\mathbb{L}(G)} \boldsymbol{\theta}^T\boldsymbol{\tau} \tag{22.146}$$

If the solution is integral, it is exact; if it is fractional, it is an approximation. This is called a (first order) **linear programming relaxtion**. The reason it is called first-order is that the constraints that are enforced are those that correspond to consistency on a tree, which is a graph of treewidth 1. It is possible to enforce higher-order consistency, using graphs with larger treewidth (see (Wainwright and Jordan 2008b, sec 8.5) for details).

How should we actually perform the optimization? We can use a generic linear programming package, but this is often very slow. Fortunately, in the case of graphical models, it is possible to devise specialised distributed message passing algorithms for solving this optimization problem, as we explain below.

22.6.2 Max-product belief propagation

The MAP objective in Equation 22.145, $\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu}$, is almost identical to the inference objective in Equation 22.23, $\max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu})$, apart from the entropy term. One heuristic way to proceed would be to consider the **zero temperature limit** of the probability distribution $\boldsymbol{\mu}$, where the probability distribution has all its mass centered on its mode (see Section 4.2.2). In such a setting, the entropy term becomes zero. We can then modify the message passing methods used to solve the inference problem so that they solve the MAP estimation problem instead. In particular, in the zero temperature limit, the sum operator becomes the max operator, which results in a method called **max-product belief propagation**.

In more detail, let

$$A(\boldsymbol{\theta}) \triangleq \max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu}) \tag{22.147}$$

Now consider an inverse temperature β going to infinity. We have

$$\begin{aligned}
\lim_{\beta\to+\infty} \frac{A(\beta\boldsymbol{\theta})}{\beta} &= \lim_{\beta\to+\infty} \frac{1}{\beta} \max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \left\{(\beta\boldsymbol{\theta})^T\boldsymbol{\mu} + \mathbb{H}(\boldsymbol{\mu})\right\} && (22.148)\\
&= \max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \left\{\boldsymbol{\theta}^T\boldsymbol{\mu} + \lim_{\beta\to+\infty} \frac{1}{\beta}\mathbb{H}(\boldsymbol{\mu})\right\} && (22.149)\\
&= \max_{\boldsymbol{\mu}\in\mathbb{M}(G)} \boldsymbol{\theta}^T\boldsymbol{\mu} && (22.150)
\end{aligned}$$

It is the concavity of the objective function that allows us to interchange the $\lim$ and $\max$ operators (see (Wainwright and Jordan 2008b, p274) for details).

Now consider the Bethe approximation, which has the form $\max_{\boldsymbol{\tau}\in\mathbb{L}(G)} \boldsymbol{\theta}^T\boldsymbol{\tau} + \mathbb{H}_{\text{Bethe}}(\boldsymbol{\tau})$. We showed that loopy BP finds a local optimum of this objective. In the zero temperature limit, this objective is equivalent to the LP relaxation of the MAP problem. Unfortunately, max-product loopy BP does not solve this LP relaxation unless the graph is a tree (Wainwright and Jordan 2008b, p211). The reason is that Bethe energy functional is not concave (except on trees), so we are not licensed to swap the limit and max operators in the above zero-temperature derivation. However, if we use tree-reweighted BP, or TRBP/ TRW, we have a concave objective. In this case, one can show (Kolmogorov and Wainwright 2005) that the max-product version of TRBP does solve the above LP relaxation.

A certain scheduling of this algorithm, known as **sequential TRBP**, **TRBP-S**, or **TRW-S**, can be shown to always converge (Kolmogorov 2006), and furthermore, it typically does so faster than the standard parallel updates. The idea is to pick an arbitrary node ordering $X_1, \ldots, X_N$. We then consider a set of trees which is a subsequence of this ordering. At each iteration, we perform max-product BP from X_1 towards X_N and back along one of these trees. It can be shown that this monotonically minimizes a lower bound on the energy, and thus is guaranteed to converge to the global optimum of the LP relaxation.

22.6.3 Graphcuts

In this section, we show how to find MAP state estimates, or equivalently, minimum energy configurations, by using the **max flow/min cut** algorithm for graphs.[8] This class of methods is known as **graphcuts** and is very widely used, especially in computer vision applications.

We will start by considering the case of MRFs with binary nodes and a restricted class of potentials; in this case, graphcuts will find the exact global optimum. We then consider the case of multiple states per node, which are assumed to have some underlying ordering; we can approximately solve this case by solving a series of binary subproblems, as we will see.

22.6.3.1 Graphcuts for the generalized Ising model

Let us start by considering a binary MRF where the edge energies have the following form:

$$E_{uv}(x_u, x_v) = \begin{cases} 0 & \text{if } x_u = x_v \\ \lambda_{st} & \text{if } x_u \neq x_v \end{cases} \tag{22.151}$$

where $\lambda_{st} \geq 0$ is the edge cost. This encourages neighboring nodes to have the same value (since we are trying to minimize energy). Since we are free to add any constant we like to the overall energy without affecting the MAP state estimate, let us rescale the local energy terms such that either $E_u(1) = 0$ or $E_u(0) = 0$.

Now let us construct a graph which has the same set of nodes as the MRF, plus two distinguished nodes: the source s and the sink$t$. If $E_u(1) = 0$, we add the edge $x_u \to t$ with cost $E_u(0)$. (This ensures that if u is not in partition $\mathcal{X}_t$, meaning u is assigned to state 0, we will pay a cost of $E_u(0)$ in the cut.) Similarly, If $E_u(0) = 0$, we add the edge $x_u \to s$ with cost $E_u(1)$. Finally, for every pair of variables that are connected in the MRF, we add edges $x_u \to x_v$ and $x_v \to x_u$, both with cost $\lambda_{u,v} \geq 0$. Figure 22.14 illustrates this construction for an MRF with 4 nodes, and with the following non-zero energy values:

$$E_1(0) = 7, E_2(1) = 2, E_3(1) = 1, E_4(1) = 6 \tag{22.152}$$

$$\lambda_{1,2} = 6, \lambda_{2,3} = 6, \lambda_{3,4} = 2, \lambda_{1,4} = 1 \tag{22.153}$$

Having constructed the graph, we compute a minimal $s - t$ cut. This is a partition of the nodes into two sets, $\mathcal{X}_s$, which are nodes connected to s, and $\mathcal{X}_t$, which are nodes connected to t. We

8. There are a variety of ways to implement this algorithm, see e.g., (Sedgewick and Wayne 2011). The best take $O(EV \log V)$ or $O(V^3)$ time, where E is the number of edges and V is the number of nodes.

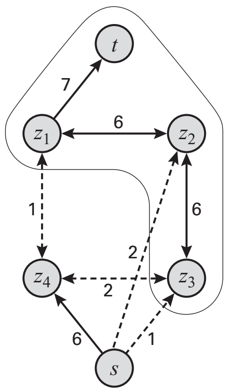


Figure 22.14 Illustration of graphcuts applied to an MRF with 4 nodes. Dashed lines are ones which contribute to the cost of the cut (for bidirected edges, we only count one of the costs). Here the min cut has cost 6. Source: Figure 13.5 from (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

pick the partition which minimizes the sum of the cost of the edges between nodes on different sides of the partition:

$$\text{cost}(\mathcal{X}_s, \mathcal{X}_t) = \sum_{x_u \in \mathcal{X}_s, x_v \in \mathcal{X}_t} \text{cost}(x_u, s_v) \tag{22.154}$$

In Figure 22.14, we see that the min-cut has cost 6.

Minimizing the cost in this graph is equivalent to minimizing the energy in the MRF. Hence nodes that are assigned to s have an optimal state of 0, and the nodes that are assigned to t have an optimal state of 1. In Figure 22.14, we see that the optimal MAP estimate is $(1, 1, 1, 0)$.

22.6.3.2 Graphcuts for binary MRFs with submodular potentials

We now discuss how to extend the graphcuts construction to binary MRFs with more general kinds of potential functions. In particular, suppose each pairwise energy satisfies the following condition:

$$E_{uv}(1,1) + E_{uv}(0,0) \leq E_{uv}(1,0) + E_{uv}(0,1) \tag{22.155}$$

In other words, the sum of the diagonal energies is less than the sum of the off-diagonal energies. In this case, we say the energies are **submodular** (Kolmogorov and Zabin 2004).[9] An example of a submodular energy is an Ising model where $\lambda_{uv} > 0$. This is also known as an **attractive MRF** or **associative MRF**, since the model "wants" neighboring states to be the same.

9. Submodularity is the discrete analog of convexity. Intuitively, it corresponds to the "law of diminishing returns", that is, the extra value of adding one more element to a set is reduced if the set is already large. More formally, we say that $f : 2^S \to R$ is submodular if for any $A \subset B \subset S$ and $x \in S$, we have $f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$. If $-f$ is submodular, then f is **supermodular**.

To apply graphcuts to a binary MRF with submodular potentials, we construct the pairwise edge weights as follows:

$$E'_{u,v}(0,1) = E_{u,v}(1,0) + E_{u,v}(0,1) - E_{u,v}(0,0) - E_{u,v}(1,1) \tag{22.156}$$

This is guaranteed to be non-negative by virtue of the submodularity assumption. In addition, we construct new local edge weights as follows: first we initialize $E'(u) = E(u)$, and then for each edge pair (u,v), we update these values as follows:

$$E'_u(1) = E'_u(1) + (E_{u,v}(1,0) - E_{u,v}(0,0)) \tag{22.157}$$

$$E'_v(1) = E'_u(1) + (E_{u,v}(1,1) - E_{u,v}(1,0)) \tag{22.158}$$

We now construct a graph in a similar way to before. Specifically, if $E'_u(1) > E'_u(0)$, we add the edge $u \to s$ with cost $E'_u(1) - E'_u(0)$, otherwise we add the edge $u \to t$ with cost $E'_u(0) - E'_u(1)$. Finally for every MRF edge for which $E'_{u,v}(0,1) > 0$, we add a graphcuts edge $x_u - x_v$ with cost $E'_{u,v}(0,1)$. (We don't need to add the edge in both directions.)

One can show (Exercise 22.1) that the min cut in this graph is the same as the minimum energy configuration. Thus we can use max flow/min cut to find the globally optimal MAP estimate (Greig et al. 1989).

22.6.3.3 Graphcuts for nonbinary metric MRFs

We now discuss how to use graphcuts for approximate MAP estimation in MRFs where each node can have multiple states (Boykov et al. 2001). However, we require that the pairwise energies form a metric. We call such a model a **metric MRF**. For example, suppose the states have a natural ordering, as commonly arises if they are a discretization of an underlying continuous space. In this case, we can define a metric of the form $E(x_s, x_t) = \min(\delta, ||x_s - x_t||)$ or a semi-metric of the form $E(x_s, x_t) = \min(\delta, (x_s - x_t)^2)$, for some constant $\delta > 0$. This energy encourages neighbors to have similar labels, but never "punishes" them by more than δ. This δ term prevents over-smoothing, which we illustrate in Figure 19.20.

One version of graphcuts is the **alpha expansion**. At each step, it picks one of the available labels or states and calls it α; then it solves a binary subproblem where each variable can choose to remain in its current state, or to become state α (see Figure 22.15(d) for an illustration). More precisely, we define a new MRF on binary nodes, and we define the energies of this new model, relative to the current assignment $\mathbf{x}$, as follows:

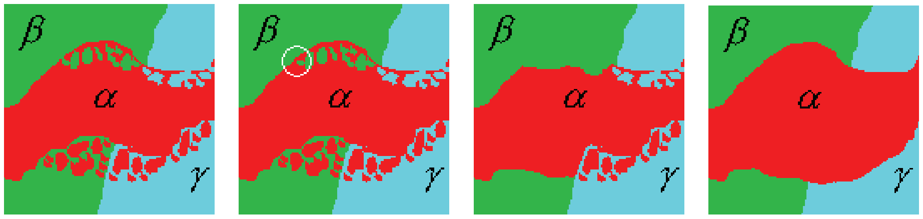
$$E'_u(0) = E_u(x_u), E'_u(1) = E_u(\alpha), E'_{u,v}(0,0) = E_{u,v}(x_u, x_v) \tag{22.159}$$

$$E'_{u,v}(0,1) = E_{u,v}(x_u, \alpha), E'_{u,v}(1,0) = E_{u,v}(\alpha, x_v), E'_{u,v}(1,1) = E_{u,v}(\alpha, \alpha) \tag{22.160}$$

To optimize E' using graph cuts (and thus figure out the optimal alpha expansion move), we require that the energies be submodular. Plugging in the definition we get the following constraint:

$$E_{u,v}(x_u, x_v) + E_{u,v}(\alpha, \alpha) \leq E_{u,v}(x_u, \alpha) + E_{u,v}(\alpha, x_v) \tag{22.161}$$

For any distance function, $E_{u,v}(\alpha, \alpha) = 0$, and the remaining inequality follows from the triangle inequality. Thus we can apply the alpha expansion move to any metric MRF.



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

Figure 22.15 (a) An image with 3 labels. (b) A standard local move (e.g., by iterative conditional modes) just flips the label of one pixel. (c) An $\alpha - \beta$ swap allows all nodes that are currently labeled as α to be relabeled as β if this decreases the energy. (d) An α expansion allows all nodes that are not currently labeled as α to be relabeled as α if this decreases the energy. Source: Figure 2 of (Boykov et al. 2001). Used with kind permission of Ramin Zabih.

At each step of alpha expansion, we find the optimal move from amongst an exponentially large set; thus we reach a **strong local optimum**, of much lower energy than the local optima found by standard greedy label flipping methods such as iterative conditional modes. In fact, one can show that, once the algorithm has converged, the energy of the resulting solution is at most $2c$ times the optimal energy, where

$$c = \max_{(u,v)\in\mathcal{E}} \frac{\max_{\alpha\neq\beta} E_{uv}(\alpha,\beta)}{\min_{\alpha\neq\beta} E_{uv}(\alpha,\beta)} \tag{22.162}$$

See Exercise 22.3 for the proof. In the case of the Potts model, $c = 1$, so we have a 2-approximation.

Another version of graphcuts is the **alpha-beta swap**. At each step, two labels are chosen, call them α and β. All the nodes currently labeled α can change to β (and vice versa) if this reduces the energy (see Figure 22.15(c) for an illustration). The resulting binary subproblem can be solved exactly, even if the energies are only semi-metric (that is, the triangle inequality need not hold; see Exercise 22.2). Although the $\alpha - \beta$ swap version can be applied to a broader class of models than the α-expansion version, it is theoretically not as powerful. Indeed, in various low-level vision problems, (Szeliski et al. 2008) show empirically that the expansion version is usually better than the swap version (see Section 22.6.4).

22.6.4 Experimental comparison of graphcuts and BP

In Section 19.6.2.7, we described lattice-structured CRFs for various low-level vision problems. (Szeliski et al. 2008) performed an extensive comparison of different approximate optimization techniques for this class of problems. Some of the results, for the problem of stereo depth estimation, are shown in Figure 22.16. We see that the graphcut and tree-reweighted max-product BP (TRW) give the best results, with regular max-product BP being much worse. In terms of speed, graphcuts is the fastest, with TRW a close second. Other algorithms, such as ICM, simulated annealing or a standard domain-specific heuristic known as normalize correlation, are

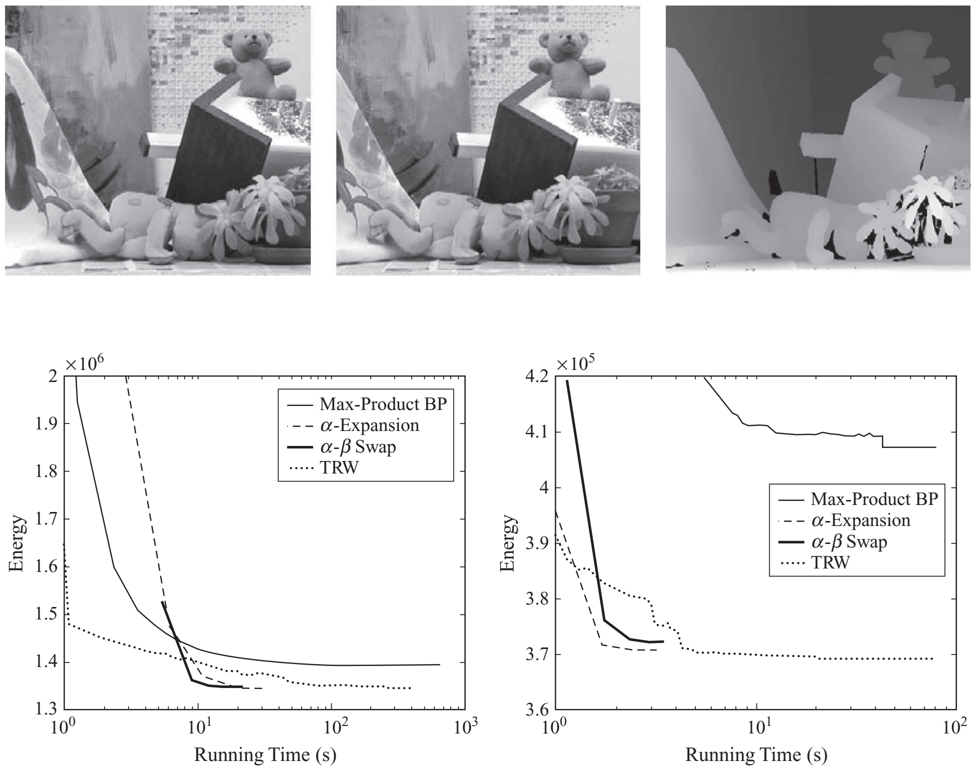
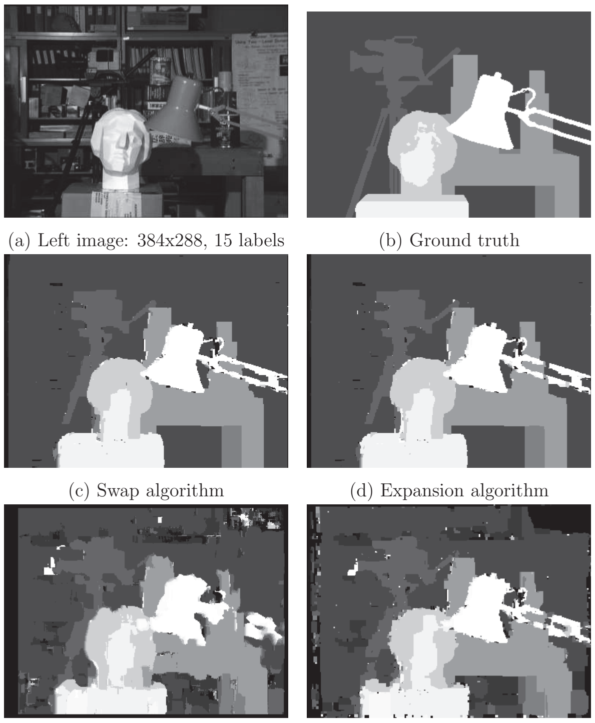


Figure 22.16 Energy minimization on a CRF for stereo depth estimation. Top row: two input images along with the ground truth depth values. Bottom row: energy vs time for 4 different optimization algorithms. Bottom left: results are for the Teddy image (shown in top row). Bottom right: results are for the Tsukuba image (shown in Figure 22.17(a)). Source: Figure 13.B.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

even worse, as shown qualitatively in Figure 22.17.

Since TRW is optimizing the dual of the relaxed LP problem, we can use its value at convergence to evaluate the optimal energy. It turns out that for many of the images in the stereo benchmark dataset, the ground truth has higher energy (lower probability) than the globally optimal estimate (Meltzer et al. 2005). This indicates that we are optimizing the wrong model. This is not surprising, since the pairwise CRF ignores known long-range constraints. Unfortunately, if we add these constraints to the model, the graph either becomes too dense (making BP slow), and/or the potentials become non-submodular (making graphcuts inapplicable).

One way around this is to generate a diverse set of local modes, using repeated applications of graph cuts, as described in (Yadollahpour et al. 2011). We can then apply a more sophisticated model, which uses global features, to rerank the solutions.



(a) Left image: 384x288, 15 labels
(b) Ground truth
(c) Swap algorithm
(d) Expansion algorithm
(e) Normalized correlation
(f) Simulated annealing

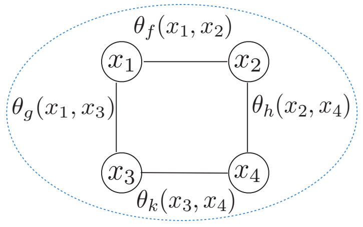
Figure 22.17 An example of stereo depth estimation using an MRF. (a) Left image, of size 384×288 pixels, from the University of Tsukuba. (The corresponding right image is similar, but not shown.) (b) Ground truth depth map, quantized to 15 levels. (c-f): MAP estimates using different methods: (c) $\alpha - \beta$ swap, (d) α expansion, (e) normalized cross correlation, (f) simulated annealing. Source: Figure 10 of (Boykov et al. 2001). Used with kind permission of Ramin Zabih.

22.6.5 Dual decomposition

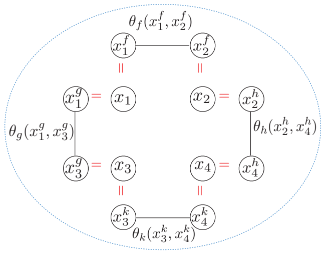
We are interested in computing

$$p^* = \max_{\mathbf{x} \in \mathcal{X}^m} \sum_{i \in V} \theta_i(x_i) + \sum_{f \in F} \theta_f(\mathbf{x}_f) \tag{22.163}$$

where F represents a set of factors. We will assume that we can tractably optimize each local factor, but the combination of all of these factors makes the problem intractable. One way to proceed is to optimize each term independently, but then to introduce constraints that force all the local estimates of the variables' values to agree with each other. We explain this in more detail below, following the presentation of (Sontag et al. 2011).



(a)



(b)

Figure 22.18 (a) A pairwise MRF with 4 different edge factors. (b) We have 4 separate variables, plus a copy of each variable for each factor it participates in. Source: Figure 1.2-1.3 of (Sontag et al. 2011). Used with kind permission of David Sontag.

22.6.5.1 Basic idea

Let us duplicate the variables x_i, once for each factor, and then force them to be equal. Specifically, let $\mathbf{x}_f^f = \{x_i^f\}_{i \in f}$ be the set of variables used by factor f. This construction is illustrated in Figure 22.18. We can reformulate the objective as follows:

$$p^* = \max_{\mathbf{x},\mathbf{x}^f} \sum_{i \in V} \theta_i(x_i) + \sum_{f \in F} \theta_f(\mathbf{x}_f^f) \qquad \text{s.t.} \quad x_i^f = x_i \;\; \forall f, i \in f \tag{22.164}$$

Let us now introduce Lagrange multipliers, or dual variables, $\delta_{fi}(k)$, to enforce these constraints. The Lagrangian becomes

$$L(\boldsymbol{\delta}, \mathbf{x}, \mathbf{x}^f) = \sum_{i \in V} \theta_i(x_i) + \sum_{f \in F} \theta_f(\mathbf{x}_f^f) \tag{22.165}$$

$$+ \sum_{f \in F} \sum_{i \in f} \sum_{\hat{x}_i} \delta_{fi}(\hat{x}_i) \left(\mathbb{I}(x_i = \hat{x}_i) - \mathbb{I}(x_i^f = \hat{x}_i) \right) \tag{22.166}$$

This is equivalent to our original problem in the following sense: for any value of $\boldsymbol{\delta}$, we have

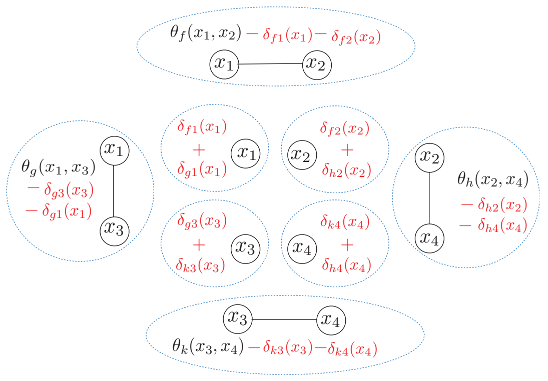
$$p^* = \max_{\mathbf{x},\mathbf{x}^f} L(\boldsymbol{\delta}, \mathbf{x}, \mathbf{x}^f) \qquad \text{s.t.} \quad x_i^f = x_i \;\; \forall f, i \in f \tag{22.167}$$

since if the constraints hold, the last term is zero. We can get an upper bound by dropping the consistency constraints, and just optimizing the following upper bound:

$$L(\boldsymbol{\delta}) \triangleq \max_{\mathbf{x},\mathbf{x}^f} L(\boldsymbol{\delta}, \mathbf{x}, \mathbf{x}^f) \tag{22.168}$$

$$= \sum_i \max_{x_i} \left(\theta_i(x_i) + \sum_{f: i \in f} \delta_{fi}(x_i) \right) + \sum_f \max_{\mathbf{x}_f} \left(\theta_f(\mathbf{x}_f) - \sum_{i \in f} \delta_{fi}(x_i) \right) \tag{22.169}$$

See Figure 22.19 for an illustration.



$\theta_f(x_1,x_2) - \delta_{f1}(x_1) - \delta_{f2}(x_2)$
x_1
x_2
$\theta_g(x_1,x_3) - \delta_{g3}(x_3) - \delta_{g1}(x_1)$
$\delta_{f1}(x_1) + \delta_{g1}(x_1)$
$\delta_{f2}(x_2) + \delta_{h2}(x_2)$
$\theta_h(x_2,x_4) - \delta_{h2}(x_2) - \delta_{h4}(x_4)$
$\delta_{g3}(x_3) + \delta_{k3}(x_3)$
$\delta_{k4}(x_4) + \delta_{h4}(x_4)$
x_3
x_4
$\theta_k(x_3,x_4) - \delta_{k3}(x_3) - \delta_{k4}(x_4)$

Figure 22.19 Illustration of dual decomposition. Source: Figure 1.2 of (Sontag et al. 2011). Used with kind permission of David Sontag.

This objective is tractable to optimize, since each $\mathbf{x}_f$ term is decoupled. Furthermore, we see that $L(\boldsymbol{\delta}) \geq p^*$, since by relaxing the consistency constraints, we are optimizing over a larger space. Furthermore, we have the property that

$$\min_{\boldsymbol{\delta}} L(\boldsymbol{\delta}) = p^* \tag{22.170}$$

so the upper bound is tight at the optimal value of $\boldsymbol{\delta}$, which enforces the original constraints.

Minimizing this upper bound is known as **dual decomposition** or **Lagrangian relaxation** (Komodakis et al. 2011; Sontag et al. 2011; Rush and Collins 2012). Furthemore, it can be shown that $L(\boldsymbol{\delta})$ is the dual to the same LP relaxation we saw before. We will discuss several possible optimization algorithms below.

The main advantage of dual decomposition from a practical point of view is that it allows one to mix and match different kinds of optimization algorithms in a convenient way. For example, we can combine a grid structured graph with local submodular factors to perform image segmentation, together with a tree structured model to perform pose estimation (see Exercise 22.4). Analogous methods can be used in natural language processing, where we often have a mix of local and global constraints (see e.g., (Koo et al. 2010; Rush and Collins 2012)).

22.6.5.2 Theoretical guarantees

What can we say about the quality of the solutions obtained in this way? To understand this, let us first introduce some more notation:

$$\overline{\theta}_i^{\boldsymbol{\delta}}(x_i) \triangleq \theta_i(x_i) + \sum_{f:i\in f} \delta_{fi}(x_i) \tag{22.171}$$

$$\overline{\theta}_f^{\boldsymbol{\delta}}(\mathbf{x}_f) \triangleq \theta_f(\mathbf{x}_f) - \sum_{i\in f} \delta_{fi}(x_i) \tag{22.172}$$

This represents a reparameterization of the original problem, in the sense that

$$\sum_i \theta_i(x_i) + \sum_f \theta_f(\mathbf{x}_f) = \sum_i \overline{\theta}_i^{\boldsymbol{\delta}}(x_i) + \sum_f \overline{\theta}_f^{\boldsymbol{\delta}}(\mathbf{x}_f) \tag{22.173}$$

and hence

$$L(\boldsymbol{\delta}) = \sum_i \max_{x_i} \overline{\theta}_i^{\boldsymbol{\delta}}(x_i) + \sum_f \max_{\mathbf{x}_f} \overline{\theta}_f^{\boldsymbol{\delta}}(\mathbf{x}_f) \tag{22.174}$$

Now suppose there is a set of dual variables $\boldsymbol{\delta}^*$ and an assignment $\mathbf{x}^*$ such that the maximizing assignments to the singleton terms agrees with the assignments to the factor terms, i.e., so that $x_i^* \in \operatorname{argmax}_{x_i} \overline{\theta}_i^{\boldsymbol{\delta}^*}(x_i)$ and $\mathbf{x}_f^* \in \operatorname{argmax}_{\mathbf{x}_f} \overline{\theta}_f^{\boldsymbol{\delta}^*}(\mathbf{x}_f)$. In this case, we have

$$L(\boldsymbol{\delta}^*) = \sum_i \overline{\theta}_i^{\boldsymbol{\delta}^*}(x_i^*) + \sum_f \overline{\theta}_f^{\boldsymbol{\delta}^*}(\mathbf{x}_f^*) = \sum_i \theta_i(x_i^*) + \sum_f \theta_f(\mathbf{x}_f^*) \tag{22.175}$$

Now since

$$\sum_i \theta_i(x_i^*) + \sum_f \theta_f(\mathbf{x}_f^*) \leq p^* \leq L(\boldsymbol{\delta}^*) \tag{22.176}$$

we conclude that $L(\boldsymbol{\delta}^*) = p^*$, so $\mathbf{x}^*$ is the MAP assignment.

So if we can find a solution where all the subproblems agree, we can be assured that it is the global optimum. This happens surprisingly often in practical problems.

22.6.5.3 Subgradient descent

$L(\boldsymbol{\delta})$ is a convex and continuous objective, but it is non-differentiable at points $\boldsymbol{\delta}$ where $\overline{\theta}_i^{\boldsymbol{\delta}}(x_i)$ or $\overline{\theta}_f^{\boldsymbol{\delta}}(\mathbf{x}_f)$ have multiple optima. One approach is to use subgradient descent. This updates all the elements of $\boldsymbol{\delta}$ at the same time, as follows:

$$\delta_{fi}^{t+1}(x_i) = \delta_{fi}^t(x_i) - \alpha_t g_{fi}^t(x_i) \tag{22.177}$$

where $\mathbf{g}^t$ the subgradient of $L(\boldsymbol{\delta})$ at $\boldsymbol{\delta}^t$. If the step sizes α_t are set appropriately (see Section 8.5.2.1), this method is guaranteed to converge to a global optimum of the dual. (See (Komodakis et al. 2011) for details.)

One can show that the gradient is given by the following sparse vector. First let $x_i^s \in \operatorname{argmax}_{x_i} \overline{\theta}_i^{\boldsymbol{\delta}^t}(x_i)$ and $\mathbf{x}_f^f \in \operatorname{argmax}_{\mathbf{x}_f} \overline{\theta}_f^{\boldsymbol{\delta}^t}(\mathbf{x}_f)$. Next let $g_{fi}(x_i) = 0$ for all elements. Finally, if $x_i^f \neq x_i^s$ (so factor f disagrees with the local term on how to set variable i), we set $g_{fi}(x_i^s) = +1$ and $g_{fi}(x_i^f) = -1$. This has the effect of decreasing $\overline{\theta}_i^{\boldsymbol{\delta}^t}(x_i^s)$ and increasing $\overline{\theta}_i^{\boldsymbol{\delta}^t}(x_i^f)$, bringing them closer to agreement. Similarly, the subgradient update will decrease the value of $\overline{\theta}_f^{\boldsymbol{\delta}^t}(x_i^f, \mathbf{x}_{f\setminus i})$ and increasing the value of $\overline{\theta}_f^{\boldsymbol{\delta}^t}(x_i^s, \mathbf{x}_{f\setminus i})$.

To compute the gradient, we need to be able to solve subproblems of the following form:

$$\operatorname*{argmax}_{\mathbf{x}_f} \overline{\theta}_f^{\boldsymbol{\delta}^t}(\mathbf{x}_f) = \operatorname*{argmax}_{\mathbf{x}_f} \left[\theta_f(\mathbf{x}_f) - \sum_{i \in f} \delta_{fi}^t(x_i) \right] \tag{22.178}$$

(In (Komodakis et al. 2011), these subproblems are called slaves, whereas $L(\boldsymbol{\delta})$ is called the master.) Obviously if the scope of factor f is small, this is simple. For example, if each factor is pairwise, and each variable has K states, the cost is just K^2. However, there are some kinds of global factors that also support exact and efficient maximization, including the following:

- Graphical models with low tree width.
- Factors that correspond to bipartite graph matchings (see e.g., (Duchi et al. 2007)). This is useful for data association problems, where we must match up a sensor reading with an unknown source. We can find the maximal matching using the so-called Hungarian algorithm in $O(|f|^3)$ time (see e.g., (Padadimitriou and Steiglitz 1982)).
- Supermodular functions. We discuss this case in more detail in Section 22.6.3.2.
- Cardinality constraints. For example, we might have a factor over a large set of binary variables that enforces that a certain number of bits are turned on; this can be useful in problems such as image segmentation. In particular, suppose $\theta_f(\mathbf{x}_f) = 0$ if $\sum_{i \in f} x_i = L$ and $\theta_f(\mathbf{x}_f) = -\infty$ otherwise. We can find the maximizing assignment in $O(|f| \log |f|)$ time as follows: first define $e_i = \delta_{fi}(1) - \delta_{fi}(0)$; now sort the e_i; finally set $x_i = 1$ for the first L values, and $x_i = 0$ for the rest (Tarlow et al. 2010).
- Factors which are constant for all but a small set S of distinguished values of $\mathbf{x}_f$. Then we can optimize over the factor in $O(|S|)$ time (Rother et al. 2009).

22.6.5.4 Coordinate descent

An alternative to updating the entire $\boldsymbol{\delta}$ vector at once (albeit sparsely) is to update it using block coordinate descent. By choosing the size of the blocks, we can trade off convergence speed with ease of the local optimization problem.

One approach, which optimizes $\delta_{fi}(x_i)$ for all $i \in f$ and all x_i at the same time (for a fixed factor f), is known as **max product linear programming** (Globerson and Jaakkola 2008). Algorithmically, this is similar to belief propagation on a factor graph. In particular, we define $\delta_{f \to i}$ as messages sent from factor f to variable i, and we define $\delta_{i \to f}$ as messages sent from variable i to factor f. These messages can be computed as follows (see (Globerson and Jaakkola 2008) for the derivation):[10]

$$\delta_{i \to f}(x_i) = \theta_i(x_i) + \sum_{g \neq f} \delta_{g \to i}(x_i) \tag{22.179}$$

$$\delta_{f \to i}(x_i) = -\delta_{i \to f}(x_i) + \frac{1}{|f|} \max_{\mathbf{x}_{f \setminus i}} \left[\theta_f(\mathbf{x}_f) + \sum_{j \in f} \delta_{j \to f}(x_j) \right] \tag{22.180}$$

We then set the dual variables $\delta_{fi}(x_i)$ to be the messages $\delta_{f \to i}(x_i)$.

For example, consider a 2×2 grid MRF, with the following pairwise factors: $\theta_f(x_1, x_2)$, $\theta_g(x_1, x_3)$, $\theta_h(x_2, x_4)$, and $\theta_k(x_3, x_4)$. The outgoing message from factor f to variable 2 is a

10. Note that we denote their $\delta_i^{-f}(x_i)$ by $\delta_{i \to f}(x_i)$.

function of all messages coming into f, and f's local factor:

$$\delta_{f\to 2}(x_2) = -\delta_{2\to f}(x_2) + \frac{1}{2}\max_{x_1}\left[\theta_f(x_1,x_2) + \delta_{1\to f}(x_1) + \delta_{2\to f}(x_2)\right] \tag{22.181}$$

Similarly, the outgoing message from variable 2 to factor f is a function of all the messages sent into variable 2 from other connected factors (in this example, just factor h) and the local potential:

$$\delta_{2\to f}(x_2) = \theta_2(_2) + \delta_{h2}(x_2) \tag{22.182}$$

The key computational bottleneck is computing the max marginals of each factor, where we max out all the variables from $\mathbf{x}_f$ except for x_i, i.e., we need to be able to compute the following max marginals efficiently:

$$\max_{\mathbf{x}_{f\setminus i}} h(\mathbf{x}_{f\setminus i}, x_i), \quad h(\mathbf{x}_{f\setminus i}, x_i) \triangleq \theta_f(\mathbf{x}_f) + \sum_{j\in f}\delta_{jf}(x_j) \tag{22.183}$$

The difference from Equation 22.178 is that we are maxing over all but one of the variables. We can solve this efficiently for low treewidth graphical models using message passing; we can also solve this efficiently for factors corresponding to bipartite matchings (Duchi et al. 2007) or to cardinality constraints (Tarlow et al. 2010). However, there are cases where maximizing over all the variables in a factor's scope is computationally easier than maximizing over all-but-one (see (Sontag et al. 2011, Sec 1.5.4) for an example); in such cases, we may prefer to use a subgradient method.

Coordinate descent is a simple algorithm that is often much faster at minimizing the dual than gradient descent, especially in the early iterations. It also reduces the objective monotonically, and does not need any step size parameters. Unfortunately, it is not guaranteed to converge to the global optimum, since $L(\boldsymbol{\delta})$ is convex but not strictly convex (which implies there may be more than one globally optimizing value). One way to ensure convergence is to replace the max function in the definition of $L(\boldsymbol{\delta})$ with the soft-max function, which makes the objective strictly convex (see e.g., (Hazan and Shashua 2010) for details).

22.6.5.5 Recovering the MAP assignment

So far, we have been focussing on finding the optimal value of $\boldsymbol{\delta}^*$. But what we really want is the optimal value of $\mathbf{x}^*$. In general, computing $\mathbf{x}^*$ from $\boldsymbol{\delta}^*$ is NP-hard, even if the LP relaxation is tight and the MAP assignment is unique (Sontag et al. 2011, Theorem 1.4). (The troublesome cases arise when there are fractional assignments with the same optimal value as the MAP estimate.)

However, suppose that each $\overline{\theta}_i^{\boldsymbol{\delta}^*}$ has a unique maximum, x_i^*; in this case, we say that $\boldsymbol{\delta}^*$ is **locally decodable** to $\mathbf{x}^*$. One can show than in this case, the LP relaxation is unique and its solution is indeed $\mathbf{x}^*$. If many, but not all, of the nodes are uniquely decodable, we can “clamp” the uniquely decodable ones to their MAP value, and then use exact inference algorithms to figure out the optimal assignment to the remaining variables. Using this method, (Meltzer et al. 2005) was able to optimally solve various stereo vision CRF estimation problems, and (Yanover et al. 2007) was able to optimally solve various protein side-chain structure predicition problems.

Another approach is to use the upper bound provided by the dual in a branch and bound search procedure (Geoffrion 1974).

Exercises

Exercise 22.1 Graphcuts for MAP estimation in binary submodular MRFs

(Source: Ex. 13.14 of (Koller and Friedman 2009).). Show that using the graph construction described in Section 22.6.3.2, the cost of the cut is equal to the energy of the corresponding assignment, up to an irrelevant constant. (Warning: this exercise involves a lot of algebraic book-keeping.)

Exercise 22.2 Graphcuts for alpha-beta swap

(Source: Ex. 13.15 of (Koller and Friedman 2009).). Show how the optimal alpha-beta swap can be found by running min-cut on an appropriately constructed graph. More precisely,

a. Define a set of binary variables $t_1, \ldots, t_n$ such that $t_i = 0$ means $x_i' = \alpha$, $t_i = 1$ if $x_i' = \beta$, and $x_i' = x_i$ is unchanged f $x_i \neq \alpha$ and $x_i \neq \beta$.

b. Define an energy function over the new variables such that $E'(\mathbf{t}) = E(\mathbf{x}) + \text{const}$.

c. Show that E' is submodular if E is a semimetric.

Exercise 22.3 Constant factor optimality for alpha-expansion

(Source: Daphne Koller.). Let $\mathcal{X}$ be a pairwise metric Markov random field over a graph $G = (V, E)$. Suppose that the variables are nonbinary and that the node potentials are nonnegative. Let $\mathcal{A}$ denote the set of labels for each $X \in \mathcal{X}$. Though it is not possible to (tractably) find the globally optimal assignment $x^\star$ in general, the α-expansion algorithm provides a method for finding assignments $\hat{x}$ that are locally optimal with respect to a large set of transformations, *i.e.*, the possible α-expansion moves.

Despite the fact that α-expansion only produces a locally optimal MAP assignment, it is possible to prove that the energy of this assignment is within a known factor of the energy of the globally optimal solution $x^\star$. In fact, this is a special case of a more general principle that applies to a wide variety of algorithms, including max-product belief propagation and more general move-making algorithms: If one can prove that the solutions obtained by the algorithm are 'strong local minima', *i.e.*, local minima with respect to a large set of potential moves, then it is possible to derive bounds on the (global) suboptimality of these solutions, and the quality of the bounds will depend on the nature of the moves considered. (There is a precise definition of 'large set of moves'.)

Consider the following approach to proving the suboptimality bound for α-expansion.

a. Let $\hat{x}$ be a local minimum with respect to expansion moves. For each $\alpha \in \mathcal{A}$, let $V^\alpha = \{s \in V \mid x_s^\star = \alpha\}$, *i.e.*, the set of nodes labelled α in the global minimum. Let x' be an assignment that is equal to $x^\star$ on V^α and equal to $\hat{x}$ elsewhere; this is an α-expansion of $\hat{x}$. Verify that $E(x^\star) \leq E(\hat{x}) \leq E(x')$.

b. Building on the previous part, show that $E(\hat{x}) \leq 2cE(x^\star)$, where $c = \max_{(s,t)\in E}\left(\frac{\max_{\alpha\neq\beta} \varepsilon_{st}(\alpha,\beta)}{\min_{\alpha\neq\beta} \varepsilon_{st}(\alpha,\beta)}\right)$ and E denotes the energy of an assignment.
Hint. Think about where x' agrees with $\hat{x}$ and where it agrees with $x^\star$.

Exercise 22.4 Dual decomposition for pose segmentation

(Source: Daphne Koller.). Two important problems in computer vision are that of parsing articulated objects (*e.g.*, the human body), called *pose estimation*, and segmenting the foreground and the background, called *segmentation*. Intuitively, these two problems are linked, in that solving either one would be easier if the solution to the other were available. We consider solving these problems simultaneously using a joint model over human poses and foreground/background labels and then using dual decomposition for MAP inference in this model.

We construct a two-level model, where the high level handles pose estimation and the low level handles pixel-level background segmentation. Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected grid over the pixels. Each node $i \in \mathcal{V}$ represents a pixel. Suppose we have one binary variable x_i for each pixel, where $x_i = 1$ means that pixel i is in the foreground. Denote the full set of these variables by $\mathbf{x} = (x_i)$.

In addition, suppose we have an undirected tree structure $T = (\mathcal{V}', \mathcal{E}')$ on the parts. For each body part, we have a discrete set of candidate poses that the part can be in, where each pose is characterized by parameters specifying its position and orientation. (These candidates are generated by a procedure external to the algorithm described here.) Define y_{jk} to be a binary variable indicating whether body part $j \in \mathcal{V}'$ is in configuration k. Then the full set of part variables is given by $\mathbf{y} = (y_{jk})$, with $j \in \mathcal{V}'$ and $k = 1, \ldots, K$, where J is the total number of body parts and K is the number of candidate poses for each part. Note that in order to describe a valid configuration, $\mathbf{y}$ must satisfy the constraint that $\sum_{k=1}^{K} y_{jk} = 1$ for each j.

Suppose we have the following energy function on pixels:

$$E_1(\mathbf{x}) = \sum_{i \in \mathcal{V}} 1[x_i = 1] \cdot \theta_i + \sum_{(i,j) \in \mathcal{E}} 1[x_i \neq x_j] \cdot \theta_{ij}.$$

Assume that the θ_{ij} arises from a metric (*e.g.*, based on differences in pixel intensities), so this can be viewed as the energy for a pairwise metric MRF with respect to G.

We then have the following energy function for parts:

$$E_2(\mathbf{y}) = \sum_{p \in \mathcal{V}'} \theta_p(y_p) + \sum_{(p,q) \in \mathcal{E}'} \theta_{pq}(y_p, y_q).$$

Since each part candidate y_{jk} is assumed to come with a position and orientation, we can compute a binary mask in the image plane. The mask assigns a value to each pixel, denoted by $\{w^i_{jk}\}_{i \in \mathcal{V}}$, where $w^i_{jk} = 1$ if pixel i lies on the skeleton and decreases as we move away. We can use this to define an energy function relating the parts and the pixels:

$$E_3(\mathbf{x}, \mathbf{y}) = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}'} \sum_{k=1}^{K} 1[x_i = 0, y_{jk} = 1] \cdot w^i_{jk}.$$

In other words, this energy term only penalizes the case where a part candidate is active but the pixel underneath is labeled as background.

Formulate the minimization of $E_1 + E_2 + E_3$ as an integer program and show how you can use dual decomposition to solve the dual of this integer program. Your solution should describe the decomposition into slaves, the method for solving each one, and the update rules for the overall algorithm. Briefly justify your design choices, particularly your choice of inference algorithms for the slaves.

23 *Monte Carlo inference*

23.1 Introduction

So far, we discussed various deterministic algorithms for posterior inference. These methods enjoy many of the benefits of the Bayesian approach, while still being about as fast as optimization-based point-estimation methods. The trouble with these methods is that they can be rather complicated to derive, and they are somewhat limited in their domain of applicability (e.g., they usually assume conjugate priors and exponential family likelihoods, although see (Wand et al. 2011) for some recent extensions of mean field to more complex distributions). Furthermore, although they are fast, their accuracy is often limited by the form of the approximation which we choose.

In this chapter, we discuss an alternative class of algorithms based on the idea of Monte Carlo approximation, which we first introduced in Section 2.7. The idea is very simple: generate some (unweighted) samples from the posterior, $\mathbf{x}^s \sim p(\mathbf{x}|\mathcal{D})$, and then use these to compute any quantity of interest, such as a posterior marginal, $p(x_1|\mathcal{D})$, or the posterior of the difference of two quantities, $p(x_1 - x_2|\mathcal{D})$, or the posterior predictive, $p(y|\mathcal{D})$, etc. All of these quantities can be approximated by $\mathbb{E}\left[f|\mathcal{D}\right] \approx \frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^s)$ for some suitable function f.

By generating enough samples, we can achieve any desired level of accuracy we like. The main issue is: how do we efficiently generate samples from a probability distribution, particularly in high dimensions? In this chapter, we discuss non-iterative methods for generating independent samples. In the next chapter, we discuss an iterative method known as Markov Chain Monte Carlo, or MCMC for short, which produces dependent samples but which works well in high dimensions. Note that sampling is a large topic. The reader should consult other books, such as (Liu 2001; Robert and Casella 2004), for more information.

23.2 Sampling from standard distributions

We briefly discuss some ways to sample from 1 or 2 dimensional distributions of standard form. These methods are often used as subroutines by more complex methods.

23.2.1 Using the 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}

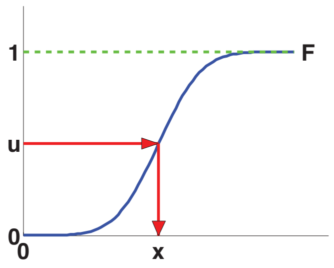


Figure 23.1 Sampling using an inverse CDF. Figure generated by `sampleCdf`.

be its inverse. Then we have the following result.

Theorem 23.2.1. *If $U \sim U(0,1)$ is a uniform rv, then $F^{-1}(U) \sim F$.*

Proof.

$$\begin{aligned} \Pr(F^{-1}(U) \le x) &= \Pr(U \le F(x)) \quad \text{(applying } F \text{ to both sides)} & (23.1) \\ &= F(x) \quad \text{(because } \Pr(U \le y) = y & (23.2) \end{aligned}$$

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., (Press et al. 1988) 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 23.1 for an illustration.

For example, consider the exponential distribution

$$\text{Expon}(x|\lambda) \triangleq \lambda e^{-\lambda x} \, \mathbb{I}(x \ge 0) \tag{23.3}$$

The cdf is

$$F(x) = 1 - e^{-\lambda x} \, \mathbb{I}(x \ge 0) \tag{23.4}$$

whose inverse is the quantile function

$$F^{-1}(p) = -\frac{\ln(1-p)}{\lambda} \tag{23.5}$$

By the above theorem, if $U \sim \text{Unif}(0,1)$, we know that $F^{-1}(U) \sim \text{Expon}(\lambda)$. Furthermore, since $1 - U \sim \text{Unif}(0,1)$ as well, we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using $-\ln(u)/\lambda$.

23.2.2 Sampling from a Gaussian (Box-Muller method)

We now 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 $p(\mathbf{z}) = \frac{1}{\pi}\mathbb{I}(z \text{ inside circle})$. Now define

$$x_i = z_i \left(\frac{-2 \ln r^2}{r^2} \right)^{\frac{1}{2}} \tag{23.6}$$

for $i = 1 : 2$, where $r^2 = z_1^2 + z_2^2$. Using the multivariate change of variables formula, we have

$$p(x_1, x_2) = p(z_1, z_2)|\frac{\partial(z_1, z_2)}{\partial(x_1, x_2)}| = \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x_1^2) \right] \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x_2^2) \right] \tag{23.7}$$

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, $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower triangular. Next we sample $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using the Box-Muller method. Finally we set $\mathbf{y} = \mathbf{L}\mathbf{x} + \boldsymbol{\mu}$. This is valid since

$$\text{cov}[\mathbf{y}] = \mathbf{L}\text{cov}[\mathbf{x}]\mathbf{L}^T = \mathbf{L}\ \mathbf{I}\ \mathbf{L}^T = \mathbf{\Sigma} \tag{23.8}$$

23.3 Rejection sampling

When the inverse cdf method cannot be used, one simple alternative is to use **rejection sampling**, which we now explain.

23.3.1 Basic idea

In rejection sampling, we create a **proposal distribution** $q(x)$ which satisifes $Mq(x) \geq \tilde{p}(x)$, for some constant M, where $\tilde{p}(x)$ is an unnormalized version of $p(x)$ (i.e., $p(x) = \tilde{p}(x)/Z_p$ for some possibly unknown constant Z_p). The function $Mq(x)$ provides an upper envelope for $\tilde{p}$. We then sample $x \sim q(x)$, which corresponds to picking a random x location, and then we sample $u \sim U(0, 1)$, which corresponds to picking a random height (y location) under the envelope. If $u > \frac{\tilde{p}(x)}{Mq(x)}$, we reject the sample, otherwise we accept it. See Figure 23.2(a). where 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 that this procedure is correct. Let

$$S = \{(x, u) : u \leq \tilde{p}(x)/Mq(x)\}, \ S_0 = \{(x, u) : x \leq x_0, u \leq \tilde{p}(x)/Mq(x)\} \tag{23.9}$$

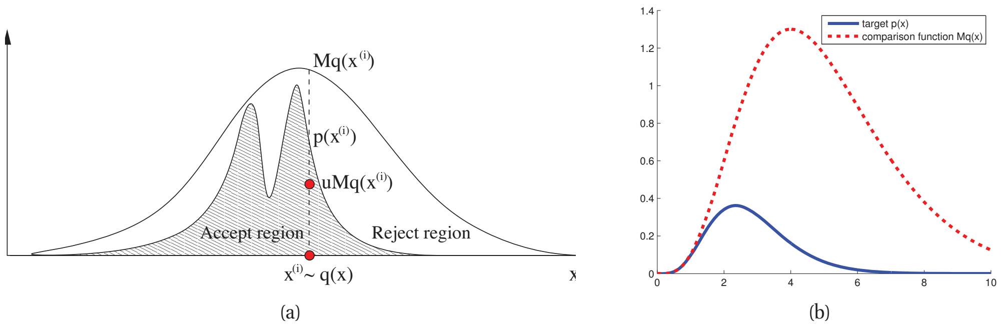


Figure 23.2 (a) Schematic illustration of rejection sampling. Source: Figure 2 of (Andrieu et al. 2003). Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\text{Ga}(\alpha = 5.7, \lambda = 2)$ distribution (solid blue) using a proposal of the form $M\text{Ga}(k, \lambda - 1)$ (dotted red), where $k = \lfloor 5.7 \rfloor = 5$. The curves touch at $\alpha - k = 0.7$. Figure generated by `rejectionSamplingDemo`.

Then the cdf of the accepted points is given by

$$P(x \le x_0 | x \text{ accepted}) = \frac{P(x \le x_0, x \text{ accepted})}{P(x \text{ accepted})} \tag{23.10}$$

$$= \frac{\int \int \mathbb{I}((x,u) \in S_0) q(x) du dx}{\int \int \mathbb{I}((x,u) \in S) q(x) du dx} = \frac{\int_{-\infty}^{x_0} \tilde{p}(x) dx}{\int_{-\infty}^{\infty} \tilde{p}(x) dx} \tag{23.11}$$

which is the cdf of $p(x)$, as desired.

How efficient is this method? Since we generate with probability $q(x)$ and accept with probability $\frac{\tilde{p}(x)}{Mq(x)}$, the probability of acceptance is

$$p(\text{accept}) = \int \frac{\tilde{p}(x)}{Mq(x)} q(x) dx = \frac{1}{M} \int \tilde{p}(x) dx \tag{23.12}$$

Hence we want to choose M as small as possible while still satisfying $Mq(x) \ge \tilde{p}(x)$.

23.3.2 Example

For example, suppose we want to sample from a Gamma distribution:[1]

$$\text{Ga}(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} \lambda^{\alpha} \exp(-\lambda x) \tag{23.13}$$

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, we cannot use this trick. However, we can use rejection sampling

1. This section is based on notes by Ioana A. Cosma, available at `http://users.aims.ac.za/~ioana/cp2.pdf`.

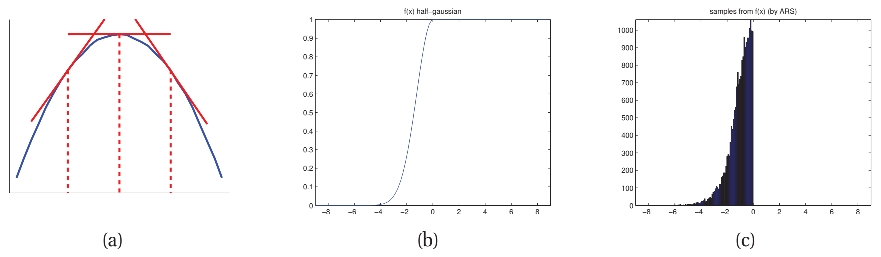


Figure 23.3 (a) Idea behind adaptive rejection sampling. We place piecewise linear upper (and lower) bounds on the log-concave density. Based on Figure 1 of (Gilks and Wild 1992). Figure generated by `arsEnvelope`. (b-c) Using ARS to sample from a half-Gaussian. Figure generated by `arsDemo`, written by Daniel Eaton.

using a $\text{Ga}(k, \lambda - 1)$ distribution as a proposal, where $k = \lfloor \alpha \rfloor$. The ratio has the form

$$\frac{p(x)}{q(x)} = \frac{\text{Ga}(x|\alpha, \lambda)}{\text{Ga}(x|k, \lambda-1)} = \frac{x^{\alpha-1}\lambda^\alpha \exp(-\lambda x)/\Gamma(\alpha)}{x^{k-1}(\lambda-1)^k \exp(-(\lambda-1)x)/\Gamma(k)} \tag{23.14}$$

$$= \frac{\Gamma(k)\lambda^\alpha}{\Gamma(\alpha)(\lambda-1)^k} x^{\alpha-k}\exp(-x) \tag{23.15}$$

This ratio attains its maximum when $x = \alpha - k$. Hence

$$M = \frac{\text{Ga}(\alpha-k|\alpha,\lambda)}{\text{Ga}(\alpha-k|k,\lambda-1)} \tag{23.16}$$

See Figure 23.2(b) for a plot. (Exercise 23.2 asks you to devise a better proposal distribution based on the Cauchy distribution.)

23.3.3 Application to Bayesian statistics

Suppose we want to draw (unweighted) samples from the posterior, $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. We can use rejection sampling with $\tilde{p}(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ as the target distribution, $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ as our proposal, and $M = p(\mathcal{D}|\hat{\boldsymbol{\theta}})$, where $\hat{\boldsymbol{\theta}} = \arg\max p(\mathcal{D}|\boldsymbol{\theta})$ is the MLE; this was first suggested in (Smith and Gelfand 1992). We accept points with probability

$$\frac{\tilde{p}(\boldsymbol{\theta})}{Mq(\boldsymbol{\theta})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D}|\hat{\boldsymbol{\theta}})} \tag{23.17}$$

Thus samples from the prior that have high likelihood are more likely to be retained in the posterior. Of course, if there is a big mismatch between prior and posterior (which will be the case if the prior is vague and the likelihood is informative), this procedure is very inefficient. We discuss better algorithms later.

23.3.4 Adaptive rejection sampling

We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to any log concave density $p(x)$. The idea is to upper bound the log density with a piecewise linear function, as illustrated in Figure 23.3(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:

$$q(x) = M_i \lambda_i \exp(-\lambda_i (x - x_{i-1})), \quad x_{i-1} < x \le x_i \tag{23.18}$$

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) (Gilks and Wild 1992). Figure 23.3(b-c) gives an example of the method in action. As with standard rejection sampling, it can be applied to unnormalized distributions.

23.3.5 Rejection sampling in high dimensions

It is clear that we want to make our proposal $q(x)$ as close as possible to the target distribution $p(x)$, while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To see this, consider sampling from $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I})$ using as a proposal $q(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_q^2 \mathbf{I})$. Obviously we must have $\sigma_q^2 \ge \sigma_p^2$ in order to be an upper bound. In D dimensions, the optimum value is given by $M = (\sigma_q/\sigma_p)^D$. The acceptance rate is $1/M$ (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.

In Chapter 24, we will describe MCMC sampling, which is a more efficient way to sample from high dimensional distributions. Sometimes this uses (adaptive) rejection sampling as a subroutine, which is known as **adaptive rejection Metropolis sampling** (Gilks et al. 1995).

23.4 Importance sampling

We now describe a Monte Carlo method known as **importance sampling** for approximating integrals of the form

$$I = \mathbb{E}[f] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{23.19}$$

23.4.1 Basic idea

The idea is to draw samples $\mathbf{x}$ in regions which have high probability, $p(\mathbf{x})$, but also where $|f(\mathbf{x})|$ is large. The result can be **super efficient**, meaning it needs less samples than if we were to sample from the exact distribution $p(\mathbf{x})$. The reason is that the samples are focussed on the important parts of space. For example, suppose we want to estimate the probability of a **rare event**. Define $f(\mathbf{x}) = \mathbb{I}(\mathbf{x} \in E)$, for some set E. Then it is better to sample from a proposal of the form $q(\mathbf{x}) \propto f(\mathbf{x}) p(\mathbf{x})$ than to sample from $p(\mathbf{x})$ itself.

Importance sampling samples from any proposal, $q(\mathbf{x})$. It then uses these samples to estimate the integral as follows:

$$\mathbb{E}[f] = \int f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{1}{S} \sum_{s=1}^{S} w_s f(\mathbf{x}^s) = \hat{I} \tag{23.20}$$

where $w_s \triangleq \frac{p(\mathbf{x}^s)}{q(\mathbf{x}^s)}$ are the **importance weights**. Note that, unlike rejection sampling, we use all the samples.

How should we choose the proposal? A natural criterion is to minimize the variance of the estimate $\hat{I} = \sum_s w_s f(\mathbf{x}^s)$. Now

$$\text{var}_{q(\mathbf{x})} [f(\mathbf{x}) w(\mathbf{x})] = \mathbb{E}_{q(\mathbf{x})} \left[f^2(\mathbf{x}) w^2(\mathbf{x}) \right] - I^2 \tag{23.21}$$

Since the last term is independent of q, we can ignore it. By Jensen's inequality, we have the following lower bound:

$$\mathbb{E}_{q(\mathbf{x})} \left[f^2(\mathbf{x}) w^2(\mathbf{x}) \right] \geq (\mathbb{E}_{q(\mathbf{x})} [|f(\mathbf{x}) w(\mathbf{x})|])^2 = \left(\int |f(\mathbf{x})| p(\mathbf{x}) d\mathbf{x} \right)^2 \tag{23.22}$$

The lower bound is obtained when we use the optimal importance distribution:

$$q^*(\mathbf{x}) = \frac{|f(\mathbf{x})| p(\mathbf{x})}{\int |f(\mathbf{x}')| p(\mathbf{x}') d\mathbf{x}'} \tag{23.23}$$

When we don't have a particular target function $f(\mathbf{x})$ in mind, we often just try to make $q(\mathbf{x})$ as close as possible to $p(\mathbf{x})$. In general, this is difficult, especially in high dimensions, but it is possible to adapt the proposal distribution to improve the approximation. This is known as **adaptive importance sampling** (Oh and Berger 1992).

23.4.2 Handling unnormalized distributions

It is frequently the case that we can evaluate the unnormalized target distribution, $\tilde{p}(\mathbf{x})$, but not its normalization constant, Z_p. We may also want to use an unnormalized proposal, $\tilde{q}(\mathbf{x})$, with possibly unknown normlization constant Z_q. We can do this as follows. First we evaluate

$$\mathbb{E}[f] = \frac{Z_q}{Z_p} \int f(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{Z_q}{Z_p} \frac{1}{S} \sum_{s=1}^{S} \tilde{w}_s f(\mathbf{x}^s) \tag{23.24}$$

where $\tilde{w}_s \triangleq \frac{\tilde{p}(\mathbf{x}^s)}{\tilde{q}(\mathbf{x}^s)}$ is the unnormalized importance weight. We can use the same set of samples to evaluate the ratio Z_p / Z_q as follows:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q} \int \tilde{p}(\mathbf{x}) d\mathbf{x} = \int \frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{1}{S} \sum_{s=1}^{S} \tilde{w}_s \tag{23.25}$$

Hence

$$\hat{I} = \frac{\frac{1}{S} \sum_s \tilde{w}_s f(\mathbf{x}^s)}{\frac{1}{S} \sum_s \tilde{w}_s} = \sum_{s=1}^{S} w_s f(\mathbf{x}^s) \tag{23.26}$$

where

$$w_s \triangleq \frac{\tilde{w}_s}{\sum_{s'} \tilde{w}_{s'}} \tag{23.27}$$

are the normalized importance weights. The resulting estimate is a ratio of two estimates, and hence is biased. However, as $S \to \infty$, we have that $\hat{I} \to I$, under weak assumptions (see e.g., (Robert and Casella 2004) for details).

23.4.3 Importance sampling for a DGM: likelihood weighting

We now describe a way to use importance sampling to generate samples from a distribution which can be represented as a directed graphical model (Chapter 10).

If we have no evidence, we can sample from the unconditional joint distribution of a DGM $p(\mathbf{x})$ as follows: first sample the root nodes, then sample their children, then sample their children, etc. This is known as **ancestral sampling**. It works because, in a DAG, we can always topologically order the nodes so that parents preceed children. (Note that there is no equivalent easy method for sampling from an unconditional *undirected* graphical model.)

Now suppose we have some evidence, so some nodes are "clamped" to observed values, and we want to sample from the posterior $p(\mathbf{x}|\mathcal{D})$. If all the variables are discrete, we can use the following simple procedure: perform ancestral sampling, but as soon as we sample a value that is inconsistent with an observed value, reject the whole sample and start again. This is known as **logic sampling** (Henrion 1988).

Needless to say, logic sampling is very inefficient, and it cannot be applied when we have real-valued evidence. However, it can be modified as follows. Sample unobserved variables as before, conditional on their parents. But don't sample observed variables; instead we just use their observed values. This is equivalent to using a proposal of the form

$$q(\mathbf{x}) = \prod_{t \notin E} p(x_t|\mathbf{x}_{\text{pa}(t)}) \prod_{t \in E} \delta_{x_t^*}(x_t) \tag{23.28}$$

where E is the set of observed nodes, and x_t^* is the observed value for node t. We should therefore give the overall sample an importance weight as follows:

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{q(\mathbf{x})} = \prod_{t \notin E} \frac{p(x_t|\mathbf{x}_{\text{pa}(t)})}{p(x_t|\mathbf{x}_{\text{pa}(t)})} \prod_{t \in E} \frac{p(x_t|\mathbf{x}_{\text{pa}(t)})}{1} = \prod_{t \in E} p(x_t|\mathbf{x}_{\text{pa}(t)}) \tag{23.29}$$

This technique is known as **likelihood weighting** (Fung and Chang 1989; Shachter and Peot 1989).

23.4.4 Sampling importance resampling (SIR)

We can draw unweighted samples from $p(x)$ by first using importance sampling (with proposal q) to generate a distribution of the form

$$p(\mathbf{x}) \approx \sum_s w_s \delta_{\mathbf{x}^s}(\mathbf{x}) \tag{23.30}$$

where w_s are the normalized importance weights. We then sample with replacement from Equation 23.30, where the probability that we pick $\mathbf{x}^s$ is w_s. Let this procedure induce a distribution denoted by $\hat{p}$. To see that this is valid, note that

$$\begin{aligned} \hat{p}(x \le x_0) &= \sum_s \mathbb{I}(x^s \le x_0) w_s = \frac{\sum_s \mathbb{I}(x^s \le x_0)\tilde{p}(x^s)/q(x^s)}{\sum_s \tilde{p}(x^s)/q(x^s)} && (23.31) \\ &\to \frac{\int \mathbb{I}(x \le x_0)\frac{\tilde{p}(x)}{q(x)} q(x) dx}{\int \frac{\tilde{p}(x)}{q(x)} q(x) dx} && (23.32) \\ &= \frac{\int \mathbb{I}(x \le x_0)\tilde{p}(x)dx}{\int \tilde{p}(x)dx} = \int \mathbb{I}(x \le x_0) p(x) dx = p(x \le x_0) && (23.33) \end{aligned}$$

This is known as **sampling importance resampling** (SIR) (Rubin 1998). The result is an unweighted approximation of the form

$$p(\mathbf{x}) \approx \frac{1}{S'} \sum_{s=1}^{S'} \delta_{\mathbf{x}^s}(\mathbf{x}) \tag{23.34}$$

Note that we typically take $S' \ll S$.

This algorithm can be used to perform Bayesian inference in low-dimensional settings (Smith and Gelfand 1992). That is, suppose we want to draw (unweighted) samples from the posterior, $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. We can use importance sampling with $\tilde{p}(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ as the unnormalized posterior, and $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ as our proposal. The normalized weights have the form

$$w_s = \frac{\tilde{p}(\boldsymbol{\theta}_s)/q(\boldsymbol{\theta}_s)}{\sum_{s'} \tilde{p}(\boldsymbol{\theta}_{s'})/q(\boldsymbol{\theta}_{s'})} = \frac{p(\mathcal{D}|\boldsymbol{\theta}_s)}{\sum_{s'} p(\mathcal{D}|\boldsymbol{\theta}_{s'})} \tag{23.35}$$

We can then use SIR to sample from $p(\boldsymbol{\theta}|\mathcal{D})$.

Of course, if there is a big discrepancy between our proposal (the prior) and the target (the posterior), we will need a huge number of importance samples for this technique to work reliably, since otherwise the variance of the importance weights will be very large, implying that most samples carry no useful information. (This issue will come up again in Section 23.5, when we discuss particle filtering.)

23.5 Particle filtering

Particle filtering (PF) is a Monte Carlo, or **simulation based**, algorithm for recursive Bayesian inference. That is, it approximates the predict-update cycle described in Section 18.3.1. It is very widely used in many areas, including tracking, time-series forecasting, online parameter learning, etc. We explain the basic algorithm below. For a book-length treatment, see (Doucet et al. 2001); for a good tutorial, see (Arulampalam et al. 2002), or just read on.

23.5.1 Sequential importance sampling

The basic idea is to appproximate the belief state (of the entire state trajectory) using a weighted set of particles:

$$p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_{1:t}^s}(\mathbf{z}_{1:t}) \tag{23.36}$$

where $\hat{w}_t^s$ is the normalized weight of sample s at time t. From this representation, we can easily compute the marginal distribution over the most recent state, $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, by simply ignoring the previous parts of the trajectory, $\mathbf{z}_{1:t-1}$. (The fact that PF samples in the space of entire trajectories has various implications which we will discuss later.)

We update this belief state using importance sampling. If the proposal has the form $q(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})$, then the importance weights are given by

$$w_t^s \propto \frac{p(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})}{q(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})} \tag{23.37}$$

which can be normalized as follows:

$$\hat{w}_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}} \tag{23.38}$$

We can rewrite the numerator recursively as follows:

$$\begin{aligned} p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_t|\mathbf{z}_{1:t}, \mathbf{y}_{1:t-1})p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} & (23.39)\\ &= \frac{p(\mathbf{y}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1})p(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} & (23.40)\\ &\propto p(\mathbf{y}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1}) & (23.41) \end{aligned}$$

where we have made the usual Markov assumptions. We will restrict attention to proposal densities of the following form:

$$q(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) = q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t})q(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1}) \tag{23.42}$$

so that we can “grow” the trajectory by adding the new state $\mathbf{z}_t$ to the end. In this case, the importance weights simplify to

$$\begin{aligned} w_t^s &\propto \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)p(\mathbf{z}_{1:t-1}^s|\mathbf{y}_{1:t-1})}{q(\mathbf{z}_t^s|\mathbf{z}_{1:t-1}^s, \mathbf{y}_{1:t})q(\mathbf{z}_{1:t-1}^s|\mathbf{y}_{1:t-1})} & (23.43)\\ &= w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{1:t-1}^s, \mathbf{y}_{1:t})} & (23.44) \end{aligned}$$

If we further assume that $q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) = q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)$, then we only need to keep the most recent part of the trajectory and observation sequence, rather than the whole history, in order to compute the new sample. In this case, the weight becomes

$$w_t^s \propto w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)} \tag{23.45}$$

Hence we can approximate the posterior filtered density using

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_t^s}(\mathbf{z}_t) \tag{23.46}$$

As $S \to \infty$, one can show that this approaches the true posterior (Crisan et al. 1999).

The basic algorithm is now very simple: for each old sample s, propose an extension using $\mathbf{z}_t^s \sim q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$, and give this new particle weight w_t^s using Equation 23.45. Unfortunately, this basic algorithm does not work very well, as we discuss below.

23.5.2 The degeneracy problem

The basic sequential importance sampling algorithm fails after a few steps because most of the particles will have negligible weight. This is called the **degeneracy problem**, and occurs because we are sampling in a high-dimensional space (in fact, the space is growing in size over time), using a myopic proposal distribution.

We can quantify the degree of degeneracy using the **effective sample size**, defined by

$$S_{\text{eff}} \triangleq \frac{S}{1 + \text{var}\left[w_t^{*s}\right]} \tag{23.47}$$

where $w_t^{*s} = p(\mathbf{z}_t^s|\mathbf{y}_{1:t})/q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ is the "true weight" of particle s. This quantity cannot be computed exactly, since we don't know the true posterior, but we can approximate it using

$$\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^{S}(w_t^s)^2} \tag{23.48}$$

If the variance of the weights is large, then we are wasting our resources updating particles with low weight, which do not contribute much to our posterior estimate.

There are two main solutions to the degeneracy problem: adding a resampling step, and using a good proposal distribution. We discuss both of these in turn.

23.5.3 The resampling step

The main improvement to the basic SIS algorithm is to monitor the effective sampling size, and whenever it drops below a threshold, to eliminate particles with low weight, and then to create replicates of the surviving particles. (Hence PF is sometimes called **survival of the fittest** (Kanazawa et al. 1995).) In particular, we generate a new set $\{\mathbf{z}_t^{s*}\}_{s=1}^S$ by sampling with replacement S times from the weighted distribution

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_t^s}(\mathbf{z}_t) \tag{23.49}$$

where the probability of choosing particle j for replication is w_t^j. (This is sometimes called **rejuvenation**.) The result is an iid *unweighted* sample from the discrete density Equation 23.49, so we set the new weights to $w_t^s = 1/S$. This scheme is illustrated in Figure 23.4.

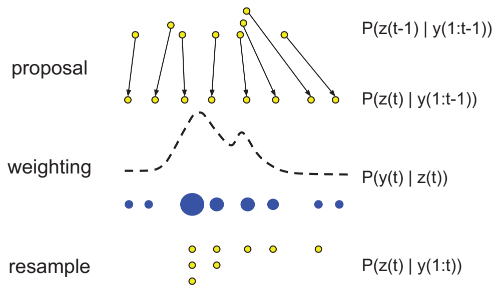


Figure 23.4 Illustration of particle filtering.

There are a variety of algorithms for peforming the resampling step. The simplest is **multinomial resampling**, which computes

$$(K_1, \ldots, K_S) \sim \text{Mu}(S, (w_t^1, \ldots, w_t^S)) \tag{23.50}$$

We then make K_s copies of $\mathbf{z}_t^s$. Various improvements exist, such as **systematic resampling residual resampling**, and **stratified sampling**, which can reduce the variance of the weights. All these methods take $O(S)$ time. See (Doucet et al. 2001) for details.

The overall particle filtering algorithm is summarized in Algorithm 6. (Note that if an estimate of the state is required, it should be computed before the resampling step, since this will result in lower variance.)

Algorithm 23.1: One step of a generic particle filter

```
1 for $s = 1 : S$ do
2     Draw $\mathbf{z}_t^s \sim q(\mathbf{z}_t | \mathbf{z}_{t-1}^s, \mathbf{y}_t)$ ;
3     Compute weight $w_t^s \propto w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)}$ ;
4 Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$ ;
5 Compute $\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^S (w_t^s)^2}$;
6 if $\hat{S}_{eff} < S_{min}$ then
7     Resample $S$ indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
8     $\mathbf{z}_t^{:} = \mathbf{z}_t^{\boldsymbol{\pi}}$;
9     $w_t^s = 1/S$ ;
```

Although the resampling step helps with the degeneracy problem, it introduces problems of its own. In particular, since the particles with high weight will be selected many times, there is a loss of diversity amongst the population. This is known as **sample impoverishment**. In the extreme case of no process noise (e.g., if we have static but unknown parameters as part of the state space), then all the particles will collapse to a single point within a few iterations.

To mitigate this problem, several solutions have been proposed. (1) Only resample when necessary, not at every time step. (The original **bootstrap filter** (Gordon 1993) resampled at every step, but this is suboptimal.) (2) After replicating old particles, sample new values using an MCMC step which leaves the posterior distribution invariant (see e.g., the **resample-move** algorithm in (Gilks and Berzuini 2001)). (3) Create a kernel density estimate on top of the particles,

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} w_t^s \kappa(\mathbf{z}_t - \mathbf{z}_t^s) \tag{23.51}$$

where κ is some smoothing kernel. We then sample from this smoothed distribution. This is known as a **regularized particle filter** (Musso et al. 2001). (4) When performing inference on static parameters, add some artificial process noise. (If this is undesirable, other algorithms must be used for online parameter estimation, e.g., (Andrieu et al. 2005)).

23.5.4 The proposal distribution

The simplest and most widely used proposal distribution is to sample from the prior:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = p(\mathbf{z}_t|\mathbf{z}_{t-1}^s) \tag{23.52}$$

In this case, the weight update simplifies to

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{z}_t^s) \tag{23.53}$$

This can be thought of a "generate and test" approach: we sample values from the dynamic model, and then evaluate how good they are after we see the data (see Figure 23.4). This is the approach used in the **condensation** algorithm (which stands for "conditional density propagation") used for visual tracking (Isard and Blake 1998). However, if the likelihood is narrower than the dynamical prior (meaning the sensor is more informative than the motion model, which is often the case), this is a very inefficient approach, since most particles will be assigned very low weight.

It is much better to actually look at the data $\mathbf{y}_t$ when generating a proposal. In fact, the optimal proposal distribution has the following form:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = \frac{p(\mathbf{y}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1}^s)}{p(\mathbf{y}_t|\mathbf{z}_{t-1}^s)} \tag{23.54}$$

If we use this proposal, the new weight is given by

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{z}_{t-1}^s) = w_{t-1}^s \int p(\mathbf{y}_t|\mathbf{z}_t')p(\mathbf{z}_t'|\mathbf{z}_{t-1}^s)d\mathbf{z}_t' \tag{23.55}$$

This proposal is optimal since, for any given $\mathbf{z}_{t-1}^s$, the new weight w_t^s takes the same value regardless of the value drawn for $\mathbf{z}_t^s$. Hence, conditional on the old values $\mathbf{z}_{\cdot t-1}$, the variance of true weights $\text{var}\left[w_t^{*s}\right]$, is zero.

In general, it is intractable to sample from $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ and to evaluate the integral needed to compute the predictive density $p(\mathbf{y}_t|\mathbf{z}_{t-1}^s)$. However, there are two cases when the optimal proposal distribution can be used. The first setting is when $\mathbf{z}_t$ is discrete, so the integral becomes a sum. Of course, if the entire state space is discrete, we can use an HMM filter instead, but in some cases, some parts of the state are discrete, and some continuous. The second setting is when $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ is Gaussian. This occurs when the dynamics are nonlinear but the observations are linear. See Exercise 23.3 for the details.

In cases where the model is not linear-Gaussian, we may still compute a Gaussian approximation to $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ using the unscented transform (Section 18.5.2) and use this as a proposal. This is known as the **unscented particle filter** (van der Merwe et al. 2000). In more general settings, we can use other kinds of **data-driven proposals**, perhaps based on discriminative models. Unlike MCMC, we do not need to worry about the proposals being reversible.

23.5.5 Application: robot localization

Consider a mobile robot wandering around an office environment. We will assume that it already has a map of the world, represented in the form of an **occupancy grid**, which just specifies whether each grid cell is empty space or occupied by an something solid like a wall. The goal is for the robot to estimate its location. This can be solved optimally using an HMM filter, since we are assuming the state space is discrete. However, since the number of states, K, is often very large, the $O(K^2)$ time complexity per update is prohibitive. We can use a particle filter as a sparse approximation to the belief state. This is known as **Monte Carlo localization**, and is described in detail in (Thrun et al. 2006).

Figure 23.5 gives an example of the method in action. The robot uses a sonar range finder, so it can only sense distance to obstacles. It starts out with a uniform prior, reflecting the fact that the owner of the robot may have turned it on in an arbitrary location. (Figuring out where you are, starting from a uniform prior, is called **global localization**.) After the first scan, which indicates two walls on either side, the belief state is shown in (b). The posterior is still fairly broad, since the robot could be in any location where the walls are fairly close by, such as a corridor or any of the narrow rooms. After moving to location 2, the robot is pretty sure it must be in the corridor, as shown in (c). After moving to location 3, the sensor is able to detect the end of the corridor. However, due to symmetry, it is not sure if it is in location I (the true location) or location II. (This is an example of **perceptual aliasing**, which refers to the fact that different things may look the same.) After moving to locations 4 and 5, it is finally able to figure out precisely where it is. The whole process is analogous to someone getting lost in an office building, and wandering the corridors until they see a sign they recognize.

In Section 23.6.3, we discuss how to estimate location and the map at the same time.

23.5.6 Application: visual object tracking

Our next example is concerned with tracking an object (in this case, a remote-controlled helicopter) in a video sequence. The method uses a simple linear motion model for the centroid of the object, and a color histogram for the likelihood model, using **Bhattacharya distance** to compare histograms. The proposal distribution is obtained by sampling from the likelihood. See (Nummiaro et al. 2003) for further details.

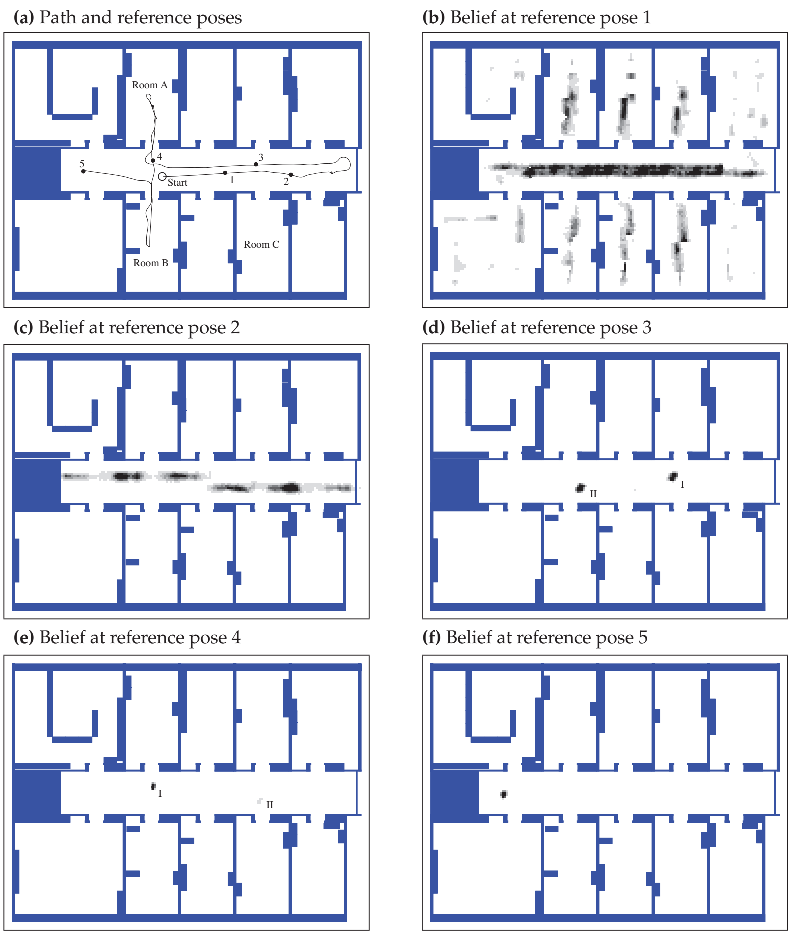
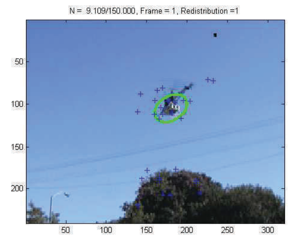
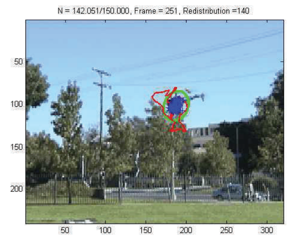

(a) Path and reference poses
Room A
Room B
Room C
Start
1
2
3
4
5
(b) Belief at reference pose 1
(c) Belief at reference pose 2
(d) Belief at reference pose 3
I
II
(e) Belief at reference pose 4
I
II
(f) Belief at reference pose 5

Figure 23.5 Illustration of Monte Carlo localization. Source: Figure 8.7 of (Thrun et al. 2006). Used with kind permission of Sebastian Thrun.

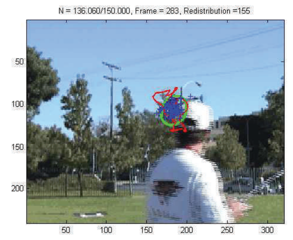
Figure 23.6 shows some example frames. The system uses $S = 250$ particles, with an effective sample size of $\hat{S}_{\text{eff}} = 134$. (a) shows the belief state at frame 1. The system has had to resample 5 times to keep the effective sample size above the threshold of 150; (b) shows the belief state at frame 251; the red lines show the estimated location of the center of the object over the last 250 frames. (c) shows that the system can handle visual clutter, as long as it does not have the same color as the target object. (d) shows that the system is confused between the grey of the helicopter and the grey of the building. The posterior is bimodal. The green ellipse, representing the posterior mean and covariance, is in between the two modes. (e) shows that the probability mass has shifted to the wrong mode: the system has lost track. (f) shows the particles spread out over the gray building; recovery of the object is very unlikely from this state using this



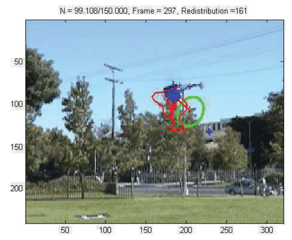
(a)



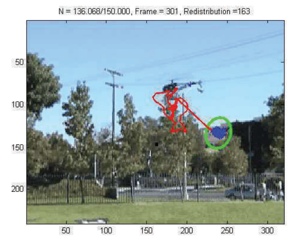
(b)



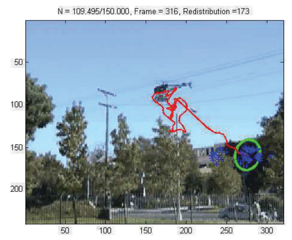
(c)



(d)



(e)



(f)

Figure 23.6 Example of particle filtering applied to visual object tracking, based on color histograms. (a-c) succesful tracking: green ellipse is on top of the helicopter. (d-f): tracker gets distracted by gray clutter in the background. See text for details. Figure generated by `pfColorTrackerDemo`, written by Sebastien Paris.

proposal.

We see that the method is able to keep track for a fairly long time, despite the presence of clutter. However, eventually it loses track of the object. Note that since the algorithm is stochastic, simply re-running the demo may fix the problem. But in the real world, this is not an option. The simplest way to improve performance is to use more particles. An alternative is to perform **tracking by detection**, by running an object detector over the image every few frames. See (Forsyth and Ponce 2002; Szeliski 2010; Prince 2012) for details.

23.5.7 Application: time series forecasting

In Section 18.2.4, we discussed how to use the Kalman filter to perform time series forecasting. This assumes that the model is a linear-Gaussian state-space model. There are many models which are either non-linear and/or non-Gaussian. For example, **stochastic volatility** models, which are widely used in finance, assume that the variance of the system and/or observation noise changes over time. Particle filtering is widely used in such settings. See e.g., (Doucet et al. 2001) and references therein for details.

23.6 Rao-Blackwellised particle filtering (RBPF)

In some models, we can partition the hidden variables into two kinds, $\mathbf{q}_t$ and $\mathbf{z}_t$, such that we can analytically integrate out $\mathbf{z}_t$ provided we know the values of $\mathbf{q}_{1:t}$. This means we only have sample $\mathbf{q}_{1:t}$, and can represent $p(\mathbf{z}_t|\mathbf{q}_{1:t})$ parametrically. Thus each particle s represents a value for $\mathbf{q}^s_{1:t}$ and a distribution of the form $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t})$. These hybrid particles are are sometimes called **distributional particles** or **collapsed particles** (Koller and Friedman 2009, Sec 12.4).

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. Hence this technique is known as **Rao-Blackwellised particle filtering** or **RBPF** for short, named after Theorem 24.20. The method is best explained using a specific example.

23.6.1 RBPF for switching LG-SSMs

A canonical example for which RBPF can be applied is the switching linear dynamical system (SLDS) model discussed in Section 18.6 (Chen and Liu 2000; Doucet et al. 2001). We can represent $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t})$ using a mean and covariance matrix for each particle s, where $q_t \in \{1, \ldots, K\}$.

If we propose from the prior, $q(q_t = k|q^s_{t-1})$, the weight update becomes

$$w^s_t \propto w^s_{t-1} p(\mathbf{y}_t|q_t = k, \mathbf{q}^s_{1:t-1}, \mathbf{y}_{1:t-1}) = w^s_{t-1} L^s_{t,k} \tag{23.56}$$

where

$$L^s_{tk} = \int p(\mathbf{y}_t|q_t = k, \mathbf{z}_t, \cancel{\mathbf{y}_{1:t-1}}, \cancel{\mathbf{q}^s_{1:t-1}}) p(\mathbf{z}_t|q_t = k, \mathbf{y}_{1:t-1} \mathbf{q}^s_{1:t-1},) d\mathbf{z}_t \tag{23.57}$$

The quantity L^s_{tk} is the predictive density for the new observation $\mathbf{y}_t$ conditioned on $q_t = k$ and the history $\mathbf{q}^s_{1:t-1}$. In the case of SLDS models, this can be computed using the normalization constant of the Kalman filter, Equation 18.41.

We give some pseudo-code in Algorithm 8. (The step marked "KFupdate" refers to the Kalman filter update equations in Section 18.3.1.) This is known as a **mixture of Kalman filters**.

If K is small, we can compute the optimal proposal distribution, which is

$$p(q_t = k|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t-1}) = \hat{p}^s_{t-1}(q_t = k|\mathbf{y}_t) \tag{23.58}$$

$$= \frac{\hat{p}^s_{t-1}(\mathbf{y}_t|q_t = k)\hat{p}^s_{t-1}(q_t = k)}{\hat{p}^s_{t-1}(\mathbf{y}_t)} \tag{23.59}$$

$$= \frac{L^s_{tk} p(q_t = k|q^s_{t-1})}{\sum_{k'} L^s_{tk'} p(q_t = k'|q^s_{t-1})} \tag{23.60}$$

Algorithm 23.2: One step of RBPF for SLDS using prior as proposal

```
1  for $s = 1 : S$ do
2      $k \sim p(q_t | q_{t-1}^s)$ ;
3      $q_t^s := k$;
4      $(\boldsymbol{\mu}_t^s, \boldsymbol{\Sigma}_t^s, L_{tk}^s) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^s, \boldsymbol{\Sigma}_{t-1}^s, \mathbf{y}_t, \boldsymbol{\theta}_k)$;
5      $w_t^s = w_{t-1}^s L_{ts}^k$;
6  Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$ ;
7  Compute $\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^S (w_t^s)^2}$;
8  if $\hat{S}_{eff} < S_{min}$ then
9      Resample $S$ indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
10     $\mathbf{q}_t^: = \mathbf{q}_t^{\boldsymbol{\pi}}$, $\boldsymbol{\mu}_t^: = \boldsymbol{\mu}_t^{\boldsymbol{\pi}}$, $\boldsymbol{\Sigma}_t^: = \boldsymbol{\Sigma}_t^{\boldsymbol{\pi}}$, ;
11     $w_t^s = 1/S$ ;
```

where we use the following shorthand:

$$\hat{p}_{t-1}^s(\cdot) = p(\cdot | \mathbf{y}_{1:t-1}, \mathbf{q}_{1:t-1}^s) \tag{23.61}$$

We then sample from $p(q_t | \mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t})$ and give the resulting particle weight

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t | \mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t-1}) = w_{t-1}^s \sum_k \left[L_{tk}^s p(q_t = k | q_{t-1}^s) \right] \tag{23.62}$$

Since the weights of the particles in Equation 23.62 are independent of the new value that is actually sampled for q_t, we can compute these weights first, and use them to decide which particles to propagate. That is, we choose the fittest particles at time $t-1$ using information from time t. This is called **look-ahead RBPF** (de Freitas et al. 2004).

In more detail, the idea is this. We pass each sample in the prior through all K models to get K posteriors, one per sample. The normalization constants of this process allow us to compute the optimal weights in Equation 23.62. We then resample S indices. Finally, for each old particle s that is chosen, we sample one new state $q_t^s = k$, and use the corresponding posterior from the K possible alternative that we have already computed. The pseudo-code is shown in Algorithm 7. This method needs $O(KS)$ storage, but has the advantage that each particle is chosen using the latest information, $\mathbf{y}_t$.

A further improvement can be obtained by exploiting the fact that the state space is discrete. Hence we can use the resampling method of (Fearnhead 2004) which avoids duplicating particles.

23.6.2 Application: tracking a maneuvering target

One application of SLDS is to track moving objects that have piecewise linear dynamics. For example, suppose we want to track an airplane or missile; q_t can specify if the object is flying normally or is taking evasive action. This is called **maneuvering target tracking**.

Figure 23.7 gives an example of an object moving in 2d. The setup is essentially the same as in Section 18.2.1, except that we add a three-state discrete Markov chain which controls the

Algorithm 23.3: One step of look-ahead RBPF for SLDS using optimal proposal

```
1  for $s = 1 : S$ do
2      for $k = 1 : K$ do
3          $(\boldsymbol{\mu}_{tk}^s, \boldsymbol{\Sigma}_{tk}^s, L_{ts}^k) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^s, \boldsymbol{\Sigma}_{t-1}^s, \mathbf{y}_t, \boldsymbol{\theta}_k)$;
4      $w_t^s = w_{t-1}^s[\sum_k L_{ts}^k p(q_t = k|q_{t-1}^s)]$;
5  Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$ ;
6  Resample $S$ indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
7  for $s \in \boldsymbol{\pi}$ do
8      Compute optimal proposal $p(k|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t}) = \frac{L_{tk}^s p(q_t=k|q_{t-1}^s)}{\sum_{k'} L_{tk}^s p(q_t=k|q_{t-1}^s)}$;
9      Sample $k \sim p(k|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t})$;
10     $q_t^s = k$, $\boldsymbol{\mu}_t^s = \boldsymbol{\mu}_{tk}^s$, $\boldsymbol{\Sigma}_t^s = \boldsymbol{\Sigma}_{tk}^s$;
11     $w_t^s = 1/S$;
```

Method	misclassification rate	MSE	Time (seconds)
PF	0.440	21.051	6.086
RBPF	0.340	18.168	10.986

Table 23.1 Comparison of PF an RBPF on the maneuvering target problem in Figure 23.7.

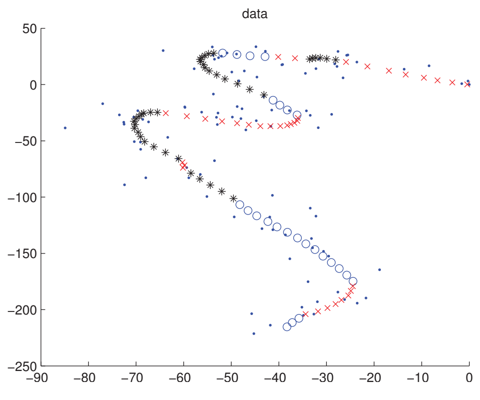
input to the system. We define $\mathbf{u}_t = 1$ and set

$$\mathbf{B}_1 = (0, 0, 0, 0)^T, \mathbf{B}_2 = (-1.225, -0.35, 1.225, 0.35)^T, \mathbf{B}_3 = (1.225, 0.35, -1.225, -0.35)^T$$

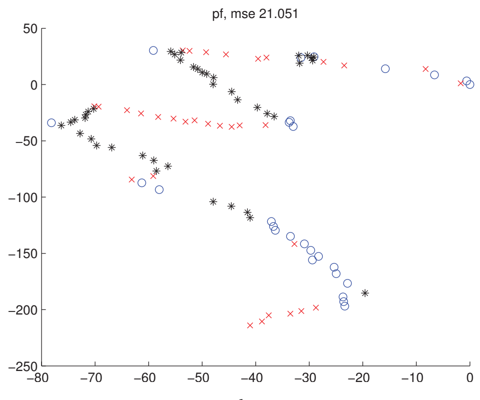
so the system will turn in different directions depending on the discrete state.

Figure 23.7(a) shows the true state of the system from a sample run, starting at $(0, 0)$: the colored symbols denote the discrete state, and the location of the symbol denotes the (x, y) location. The small dots represent noisy observations. Figure 23.7(b) shows the estimate of the state computed using particle filtering with 500 particles, where the proposal is to sample from the prior. The colored symbols denote the MAP estimate of the state, and the location of the symbol denotes the MMSE (minimum mean square error) estimate of the location, which is given by the posterior mean. Figure 23.7(c) shows the estimate computing using RBPF with 500 particles, using the optimal proposal distribution. A more quantitative comparison is shown in Table 23.1. We see that RBPF has slightly better performance, although it is also slightly slower.

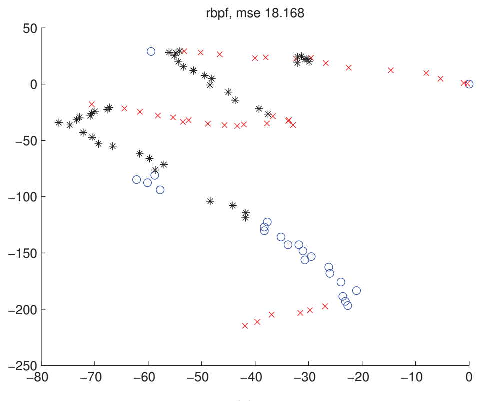
Figure 23.8 visualizes the belief state of the system. In (a) we show the distribution over the discrete states. We see that the particle filter estimate of the belief state (second column) is not as accurate as the RBPF estimate (third column) in the beginning, although after the first few observations performance is similar for both methods. In (b), we plot the posterior over the x locations. For simplicity, we use the PF estimate, which is a set of weighted samples, but we could also have used the RBPF estimate, which is a set of weighted Gaussians.



(a)



(b)

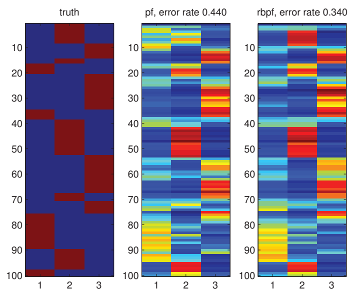


(c)

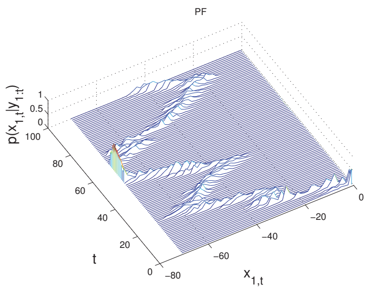
Figure 23.7 (a) A maneuvering target. The colored symbols represent the hidden discrete state. (b) Particle filter estimate. (c) RBPF estimate. Figure generated by `rbpfManeuverDemo`, based on code by Nando de Freitas.

23.6.3 Application: Fast SLAM

In Section 18.2.2, we introduced the problem of simultaneous localization and mapping or SLAM for mobile robotics. The main problem with the Kalman filter implementation is that it is cubic in the number of landmarks. However, by looking at the DGM in Figure 18.2, we see that, conditional on knowing the robot's path, $\mathbf{q}_{1:t}$, where $\mathbf{q}_t \in \mathbb{R}^2$, the landmark locations $\mathbf{z} \in \mathbb{R}^{2L}$ are independent. (We assume the landmarks don't move, so we drop the t subscript). That is, $p(\mathbf{z}|\mathbf{q}_{1:t}, \mathbf{y}_{1:t}) = \prod_{l=1}^{L} p(\mathbf{z}_l|\mathbf{q}_{1:t}, \mathbf{y}_{1:t})$. Consequently we can use RBPF, where we sample the robot's trajectory, $\mathbf{q}_{1:t}$, and we run L independent 2d Kalman filters inside each particle. This takes $O(L)$ time per particle. Fortunately, the number of particles needed for good performance is quite small (this partly depends on the control / exploration policy), so the algorithm is essentially linear in the number of particles. This technique has the additional advantage that



(a)



(b)

Figure 23.8 Belief states corresponding to Figure 23.7. (a) Discrete state. The system starts in state 2 (red x in Figure 23.7), then moves to state 3 (black * in Figure 23.7), returns briefly to state 2, then switches to state 1 (blue circle in Figure 23.7), etc. (b) Horizontal location (PF estimate). Figure generated by `rbpfManeuverDemo`, based on code by Nando de Freitas.

it is easy to use sampling to handle the data association ambiguity, and that it allows for other representations of the map, such as occupancy grids. This idea was first suggested in (Murphy 2000), and was subsequently extended and made practical in (Thrun et al. 2004), who christened the technique **FastSLAM**. See `rbpfSlamDemo` for a simple demo in a discrete grid world.

Exercises

Exercise 23.1 Sampling from a Cauchy

Show how to use inverse probability transform to sample from a standard Cauchy, $\mathcal{T}(x|0, 1, 1)$.

Exercise 23.2 Rejection sampling from a Gamma using a Cauchy proposal

Show how to use a Cauchy proposal to perform rejection sampling from a Gamma distribution. Derive the optimal constant M, and plot the density and its upper envelope.

Exercise 23.3 Optimal proposal for particle filtering with linear-Gaussian measurement model

Consider a state-space model of the following form:

$$\mathbf{z}_t = f_t(\mathbf{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_{t-1}) \quad (23.63)$$
$$\mathbf{y}_t = \mathbf{H}_t\mathbf{z}_t + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (23.64)$$

Derive expressions for $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)$ and $p(\mathbf{y}_t|\mathbf{z}_{t-1})$, which are needed to compute the optimal (minimum variance) proposal distribution. Hint: use Bayes rule for Gaussians.

24 *Markov chain Monte Carlo (MCMC) inference*

24.1 Introduction

In Chapter 23, we introduced some simple Monte Carlo methods, including rejection sampling and importance sampling. The trouble with these methods is that they do not work well in high dimensional spaces. The most popular method for sampling from high-dimensional distributions is **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 17.2) on the state space $\mathcal{X}$ whose stationary distribution is the target density $p^*(\mathbf{x})$ of interest (this may be a prior or a posterior). 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 $\mathbf{x}$ is proportional to $p^*(\mathbf{x})$. By drawing (correlated!) samples $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots,$ from the chain, we can perform Monte Carlo integration wrt p^*. We give the details below.

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 (Metropolis et al. 1953) in a chemistry journal. An extension was published in the statistics literature in (Hastings 1970), but was largely unnoticed. A special case (Gibbs sampling, Section 24.2) was independently invented in 1984 in the context of Ising models and was published in (Geman and Geman 1984). But it was not until (Gelfand and Smith 1990) 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.

It is worth briefly comparing MCMC to variational inference (Chapter 21). The advantages of variational inference are (1) for small to medium problems, it is usually faster; (2) it is deterministic; (3) is it easy to determine when to stop; (4) it often provides a lower bound on the log likelihood. The advantages of sampling are: (1) it is often easier to implement; (2) it is applicable to a broader range of models, such as models whose size or structure changes depending on the values of certain variables (e.g., as happens in matching problems), or models without nice conjugate priors; (3) sampling can be faster than variational methods when applied to really huge models or datasets.[2]

1. Source: `http://www.siam.org/pdf/news/637.pdf`.
2. The reason is that sampling passes specific values of variables (or sets of variables), whereas in variational inference, we pass around distributions. Thus sampling passes sparse messages, whereas variational inference passes dense messages For comparisons of the two approaches, see e.g., (Yoshida and West 2010) and articles in (Bekkerman et al.

24.2 Gibbs sampling

In this section, we present one of the most popular MCMC algorithms, known as **Gibbs sampling**.[3] (In physics, this method is known as **Glauber dynamics** or the **heat bath** method.) This is the MCMC analog of coordinate descent.

24.2.1 Basic idea

The idea behind Gibbs sampling is that we sample each variable in turn, conditioned on the values of all the other variables in the distribution. That is, given a joint sample $\mathbf{x}^s$ of all the variables, we generate a new sample $\mathbf{x}^{s+1}$ by sampling each component in turn, based on the most recent values of the other variables. 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. If x_i is a visible variable, we do not sample it, since its value is already known.

The expression $p(x_i|\mathbf{x}_{-i})$ is called the **full conditional** for variable i. In general, x_i may only depend on some of the other variables. If we represent $p(\mathbf{x})$ as a graphical model, we can infer the dependencies by looking at i's Markov blanket, which are its neighbors in the graph. Thus to sample x_i, we only need to know the values of i's neighbors. In this sense, Gibbs sampling is a distributed algorithm. However, it is not a parallel algorithm, since the samples must be generated sequentially.

For reasons that we will explain in Section 24.4.1, it is necessary to discard some of the initial samples until the Markov chain has **burned in**, or entered its stationary distribution. We discuss how to estimate when burnin has occured in Section 24.4.1. In the examples below, we just discard the initial 25% of the samples, for simplicity.

24.2.2 Example: Gibbs sampling for the Ising model

In Section 21.3.2, we applied mean field to an Ising model. Here we apply Gibbs sampling.

Gibbs sampling in pairwise MRF/CRF takes the form

$$p(x_t|\mathbf{x}_{-t}, \boldsymbol{\theta}) \propto \prod_{s \in \text{nbr}(t)} \psi_{st}(x_s, x_t) \tag{24.1}$$

In the case of an Ising model with edge potentials $\psi(x_s, x_t) = \exp(Jx_sx_t)$, where $x_t \in$

2011)

3. Josiah Willard Gibbs, 1839–1903, was an American physicist.

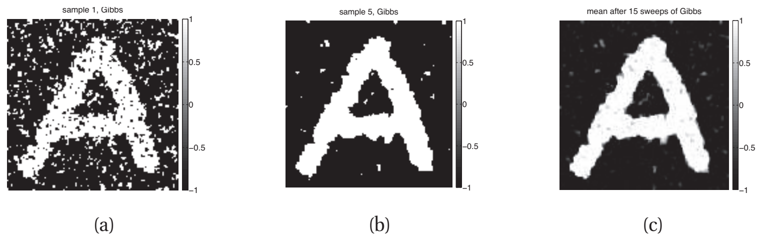


Figure 24.1 Example of image denoising. We use an Ising prior with $W_{ij} = J = 1$ and a Gaussian noise model with $\sigma = 2$. We use Gibbs sampling (Section 24.2) to perform approximate inference. (a) Sample from the posterior after one sweep over the image. (b) Sample after 5 sweeps. (c) Posterior mean, computed by averaging over 15 sweeps. Compare to Figure 21.3 which shows the results of using mean field inference. Figure generated by `isingImageDenoiseDemo`.

$\{-1, +1\}$, the full conditional becomes

$$p(x_t = +1|\mathbf{x}_{-t}, \boldsymbol{\theta}) = \frac{\prod_{s \in \text{nbr}(t)} \psi_{st}(x_t = +1, x_s)}{\prod_{s \in \text{nbr}(t)} \psi(s_t = +1, x_s) + \prod_{s \in \text{nbr}(t)} \psi(x_t = -1, x_s)} \tag{24.2}$$

$$= \frac{\exp[J \sum_{s \in \text{nbr}(t)} x_s]}{\exp[J \sum_{s \in \text{nbr}(t)} x_s] + \exp[-J \sum_{s \in \text{nbr}(t)} x_s]} \tag{24.3}$$

$$= \frac{\exp[J\eta_t]}{\exp[J\eta_t] + \exp[-J\eta_t]} = \text{sigm}(2J\eta_t) \tag{24.4}$$

where J is the coupling strength, $\eta_t \triangleq \sum_{s \in \text{nbr}(t)} x_t$ and $\text{sigm}(u) = 1/(1+e^{-u})$ is the sigmoid function. It is easy to see that $\eta_t = x_t(a_t - d_t)$, where a_t is the number of neighbors that agree with (have the same sign as) t, and d_t is the number of neighbors who disagree. If this number is equal, the "forces" on x_t cancel out, so the full conditional is uniform.

We can combine an Ising prior with a local evidence term ψ_t. For example, with a Gaussian observation model, we have $\psi_t(x_t) = \mathcal{N}(y_t|x_t, \sigma^2)$. The full conditional becomes

$$p(x_t = +1|\mathbf{x}_{-t}, \mathbf{y}, \boldsymbol{\theta}) = \frac{\exp[J\eta_t]\psi_t(+1)}{\exp[J\eta_t]\psi_t(+1) + \exp[-J\eta_t]\psi_t(-1)} \tag{24.5}$$

$$= \text{sigm}\left(2J\eta_t - \log \frac{\psi_t(+1)}{\psi_t(-1)}\right) \tag{24.6}$$

Now the probability of x_t 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 24.1 for an example of this algorithm applied to a simple image denoising problem. The results are similar to mean field (Figure 21.3) except that the final estimate (based on averaging the samples) is somewhat "blurrier", due to the fact that mean field tends to be over-confident.

24.2.3 Example: Gibbs sampling for inferring the parameters of a GMM

It is straightforward to derive a Gibbs sampling algorithm to "fit" a mixture model, especially if we use conjugate priors. We will focus on the case of mixture of Gaussians, although the results are easily extended to other kinds of mixture models. (The derivation, which follows from the results of Section 4.6, is much easier than the corresponding variational Bayes algorithm in Section 21.6.1.)

Suppose we use a semi-conjugate prior. Then the full joint distribution is given by

$$\begin{aligned} p(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\mathbf{z}|\boldsymbol{\pi}) p(\boldsymbol{\pi}) \prod_{k=1}^{K} p(\boldsymbol{\mu}_k) p(\boldsymbol{\Sigma}_k) && (24.7) \\ &= \left(\prod_{i=1}^{N} \prod_{k=1}^{K} (\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times && (24.8) \\ &\quad \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_0, \mathbf{V}_0) \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_0, \nu_0) && (24.9) \end{aligned}$$

We use the same prior for each mixture component. The full conditionals are as follows. For the discrete indicators, we have

$$p(z_i = k|\mathbf{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \propto \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (24.10)$$

For the mixing weights, we have (using results from Section 3.4)

$$p(\boldsymbol{\pi}|\mathbf{z}) = \mathrm{Dir}(\{\alpha_k + \sum_{i=1}^{N} \mathbb{I}(z_i = k)\}_{k=1}^{K}) \quad (24.11)$$

For the means, we have (using results from Section 4.6.1)

$$\begin{aligned} p(\boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k, \mathbf{z}, \mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_k, \mathbf{V}_k) && (24.12) \\ \mathbf{V}_k^{-1} &= \mathbf{V}_0^{-1} + N_k \boldsymbol{\Sigma}_k^{-1} && (24.13) \\ \mathbf{m}_k &= \mathbf{V}_k(\boldsymbol{\Sigma}_k^{-1} N_k \overline{\mathbf{x}}_k + \mathbf{V}_0^{-1} \mathbf{m}_0) && (24.14) \\ N_k &\triangleq \sum_{i=1}^{N} \mathbb{I}(z_i = k) && (24.15) \\ \overline{\mathbf{x}}_k &\triangleq \frac{\sum_{i=1}^{N} \mathbb{I}(z_i = k) \mathbf{x}_i}{N_k} && (24.16) \end{aligned}$$

For the covariances, we have (using results from Section 4.6.2)

$$\begin{aligned} p(\boldsymbol{\Sigma}_k|\boldsymbol{\mu}_k, \mathbf{z}, \mathbf{x}) &= \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_k, \nu_k) && (24.17) \\ \mathbf{S}_k &= \mathbf{S}_0 + \sum_{i=1}^{N} \mathbb{I}(z_i = k)(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T && (24.18) \\ \nu_k &= \nu_0 + N_k && (24.19) \end{aligned}$$

See `gaussMissingFitGibbs` for some Matlab code. (This code can also sample missing values for $\mathbf{x}$, if necessary.)

24.2.3.1 Label switching

Although it is simple to implement, Gibbs sampling for mixture models has a fundamental weakness. The problem is that the parameters of the model $\boldsymbol{\theta}$, and the indicator functions $\mathbf{z}$, are unidentifiable, since we can arbitrarily permute the hidden labels without affecting the likelihood (see Section 11.3.1). Consequently, we cannot just take a Monte Carlo average of the samples to compute posterior means, since what one sample considers the parameters for cluster 1 may be what another sample considers the parameters for cluster 2. Indeed, if we could average over all modes, we would find $\mathbb{E}[\boldsymbol{\mu}_k|\mathcal{D}]$ is the same for all k (assuming a symmetric prior). This is called the **label switching** problem.

This problem does not arise in EM or VBEM, which just "lock on" to a single mode. However, it arises in any method that visits multiple modes. In 1d problems, one can try to prevent this problem by introducing constraints on the parameters to ensure identifiability, e.g., $\mu_1 < \mu_2 < \mu_3$ (Richardson and Green 1997). However, this does not always work, since the likelihood might overwhelm the prior and cause label switching anyway. Furthermore, this technique does not scale to higher dimensions. Another approach is to post-process the samples by searching for a global label permutation to apply to each sample that minimizes some loss function (Stephens 2000); however, this can be slow.

Perhaps the best solution is simply to "not ask" questions that cannot be uniquely identified. For example, instead of asking for the probability that data point i belongs to cluster k, ask for the probability that data points i and j belong to the same cluster. The latter question is invariant to the labeling. Furthermore, it only refers to observable quantities (are i and j grouped together or not), rather than referring to unobservable quantities, such as latent clusters. This approach has the further advantage that it extends to infinite mixture models, discussed in Section 25.2, where K is unbounded; in such models, the notion of a hidden cluster is not well defined, but the notion of a partitioning of the data *is* well defined

24.2.4 Collapsed Gibbs sampling *

In some cases, we can analytically integrate out some of the unknown quantities, and just sample the rest. This is called a **collapsed Gibbs sampler**, and it tends to be much more efficient, since it is sampling in a lower dimensional space.

More precisely, suppose we sample $\mathbf{z}$ and integrate out $\boldsymbol{\theta}$. Thus the $\boldsymbol{\theta}$ parameters do not participate in the Markov chain; consequently we can draw conditionally independent samples $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathbf{z}^s, \mathcal{D})$, which will have much lower variance than samples drawn from the joint state space (Liu et al. 1994). This process is called **Rao-Blackwellisation**, named after the following theorem:

Theorem 24.2.1 (Rao-Blackwell)**.** *Let $\mathbf{z}$ and $\boldsymbol{\theta}$ be dependent random variables, and $f(\mathbf{z}, \boldsymbol{\theta})$ be some scalar function. Then*

$$\text{var}_{\mathbf{z},\boldsymbol{\theta}}[f(\mathbf{z}, \boldsymbol{\theta})] \geq \text{var}_{\mathbf{z}}[\mathbb{E}_{\boldsymbol{\theta}}[f(\mathbf{z}, \boldsymbol{\theta})|\mathbf{z}]] \tag{24.20}$$

This theorem guarantees that the variance of the estimate created by analytically integrating out $\boldsymbol{\theta}$ will always be lower (or rather, will never be higher) than the variance of a direct MC estimate. In collapsed Gibbs, we sample $\mathbf{z}$ with $\boldsymbol{\theta}$ integrated out; the above Rao-Blackwell theorem still applies in this case (Liu et al. 1994).

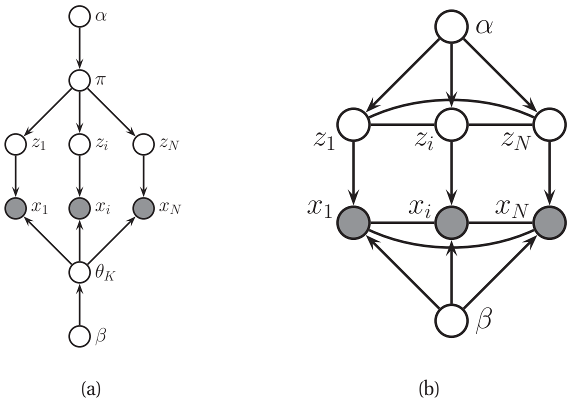


Figure 24.2 (a) A mixture model. (b) After integrating out the parameters.

We will encounter Rao-Blackwellisation again in Section 23.6. Although it can reduce statistical variance, it is only worth doing if the integrating out can be done quickly, otherwise we will not be able to produce as many samples per second as the naive method. We give an example of this below.

24.2.4.1 Example: collapsed Gibbs for fitting a GMM

Consider a GMM with a fully conjugate prior. In this case we can analytically integrate out the model parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\pi}$, and just sample the indicators $\mathbf{z}$. Once we integrate out $\boldsymbol{\pi}$, all the z_i nodes become inter-dependent. Similarly, once we integrate out $\boldsymbol{\theta}_k$, all the $\mathbf{x}_i$ nodes become inter-dependent, as shown in Figure 24.2(b). Nevertheless, we can easily compute the full conditionals as follows:

$$\begin{aligned}
p(z_i = k|\mathbf{z}_{-i}, \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha}, \cancel{\boldsymbol{\beta}})p(\mathbf{x}|z_i = k, \mathbf{z}_{-i}, \cancel{\boldsymbol{\alpha}}, \boldsymbol{\beta}) && (24.21)\\
&\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha})p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \\
&\quad p(\mathbf{x}_{-i}|\cancel{z_i = k}, \mathbf{z}_{-i}, \boldsymbol{\beta}) && (24.22)\\
&\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha})p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) && (24.23)
\end{aligned}$$

where $\boldsymbol{\beta} = (\mathbf{m}_0, \mathbf{V}_0, \mathbf{S}_0, \nu_0)$ are the hyper-parameters for the class-conditional densities. The first term can be obtained by integrating out $\boldsymbol{\pi}$. Suppose we use a symmetric prior of the form $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$, where $\alpha_k = \alpha/K$. From Equation 5.26 we have

$$p(z_1, \ldots, z_N|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)} \qquad (24.24)$$

Hence

$$p(z_i = k|\mathbf{z}_{-i}, \alpha) = \frac{p(\mathbf{z}_{1:N}|\alpha)}{p(\mathbf{z}_{-i}|\alpha)} = \frac{\frac{1}{\Gamma(N+\alpha)}}{\frac{1}{\Gamma(N+\alpha-1)}} \times \frac{\Gamma(N_k + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} \tag{24.25}$$

$$= \frac{\Gamma(N+\alpha-1)}{\Gamma(N+\alpha)} \frac{\Gamma(N_{k,-i} + 1 + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} = \frac{N_{k,-i} + \alpha/K}{N + \alpha - 1} \tag{24.26}$$

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 24.23, which is the posterior predictive distribution for $\mathbf{x}_i$ given all the other data and all the assignments, we use the fact that

$$p(\mathbf{x}_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k, \boldsymbol{\beta}) = p(\mathbf{x}_i|\mathcal{D}_{-i,k}) \tag{24.27}$$

where $\mathcal{D}_{-i,k} = \{\mathbf{x}_j : z_j = k, j \neq i\}$ is all the data assigned to cluster k except for $\mathbf{x}_i$. If we use a conjugate prior for $\boldsymbol{\theta}_k$, we can compute $p(\mathbf{x}_i|\mathcal{D}_{-i,k})$ 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 $\mathbf{x}_i$'s statistics from its current cluster (namely z_i), and then evaluate $\mathbf{x}_i$ under each cluster's posterior predictive. Once we have picked a new cluster, we add $\mathbf{x}_i$'s statistics to this new cluster.

Some pseudo-code for one step of the algorithm is shown in Algorithm 1, based on (Sudderth 2006, p94). (We update the nodes in random order to improve the mixing time, as suggested in (Roberts and Sahu 1997).) We can initialize the sample by sequentially sampling from $p(z_i|\mathbf{z}_{1:i-1}, \mathbf{x}_{1:i})$. (See `fmGibbs` for some Matlab code, by Yee-Whye Teh.) In the case of GMMs, both the naive sampler and collapsed sampler take $O(NKD)$ time per step.

Algorithm 24.1: Collapsed Gibbs sampler for a mixture model

```
1 for each $i = 1 : N$ in random order do
2     Remove $\mathbf{x}_i$'s sufficient statistics from old cluster $z_i$ ;
3     for each $k = 1 : K$ do
4         Compute $p_k(\mathbf{x}_i) \triangleq p(\mathbf{x}_i|\{\mathbf{x}_j : z_j = k, j \neq i\})$ ;
5     Compute $p(z_i = k|\mathbf{z}_{-i}, \mathcal{D}) \propto (N_{k,-i} + \alpha/K)p_k(\mathbf{x}_i)$;
6     Sample $z_i \sim p(z_i|\cdot)$ ;
7     Add $\mathbf{x}_i$'s sufficient statistics to new cluster $z_i$
```

A comparison of this method with the standard Gibbs sampler is shown in Figure 24.3. The vertical axis is the data log probability at each iteration, computed using

$$\log p(\mathcal{D}|\mathbf{z}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left[\pi_{z_i} p(\mathbf{x}_i|\boldsymbol{\theta}_{z_i})\right] \tag{24.28}$$

To compute this quantity using the collapsed sampler, we have to sample $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:K})$ given the data and the current assignment $\mathbf{z}$.

In Figure 24.3 we see that the collapsed sampler does indeed generally work better than the vanilla sampler. Occasionally, however, both methods can get stuck in poor local modes. (Note

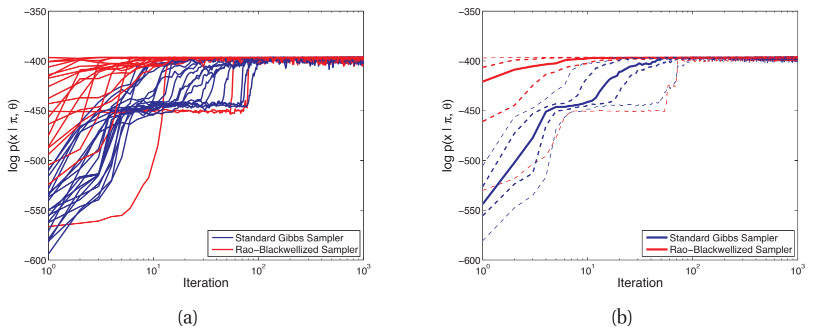


Figure 24.3 Comparison of collapsed (red) and vanilla (blue) Gibbs sampling for a mixture of $K = 4$ two-dimensional Gaussians applied to $N = 300$ data points (shown in Figure 25.7). We plot log probability of the data vs iteration. (a) 20 different random initializations. (b) logprob averaged over 100 different random initializations. Solid line is the median, thick dashed in the 0.25 and 0.75 quantiles, and thin dashed are the 0.05 and 0.95 quintiles. Source: Figure 2.20 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

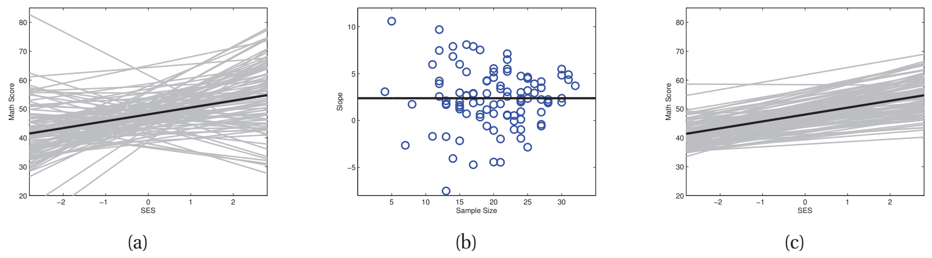


Figure 24.4 (a) Least squares regression lines for math scores vs socio-economic status for 100 schools. Population mean (pooled estimate) is in bold. (b) Plot of $\hat{w}_{2j}$ (the slope) vs N_j (sample size) for the 100 schools. The extreme slopes tend to correspond to schools with smaller sample sizes. (c) Predictions from the hierarchical model. Population mean is in bold. Based on Figure 11.1 of (Hoff 2009). Figure generated by `multilevelLinregDemo`, written by Emtiyaz Khan.

that the error bars in Figure 24.3(b) are averaged over starting values, whereas the theorem refers to MC samples in a single run.)

24.2.5 Gibbs sampling for hierarchical GLMs

Often we have data from multiple related sources. If some sources are more reliable and/or data-rich than others, it makes sense to model all the data simultaneously, so as to enable the borrowing of statistical strength. One of the most natural way to solve such problems is to use hierarchical Bayesian modeling, also called multi-level modeling. In Section 9.6, we discussed a way to perform approximate inference in such models using variational methods. Here we discuss how to use Gibbs sampling.

To explain the method, consider the following example. Suppose we have data on students

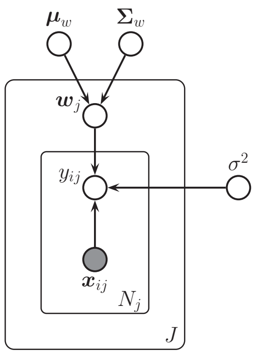


Figure 24.5 Multi-level model for linear regression.

in different schools. Such data is naturally modeled in a two-level hierarchy: we let y_{ij} be the response variable we want to predict for student i in school j. This prediction can be based on school and student specific covariates, $\mathbf{x}_{ij}$. Since the quality of schools varies, we want to use a separate parameter for each school. So our model becomes

$$y_{ij} = \mathbf{x}_{ij}^T \mathbf{w}_j + \epsilon_{ij} \tag{24.29}$$

We will illustrate this model below, using a dataset from (Hoff 2009, p197), where x_{ij} is the socio-economic status (SES) of student i in school y, and y_{ij} is their math score.

We could fit each $\mathbf{w}_j$ separately, but this can give poor results if the sample size of a given school is small. This is illustrated in Figure 24.4(a), which plots the least squares regression line estimated separately for each of the $J = 100$ schools. We see that most of the slopes are positive, but there are a few "errant" cases where the slope is negative. It turns out that the lines with extreme slopes tend to be in schools with small sample size, as shown in Figure 24.4(b). Thus we may not necessarily trust these fits.

We can get better results if we construct a hierarchical Bayesian model, in which the $\mathbf{w}_j$ are assumed to come from a common prior: $\mathbf{w}_j \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$. This is illustrated in Figure 24.5. In this model, the schools with small sample size borrow statistical strength from the schools with larger sample size, because the $\mathbf{w}_j$'s are correlated via the latent common parents $(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$. (It is crucial that these hyper-parameters be inferrred from data; if they were fixed constants, the $\mathbf{w}_j$ would be conditionally independent, and there would be no information sharing between them.)

To complete the model specification, we must specify priors for the shared parameters. Following (Hoff 2009, p198), we will use the following semi-conjugate forms, for convenience:

$$\boldsymbol{\mu}_w \sim \mathcal{N}(\boldsymbol{\mu}_0, \mathbf{V}_0) \tag{24.30}$$

$$\boldsymbol{\Sigma}_w \sim \text{IW}(\eta_0, \mathbf{S}_0^{-1}) \tag{24.31}$$

$$\sigma^2 \sim \text{IG}(\nu_0/2, \nu_0\sigma_0^2/2) \tag{24.32}$$

Given this, it is simple to show that the full conditionals needed for Gibbs sampling have the following forms. For the group-specific weights:

$$p(\mathbf{w}_j|\mathcal{D}_j, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{w}_j|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \tag{24.33}$$

$$\boldsymbol{\Sigma}_j^{-1} = \boldsymbol{\Sigma}^{-1} + \mathbf{X}_j^T\mathbf{X}_j/\sigma^2 \tag{24.34}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\Sigma}_j(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \mathbf{X}_j^T\mathbf{y}_j/\sigma^2) \tag{24.35}$$

For the overall mean:

$$p(\boldsymbol{\mu}_w|\mathbf{w}_{1:J}, \boldsymbol{\Sigma}_w) = \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N) \tag{24.36}$$

$$\boldsymbol{\Sigma}_N^{-1} = \mathbf{V}_0^{-1} + J\boldsymbol{\Sigma}^{-1} \tag{24.37}$$

$$\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N(\mathbf{V}_0^{-1}\boldsymbol{\mu}_0 + J\boldsymbol{\Sigma}^{-1}\overline{\mathbf{w}}) \tag{24.38}$$

where $\overline{\mathbf{w}} = \frac{1}{J}\sum_j \mathbf{w}_j$. For the overall covariance:

$$p(\boldsymbol{\Sigma}_w|\boldsymbol{\mu}_w, \mathbf{w}_{1:J}) = \text{IW}((\mathbf{S}_0 + \mathbf{S}_\mu)^{-1}, \eta_0 + J) \tag{24.39}$$

$$\mathbf{S}_\mu = \sum_j (\mathbf{w}_j - \boldsymbol{\mu}_w)(\mathbf{w}_j - \boldsymbol{\mu}_w)^T \tag{24.40}$$

For the noise variance:

$$p(\sigma^2|\mathcal{D}, \mathbf{w}_{1:J}) = \text{IG}([\nu_0 + N]/2, [\nu_0\sigma_0^2 + \text{SSR}(\mathbf{w}_{1:J})]/2) \tag{24.41}$$

$$\text{SSR}(\mathbf{w}_{1:J}) = \sum_{j=1}^{J}\sum_{i=1}^{N_j}(y_{ij} - \mathbf{w}_j^T\mathbf{x}_{ij})^2 \tag{24.42}$$

Applying Gibbs sampling to our hierarchical model, we get the results shown in Figure 24.4(c). The light gray lines plot the mean of the posterior predictive distribution for each school:

$$\mathbb{E}[y_j|\mathbf{x}_{ij}] = \mathbf{x}_{ij}^T\hat{\mathbf{w}}_j \tag{24.43}$$

where

$$\hat{\mathbf{w}}_j = \mathbb{E}[\mathbf{w}_j|\mathcal{D}] \approx \frac{1}{S}\sum_{s=1}^{S}\mathbf{w}_j^{(s)} \tag{24.44}$$

The dark gray line in the middle plots the prediction using the overall mean parameters, $\mathbf{x}_{ij}^T\hat{\boldsymbol{\mu}}_w$. We see that the method has regularized the fits quite nicely, without enforcing too much uniformity. (The amount of shrinkage is controlled by $\boldsymbol{\Sigma}_w$, which in turns depends on the hyper-parameters; in this example, we used vague values.)

24.2.6 BUGS and JAGS

One reason Gibbs sampling is so popular is that it is possible to design general purpose software that will work for almost any model. This software just needs a model specification, usually in the form a directed graphical model (specified in a file, or created with a graphical user interface), and a library of methods for sampling from different kinds of full conditionals. (This can often be done using adaptive rejection sampling, described in Section 23.3.4.) An example of such a package is **BUGS** (Lunn et al. 2000), which stands for "Bayesian updating using Gibbs Sampling". BUGS is very widely used in biostatistics and social science. Another more recent, but very similar, package is **JAGS** (Plummer 2003), which stands for "Just Another Gibbs Sampler". This uses a similar model specification language to BUGS.

For example, we can describe the model in Figure 24.5 as follows:

```
model {
 for (i in 1:N) {
    for (j in 1:J) {
       y[i,j] ~ dnorm(y.hat[i,j], tau.y)
       y.hat[i,j] <- inprod(W[j, ], X[i, j, ])
    }
}
tau.y <- pow(sigma.y, -2)
sigma.y ~ dunif(0,100)

for (j in 1:J) {
  W[j,] ~ dmnorm(mu, SigmaInv)
}
SigmaInv  ~ dwish(S0[,], eta0)
mu ~ dmnorm(mu0, V0inv)
}
```

We can then just pass this model to BUGS or JAGS, which will generate samples for us. See the webpages for details.

Although this approach is appealing, unfortunately it can be much slower than using hand-written code, especially for complex models. There has been some work on automatically deriving model-specific optimized inference code (Fischer and Schumann 2003), but fast code still typically requires human expertise.

24.2.7 The Imputation Posterior (IP) algorithm

The **Imputation Posterior** or IP algorithm (Tanner and Wong 1987) is a special case of Gibbs sampling in which we group the variables into two classes: hidden variables $\mathbf{z}$ and parameters $\boldsymbol{\theta}$. This should sound familiar: it is basically an MCMC version of EM, where the E step gets replaced by the I step, and the M step gets replaced the P step. This is an example of a more general strategy called **data augmentation**, whereby we introduce auxiliary variables in order to simplify the posterior computations (here the computation of $p(\boldsymbol{\theta}|\mathcal{D})$). See (Tanner 1996; van Dyk and Meng 2001) for more information.

24.2.8 Blocking 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, it will take a long time to move away from the current state. This is illustrated in Figure 24.6, where we illustrate sampling from a 2d Gaussian (see Exercise 24.1 for the details). If the variables are highly correlated, the algorithm

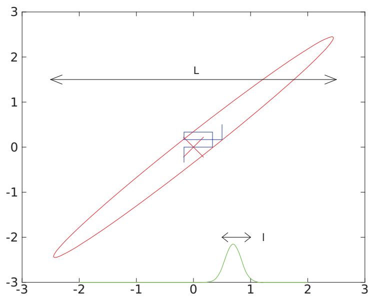


Figure 24.6 Illustration of potentially slow sampling when using Gibbs sampling for a skewed 2D Gaussian. Based on Figure 11.11 of (Bishop 2006b). Figure generated by `gibbsGaussDemo`.

will move very slowly through the state space. In particular, the size of the moves is controlled by the variance of the conditional distributions. If this is ℓ in the x_1 direction, and the support of the distribution is L along this dimension, then we need $O((L/\ell)^2)$ steps to obtain an independent sample.

In some cases we can efficiently sample groups of variables at a time. This is called **blocking Gibbs sampling** or **blocked Gibbs sampling** (Jensen et al. 1995; Wilkinson and Yeung 2002), and can make much bigger moves through the state space.

24.3 Metropolis Hastings algorithm

Although Gibbs sampling is simple, it is somewhat restricted in the set of models to which it can be applied. For example, it is not much help in computing $p(\mathbf{w}|\mathcal{D})$ for a logistic regression model, since the corresponding graphical model has no useful Markov structure. In addition, Gibbs sampling can be quite slow, as we mentioned above.

Fortunately, there is a more general algorithm that can be used, known as the **Metropolis Hastings** or **MH** algorithm, which we describe below.

24.3.1 Basic idea

The basic idea in MH is that at each step, we propose to move from the current state $\mathbf{x}$ to a new state $\mathbf{x}'$ with probability $q(\mathbf{x}'|\mathbf{x})$, 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. A commonly used proposal is a symmetric Gaussian distribution centered on the current state, $q(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x}'|\mathbf{x}, \mathbf{\Sigma})$; this is called a **random walk Metropolis algorithm**. We discuss how to choose $\mathbf{\Sigma}$ in Section 24.3.3. If we use a proposal of the form $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}')$, 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 23.4).

Having proposed a move to $\mathbf{x}'$, we then decide whether to **accept** this proposal or not according to some formula, which ensures that the fraction of time spent in each state is proportional to $p^*(\mathbf{x})$. If the proposal is accepted, the new state is $\mathbf{x}'$, otherwise the new state is the same as the current state, $\mathbf{x}$ (i.e., we repeat the sample).

If the proposal is symmetric, so $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}|\mathbf{x}')$, the acceptance probability is given by the following formula:

$$r = \min(1, \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}) \tag{24.45}$$

We see that if $\mathbf{x}'$ is more probable than $\mathbf{x}$, we definitely move there (since $\frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})} > 1$), but if $\mathbf{x}'$ 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 24.3.6, we prove that this procedure ensures that the fraction of time we spend in each state $\mathbf{x}$ is proportional to $p^*(\mathbf{x})$.

If the proposal is asymmetric, so $q(\mathbf{x}'|\mathbf{x}) \neq q(\mathbf{x}|\mathbf{x}')$, we need the **Hastings correction**, given by the following:

$$r = \min(1, \alpha) \tag{24.46}$$

$$\alpha = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})}{p^*(\mathbf{x})/q(\mathbf{x}|\mathbf{x}')} \tag{24.47}$$

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 $p^*(\mathbf{x}) = \frac{1}{Z}\tilde{p}(\mathbf{x})$, where $\tilde{p}(\mathbf{x})$ is an unnormalized distribution and Z is the normalization constant. Then

$$\alpha = \frac{(\tilde{p}(\mathbf{x}')/Z)\; q(\mathbf{x}|\mathbf{x}')}{(\tilde{p}(\mathbf{x})/Z)\; q(\mathbf{x}'|\mathbf{x})} \tag{24.48}$$

so the Z's cancel. Hence we can sample from p^* even if Z is unknown. In particular, all we have to do is evaluate $\tilde{p}$ pointwise, where $\tilde{p}(\mathbf{x}) = p^*(\mathbf{x})Z$.

The overall algorithm is summarized in Algorithm 2.

24.3.2 Gibbs sampling is a special case of MH

It turns out that Gibbs sampling, which we discussed in Section 24.2, is a special case of MH. In particular, it is equivalent to using MH with a sequence of proposals of the form

$$q(\mathbf{x}'|\mathbf{x}) = p(x_i'|\mathbf{x}_{-i})\mathbb{I}(\mathbf{x}_{-i}' = \mathbf{x}_{-i}) \tag{24.49}$$

That is, we move to a new state where x_i is sampled from its full conditional, but $\mathbf{x}_{-i}$ is left unchanged.

We now prove that the acceptance rate of each such proposal is 1, so the overall algorithm also has an acceptance rate of 100%. We have

$$\alpha = \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p(x_i'|\mathbf{x}_{-i}')p(\mathbf{x}_{-i}')p(x_i|\mathbf{x}_{-i}')}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})} \tag{24.50}$$

$$= \frac{p(x_i'|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i|\mathbf{x}_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})} = 1 \tag{24.51}$$

Algorithm 24.2: Metropolis Hastings algorithm

```
1 Initialize $x^0$ ;
2 for $s = 0, 1, 2, \ldots$ do
3     Define $x = x^s$;
4     Sample $x' \sim q(x'|x)$;
5     Compute acceptance probability
          $\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$
      Compute $r = \min(1, \alpha)$;
6     Sample $u \sim U(0, 1)$ ;
7     Set new sample to
          $x^{s+1} = \begin{cases} x' & \text{if } u < r \\ x^s & \text{if } u \geq r \end{cases}$
```

where we exploited the fact that $\mathbf{x}'_{-i} = \mathbf{x}_{-i}$, and that $q(\mathbf{x}'|\mathbf{x}) = p(x'_i|\mathbf{x}_{-i})$.

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 24.2.8). Fortunately, there are many other kinds of proposals we can use, as we discuss below.

24.3.3 Proposal distributions

For a given target distribution p^*, a proposal distribution q is valid or admissible if it gives a non-zero probability of moving to the states that have non-zero probability in the target. Formally, we can write this as

$$\text{supp}(p^*) \subseteq \cup_x \text{supp}(q(\cdot|x)) \tag{24.52}$$

For example, a Gaussian random walk proposal has non-zero probability density on the entire state space, and hence is a valid proposal for any continuous state space.

Of course, in practice, it is important that the proposal spread its probability mass in just the right way. Figure 24.7 shows an example where we use MH to sample from a mixture of two 1D Gaussians using a random walk proposal, $q(x'|x) = \mathcal{N}(x'|x, v)$. This is a somewhat tricky target distribution, since it consists of two well separated modes. It is very important to set the variance of the proposal v correctly: If the variance is too low, the chain will only explore one of the modes, as shown in Figure 24.7(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 24.7(b). If we set the proposal's variance just right, we get the trace in Figure 24.7(c), where the samples clearly explore the support of the target distribution. We discuss how to tune the proposal below.

One big advantage of Gibbs sampling is that one does not need to choose the proposal

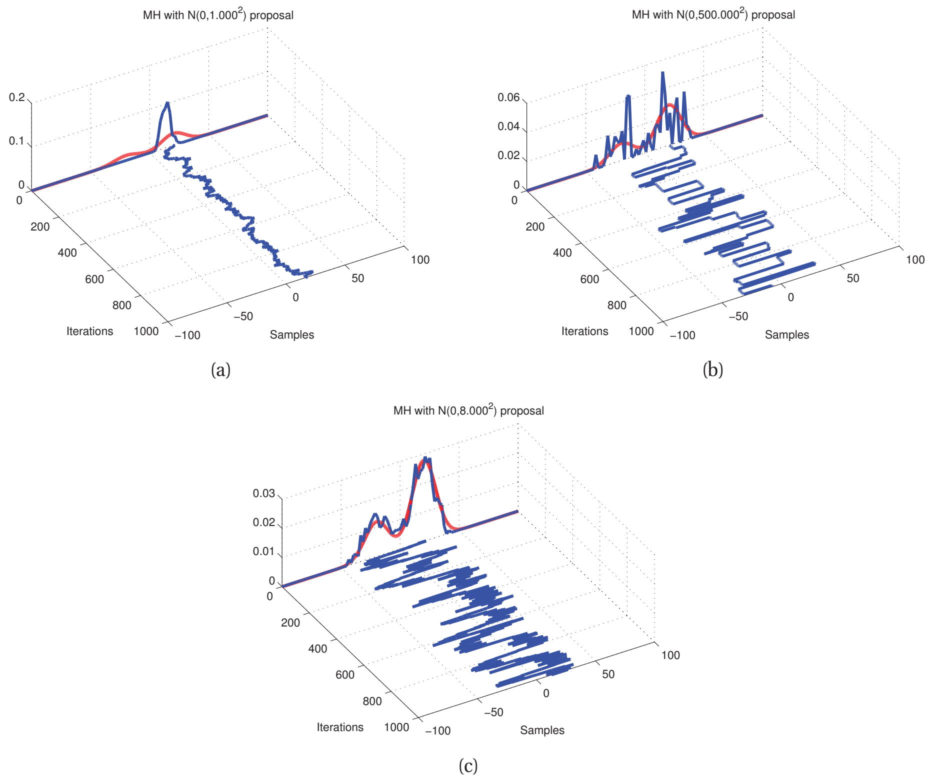


Figure 24.7 An example of the Metropolis Hastings algorithm for sampling from a mixture of two 1D Gaussians ($\boldsymbol{\mu} = (-20, 20)$, $\boldsymbol{\pi} = (0.3, 0.7)$, $\boldsymbol{\sigma} = (100, 100)$), using a Gaussian proposal with variances of $v \in \{1, 500, 8\}$. (a) When $v = 1$, the chain gets trapped near the starting state and fails to sample from the mode at $\mu = -20$. (b) When $v = 500$, 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 $v = 8$ is just right and leads to a good approximation of the true distribution (shown in red). Figure generated by `mcmcGmmDemo`. Based on code by Christophe Andrieu and Nando de Freitas.

distribution, and furthermore, the acceptance rate is 100%. Of course, a 100% acceptance can trivially be achieved by using a proposal with variance 0 (assuming we start at a mode), but this is obviously not exploring the posterior. So having a high acceptance is not the ultimate goal. We can increase the amount of exploration by increasing the variance of the Gaussian kernel. Often one experiments with different parameters until the acceptance rate is between 25% and 40%, which theory suggests is optimal, at least for Gaussian target distributions. These short initial runs, used to tune the proposal, are called **pilot runs**.

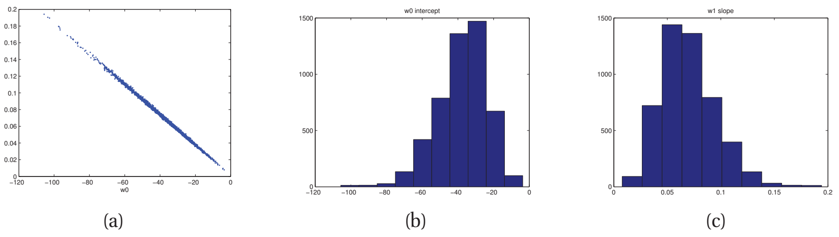


Figure 24.8 (a) Joint posterior of the parameters for 1d logistic regression when applied to some SAT data. (b) Marginal for the offset w_0. (c) Marginal for the slope w_1. We see that the marginals do not capture the fact that the parameters are highly correlated. Figure generated by `logregSatMhDemo`.

24.3.3.1 Gaussian proposals

If we have a continuous state space, the Hessian $\mathbf{H}$ at a local mode $\hat{\mathbf{w}}$ can be used to define the covariance of a Gaussian proposal distribution. This approach has the advantage that the Hessian models the local curvature and length scales of each dimension; this approach therefore avoids some of the slow mixing behavior of Gibbs sampling shown in Figure 24.6.

There are two obvious approaches: (1) an independence proposal, $q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}(\mathbf{w}'|\hat{\mathbf{w}}, \mathbf{H}^{-1})$ or (2), a random walk proposal, $q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}(\mathbf{w}'|\mathbf{w}, s^2\mathbf{H}^{-1})$, where s^2 is a scale factor chosen to facilitate rapid mixing. (Roberts and Rosenthal 2001) prove that, if the posterior is Gaussian, the asymptotically optimal value is to use $s^2 = 2.38^2/D$, where D is the dimensionality of $\mathbf{w}$; this results in an acceptance rate of 0.234.

For example, consider MH for binary logistic regression. From Equation 8.7, we have that the Hessian of the log-likelihood is $\mathbf{H}_l = \mathbf{X}^T\mathbf{D}\mathbf{X}$, where $\mathbf{D} = \text{diag}(\mu_i(1-\mu_i))$ and $\mu_i = \text{sigm}(\hat{\mathbf{w}}^T\mathbf{x}_i)$. If we assume a Gaussian prior, $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{V}_0)$, we have $\mathbf{H} = \mathbf{V}_0^{-1} + \mathbf{H}_l$, so the asymptotically optimal Gaussian proposal has the form

$$q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}\left(\mathbf{w}, \frac{2.38^2}{D}\left(\mathbf{V}_0^{-1} + \mathbf{X}^T\mathbf{D}\mathbf{X}\right)^{-1}\right) \tag{24.53}$$

See (Gamerman 1997; Rossi et al. 2006; Fruhwirth-Schnatter and Fruhwirth 2010) for further details. The approach is illustrated in Figure 24.8, where we sample parameters from a 1d logistic regression model fit to some SAT data. We initialize the chain at the mode, computed using IRLS, and then use the above random walk Metropolis sampler.

If you cannot afford to compute the mode or its Hessian $\mathbf{XDX}$, an alternative approach, suggested in (Scott 2009), is to approximate the above proposal as follows:

$$q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}\left(\mathbf{w}, \left(\mathbf{V}_0^{-1} + \frac{6}{\pi^2}\mathbf{X}^T\mathbf{X}\right)^{-1}\right) \tag{24.54}$$

24.3.3.2 Mixture proposals

If one doesn't know what kind of proposal to use, one can try a **mixture proposal**, which is a convex combination of base proposals:

$$q(\mathbf{x}'|\mathbf{x}) = \sum_{k=1}^{K} w_k q_k(\mathbf{x}'|\mathbf{x}) \tag{24.55}$$

where w_k are the mixing weights. As long as each q_k is individually valid, the overall proposal will also be valid.

24.3.3.3 Data-driven MCMC

The most efficient proposals depend not just on the previous hidden state, but also the visible data, i.e., they have the form $q(\mathbf{x}'|\mathbf{x}, \mathcal{D})$. This is called **data-driven MCMC** (see e.g., (Tu and Zhu 2002)). To create such proposals, one can sample $(\mathbf{x}, \mathcal{D})$ pairs from the forwards model and then train a discriminative classifier to predict $p(\mathbf{x}|f(\mathcal{D}))$, where $f(\mathcal{D})$ are some features extracted from the visible data.

Typically $\mathbf{x}$ is a high-dimensional vector (e.g., position and orientation of all the limbs of a person in a visual object detector), so it is hard to predict the entire state vector, $p(\mathbf{x}|f(\mathcal{D}))$. Instead we might train a discriminative detector to predict parts of the state-space, $p(x_k|f_k(\mathcal{D}))$, such as the location of just the face of a person. We can then use a proposal of the form

$$q(\mathbf{x}'|\mathbf{x}, \mathcal{D}) = \pi_0 q_0(\mathbf{x}'|\mathbf{x}) + \sum_k \pi_k q_k(x'_k|f_k(\mathcal{D})) \tag{24.56}$$

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. For added efficiency, the discriminative proposals should suggest joint changes to multiple variables, but this is often hard to do.

The overall procedure is a form of **generate and test**: the discriminative proposals $q(\mathbf{x}'|\mathbf{x})$ generate new hypotheses, which are then "tested" by computing the posterior ratio $\frac{p(\mathbf{x}'|\mathcal{D})}{p(\mathbf{x}|\mathcal{D})}$, to see if the new hypothesis is better or worse. By adding an annealing step, one can modify the algorithm to find posterior modes; this is called **simulated annealing**, and is described in Section 24.6.1. One advantage of using the mode-seeking version of the algorithm is that we do not need to ensure the proposal distribution is reversible.

24.3.4 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., (Andrieu and Thoms 2008) for details.

24.3.5 Initialization and mode hopping

It is necessary to start MCMC in an initial state that has non-zero probability. If the model has deterministic constraints, finding such a legal configuration may be a hard problem in itself. It is therefore common to initialize MCMC methods at a local mode, found using an optimizer.

In some domains (especially with discrete state spaces), it is a more effective use of computation time to perform multiple restarts of an optimizer, and to average over these modes, rather than exploring similar points around a local mode. However, in continuous state spaces, the mode contains negligible volume (Section 5.2.1.3), so it is necessary to locally explore around each mode, in order to visit enough posterior probability mass.

24.3.6 Why MH works *

To prove that the MH procedure generates samples from p^*, we have to use a bit of Markov chain theory, so be sure to read Section 17.2.3 first.

The MH algorithm defines a Markov chain with the following transition matrix:

$$p(\mathbf{x}'|\mathbf{x}) = \begin{cases} q(\mathbf{x}'|\mathbf{x})r(\mathbf{x}'|\mathbf{x}) & \text{if } \mathbf{x}' \neq \mathbf{x} \\ q(\mathbf{x}|\mathbf{x}) + \sum_{\mathbf{x}'\neq\mathbf{x}} q(\mathbf{x}'|\mathbf{x})(1 - r(\mathbf{x}'|\mathbf{x})) & \text{otherwise} \end{cases} \tag{24.57}$$

This follows from a case analysis: if you move to $\mathbf{x}'$ from $\mathbf{x}$, you must have proposed it (with probability $q(\mathbf{x}'|\mathbf{x})$) and it must have been accepted (with probability $r(\mathbf{x}'|\mathbf{x})$); otherwise you stay in state $\mathbf{x}$, either because that is what you proposed (with probability $q(\mathbf{x}|\mathbf{x})$), or because you proposed something else (with probability $q(\mathbf{x}'|\mathbf{x})$) but it was rejected (with probability $1 - r(\mathbf{x}'|\mathbf{x})$).

Let us analyse this Markov chain. Recall from Section 17.2.3.4 that a chain satisfies **detailed balance** if

$$p(\mathbf{x}'|\mathbf{x})p^*(\mathbf{x}) = p(\mathbf{x}|\mathbf{x}')p^*(\mathbf{x}') \tag{24.58}$$

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 24.58 holds, we say that p^* is an **invariant** distribution wrt the Markov transition kernel q.)

Theorem 24.3.1. *If the transition matrix defined by the MH algorithm (given by Equation 24.57) is ergodic and irreducible, then p^* is its unique limiting distribution.*

Proof. Consider two states $\mathbf{x}$ and $\mathbf{x}'$. Either

$$p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) < p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.59}$$

or

$$p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.60}$$

We will ignore ties (which occur with probability zero for continuous distributions). Without loss of generality, assume that $p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')$. Hence

$$\alpha(\mathbf{x}'|\mathbf{x}) = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} < 1 \tag{24.61}$$

Hence we have $r(\mathbf{x}'|\mathbf{x}) = \alpha(\mathbf{x}'|\mathbf{x})$ and $r(\mathbf{x}|\mathbf{x}') = 1$.

Now to move from $\mathbf{x}$ to $\mathbf{x}'$ we must first propose $\mathbf{x}'$ and then accept it. Hence

$$p(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})r(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})\frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}q(\mathbf{x}|\mathbf{x}') \tag{24.62}$$

Hence

$$p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.63}$$

The backwards probability is

$$p(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}')r(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}') \tag{24.64}$$

since $r(\mathbf{x}|\mathbf{x}') = 1$. Inserting this into Equation 24.63 we get

$$p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')p(\mathbf{x}|\mathbf{x}') \tag{24.65}$$

so detailed balance holds wrt p^*. Hence, from Theorem 17.2.3, p^* is a stationary distribution. Furthermore, from Theorem 17.2.2, this distribution is unique, since the chain is ergodic and irreducible. □

24.3.7 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 $\mathbf{x}_m \in \mathcal{X}_m$ (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** (Green 2003). 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. 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 in different dimensionality spaces, which is meaningless. It is like trying to compare a sphere with a circle. The solution, proposed by (Green 1998) 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.

Unfortunately, we do not have space to go into details here. Suffice it to say that the method can be made to work in theory, although it is a bit tricky in practice. 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, (Denison et al. 2002) includes many examples of applications of collapsed RJMCMC applied to Bayesian inference fro adaptive basis-function models. They sample basis functions from a fixed set of candidates (e.g., centered on the data points), and integrate out the other parameters analytically. This provides a Bayesian alternative to using RVMs or SVMs.

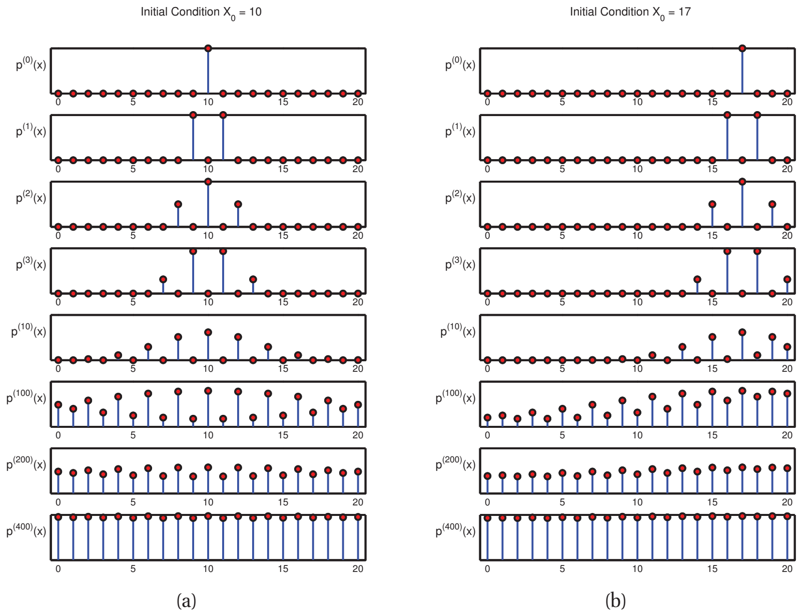


Figure 24.9 Illustration of convergence to the uniform distribution over $\{0, 1, \ldots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Based on Figures 29.14 and 29.15 of (MacKay 2003). Figure generated by `randomWalk0to20Demo`.

24.4 Speed and accuracy of MCMC

In this section, we discuss a number of important theoretical and practical issues to do with MCMC.

24.4.1 The burn-in phase

We start MCMC from an arbitrary initial state. As we explained in Section 17.2.3, only when the chain has "forgotten" where it started from will the samples be coming from the chain's stationary distribution. 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 24.9, 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 100 steps until the chain has "forgotten" where it started from.

It is difficult to diagnose when the chain has burned in, an issue we discuss in more detail below. (This is one of the fundamental weaknesses of MCMC.) As an interesting example of what can happen if you start collecting samples too early, consider the Potts model. Figure 24.10(a), shows a sample after 500 iterations of Gibbs sampling. This suggests that the model likes

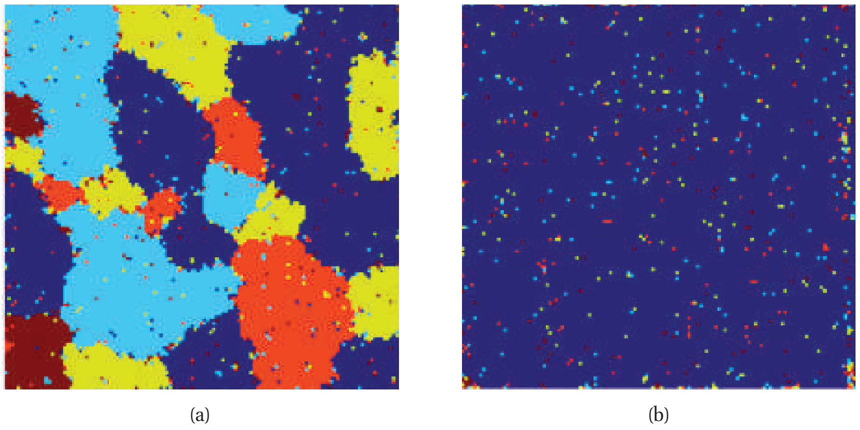


Figure 24.10 Illustration of problems caused by poor mixing. (a) One sample from a 5-state Potts model on a 128×128 grid with 8 nearest neighbor connectivity and $J = 2/3$ (as in (Geman and Geman 1984)), after 200 iterations. (b) One sample from the same model after 10,000 iterations. Used with kind permission of Erik Sudderth.

medium-sized regions where the label is the same, implying the model would make a good prior for image segmentation. Indeed, this was suggested in the original Gibbs sampling paper (Geman and Geman 1984).

However, it turns out that if you run the chain long enough, you get isolated speckles, as in Figure 24.10(b). The results depend on the coupling strength, but in general, it is very hard to find a setting which produces nice medium-sized blobs: most parameters result in a few super-clusters, or lots of small fragments. In fact, there is a rapid phase transition between these two regimes. This led to a paper called "The Ising/Potts model is not well suited to segmentation tasks" (Morris et al. 1996). It is possible to create priors more suited to image segmentation (e.g., (Sudderth and Jordan 2008)), but the main point here is that sampling before reaching convergence can lead to erroneous conclusions.

24.4.2 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

$$\tau_\epsilon(x_0) \triangleq \min\{t : ||\delta_{x_0}(x)T^t - p^*||_1 \leq \epsilon\} \tag{24.66}$$

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

$$\tau_\epsilon \triangleq \max_{x_0} \tau_\epsilon(x_0) \tag{24.67}$$

The mixing time is determined by the eigengap $\gamma = \lambda_1 - \lambda_2$, which is the difference of the

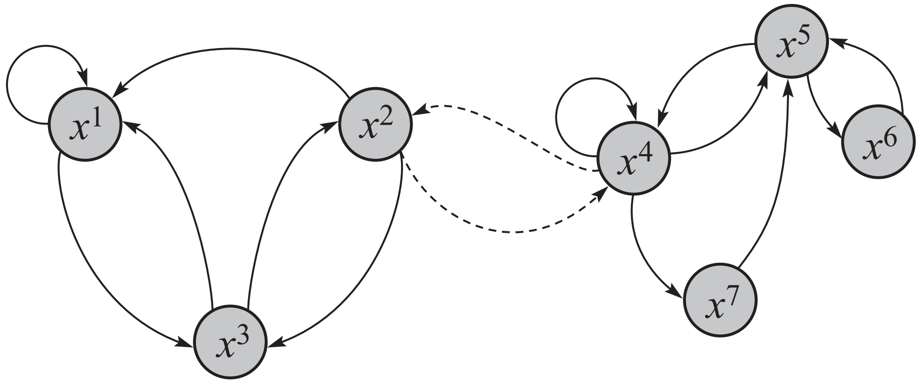


Figure 24.11 A Markov chain with low conductance. The dotted arcs represent transitions with very low probability. Source: Figure 12.6 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

first and second eigenvalues of the transition matrix. In particular, one can show that

$$\tau_\epsilon \leq O(\frac{1}{\gamma}\log\frac{n}{\epsilon}) \tag{24.68}$$

where n is the number of states. Since computing the transition matrix can be hard to do, especially for high dimensional and/or continuous state spaces, it is useful to find other ways to estimate the mixing time.

An alternative approach is to examine the geometry of the state space. For example, consider the chain in Figure 24.11. 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.) We define the **conductance** ϕ of a chain as the minimum probability, over all subsets of states, of transitioning from that set to its complement:

$$\phi \triangleq \min_{S:0\leq p^*(S)\leq 0.5} \frac{\sum_{x\in S, x'\in S^c} T(x\to x')}{p^*(S)}, \tag{24.69}$$

One can show that

$$\tau_\epsilon \leq O(\frac{1}{\phi^2}\log\frac{n}{\epsilon}) \tag{24.70}$$

Hence chains with low conductance have high mixing time. For example, distributions with well-separated modes usually have high mixing time. Simple MCMC methods often do not work well in such cases, and more advanced algorithms, such as parallel tempering, are necessary (see e.g., (Liu 2001)).

24.4.3 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. In practice various heuristics have been proposed to diagnose convergence — see (Geyer 1992; Cowles and Carlin 1996; Brooks and Roberts 1998) for a review. Strictly speaking, these methods do not diagnose convergence, but rather non-convergence. That is, the method may claim the chain has converged when in fact it has not. This is a flaw common to all convergence diagnostics, since diagnosing convergence is computationally intractable in general (Bhatnagar et al. 2010).

One of the simplest approaches to assessing when the method has converged is to run multiple chains from very different **overdispersed** starting points, and to plot the samples of some variables of interest. 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.

Figure 24.12 gives an example. We show the traceplot for x which was sampled from a mixture of two 1D Gaussians using four different methods: MH with a symmetric Gaussian proposal of variance $\sigma^2 \in \{1, 8, 500\}$, and Gibbs sampling. We see that $\sigma^2 = 1$ has not mixed, which is also evident from Figure 24.7(a), which shows that a single chain never leaves the area where it started. The results for the other methods indicate that the chains rapidly converge to the stationary distribution, no matter where they started. (The sticky nature of the $\sigma^2 = 500$ proposal is very evident. This reduces the computational efficiency, as we discuss below, but not the statistical validity.)

24.4.3.1 Estimated potential scale reduction (EPSR)

We can assess convergence more quantitatively as follows. The basic idea is to compare the variance of a quantity within each chain to its variance across chains. More precisely, suppose we collect S samples (after burn-in) from each of C chains of D variables, x_{isc}, $i = 1 : D$, $s = 1 : S$, $c = 1 : C$. Let y_{sc} be a scalar quantity of interest derived from $\mathbf{x}_{1:D,s,c}$ (e.g., $y_{sc} = x_{isc}$ for some chosen i). Define the within-sequence mean and overall mean as

$$\overline{y}_{\cdot c} \triangleq \frac{1}{S}\sum_{s=1}^{S} y_{sc}, \quad \overline{y}_{\cdot\cdot} \triangleq \frac{1}{C}\sum_{c=1}^{C}\overline{y}_{\cdot c} \tag{24.71}$$

Define the between-sequence and within-sequence variance as

$$B \triangleq \frac{S}{C-1}\sum_{c=1}^{C}(\overline{y}_{\cdot c} - \overline{y}_{\cdot\cdot})^2, \quad W \triangleq \frac{1}{C}\sum_{c=1}^{C}\left[\frac{1}{S-1}\sum_{s=1}^{S}(y_{sc} - \overline{y}_{\cdot c})^2\right] \tag{24.72}$$

We can now construct two estimates of the variance of y. The first estimate is W: this should underestimate $\mathrm{var}[y]$ if the chains have not ranged over the full posterior. The second estimate is

$$\hat{V} = \frac{S-1}{S}W + \frac{1}{S}B \tag{24.73}$$

This is an estimate of $\mathrm{var}[y]$ that is unbiased under stationarity, but is an overestimate if the starting points were overdispersed (Gelman and Rubin 1992). From this, we can define the following convergence diagnostic statistic, known as the **estimated potential scale reduction** or **EPSR**:

$$\hat{R} \triangleq \sqrt{\frac{\hat{V}}{W}} \tag{24.74}$$

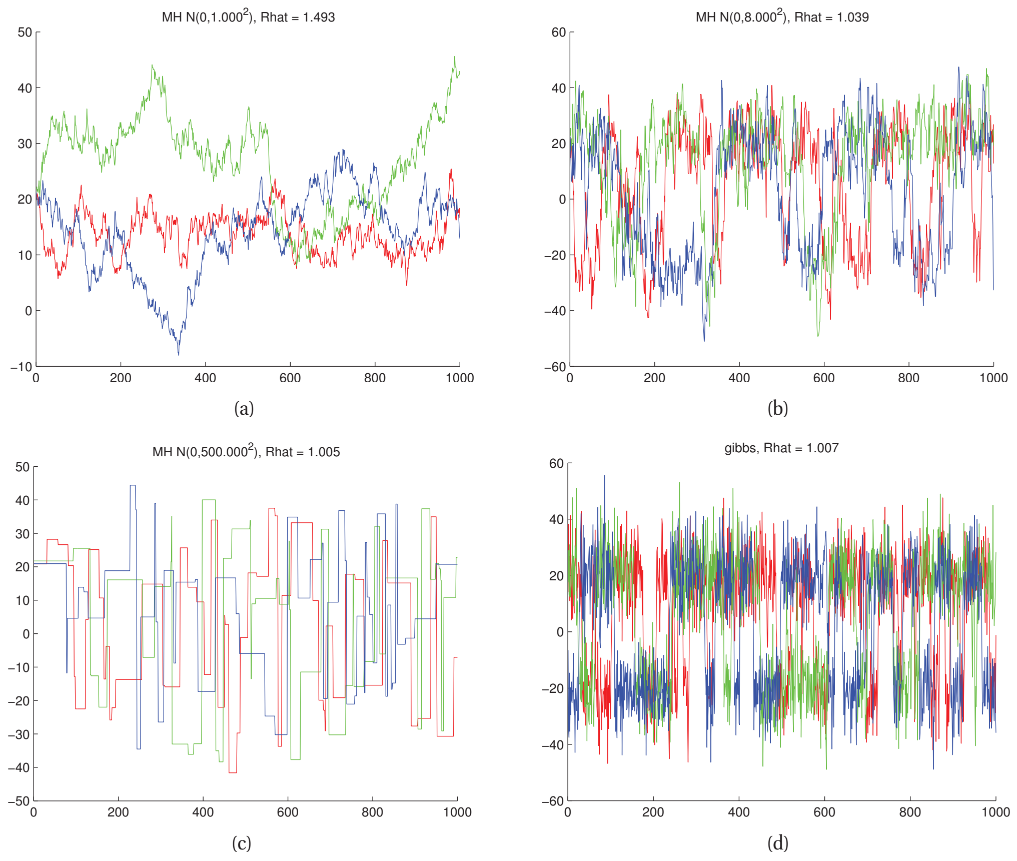


Figure 24.12 Traceplots for MCMC samplers. Each color represents the samples from a different starting point. (a-c) MH with proposal $\mathcal{N}(x'|x, \sigma^2)$ for $\sigma^2 \in \{1, 8, 500\}$, corresponding to Figure 24.7. (d) Gibbs sampling. Figure generated by `mcmcGmmDemo`.

This quantity, which was first proposed in (Gelman and Rubin 1992), measures the degree to which the posterior variance would decrease if we were to continue sampling in the $S \to \infty$ limit. If $\hat{R} \approx 1$ for any given quantity, then that estimate is reliable (or at least is not unreliable). The $\hat{R}$ values for the four samplers in Figure 24.12 are 1.493, 1.039, 1.005 and 1.007. So this diagnostic has correctly identified that the sampler using the first ($\sigma^2 = 1$) proposal is untrustworthy.

24.4.4 Accuracy of MCMC

The samples produced by MCMC are auto-correlated, and this reduces their information content relative to independent or "perfect" samples. We can quantify this as follows.[4] Suppose we want

4. This Section is based on (Hoff 2009, Sec 6.6).

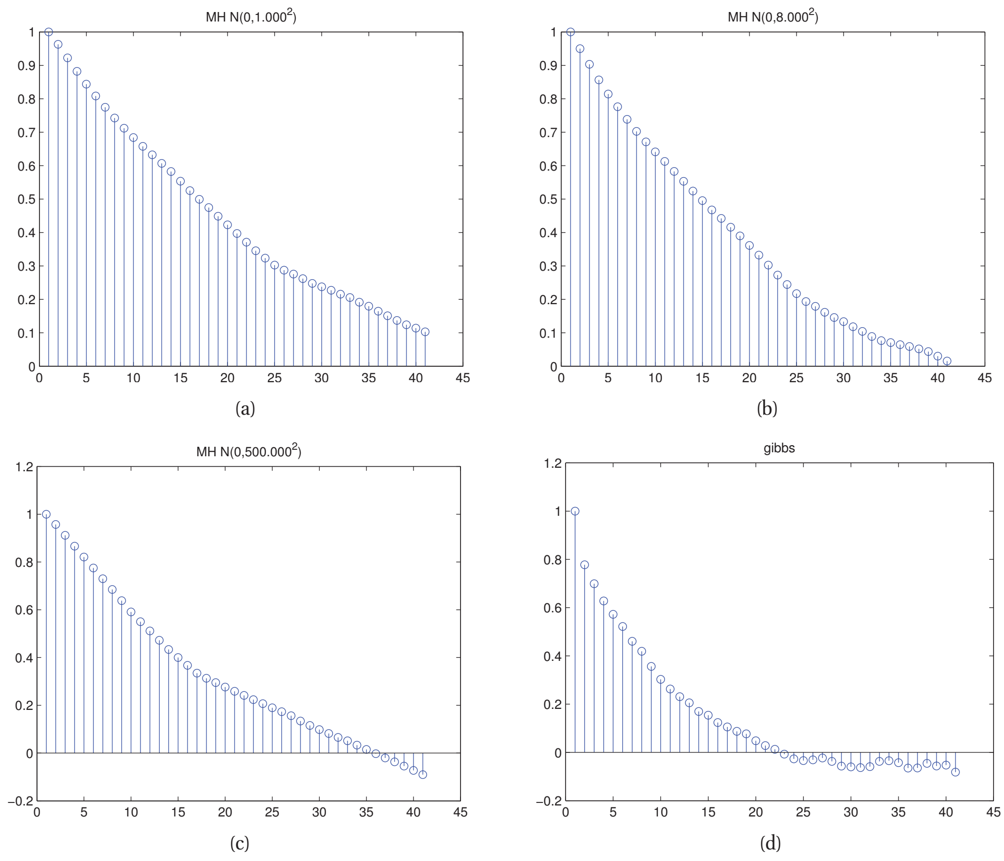


Figure 24.13 Autocorrelation functions corresponding to Figure 24.12. Figure generated by `mcmcGmmDemo`.

to estimate the mean of $f(X)$, for some function f, where $X \sim p()$. Denote the true mean by

$$f^* \triangleq \mathbb{E}[f(X)] \tag{24.75}$$

A Monte Carlo estimate is given by

$$\overline{f} = \frac{1}{S}\sum_{s=1}^{S} f_s \tag{24.76}$$

where $f_s \triangleq f(x_s)$ and $x_s \sim p(x)$. An MCMC estimate of the variance of this estimate is given by

$$\begin{aligned}
\text{Var}_{MCMC}[\overline{f}] &= \mathbb{E}\left[(\overline{f} - f^*)^2\right] & (24.77)\\
&= \mathbb{E}\left[\left\{\frac{1}{S}\sum_{s=1}^{S}(f_s - f^*)\right\}^2\right] & (24.78)\\
&= \frac{1}{S^2}\mathbb{E}\left[\sum_{s=1}^{S}(f_s - f^*)^2\right] + \frac{1}{S^2}\sum_{s\neq t}\mathbb{E}\left[(f_s - f^*)(f_t - f^*)\right] & (24.79)\\
&= \text{Var}_{MC}(\overline{f}) + \frac{1}{S^2}\sum_{s\neq t}\mathbb{E}\left[(f_s - f^*)(f_t - f^*)\right] & (24.80)
\end{aligned}$$

where the first term is the Monte Carlo estimate of the variance if the samples weren't correlated, and the second term depends on the correlation of the samples. We can measure this as follows. Define the sample-based auto-correlation at lag t of a set of samples $f_1, \ldots, f_S$ as follows:

$$\rho_t \triangleq \frac{\frac{1}{S-t}\sum_{s=1}^{S-t}(f_s - \overline{f})(f_{s+t} - \overline{f})}{\frac{1}{S-1}\sum_{s=1}^{S}(f_s - \overline{f})^2} \tag{24.81}$$

This is called the **autocorrelation function** (ACF). This is plotted in Figure 24.13 for our four samplers for the Gaussian mixture model. We see that the ACF of the Gibbs sampler (bottom right) dies off to 0 much more rapidly than the MH samplers, indicating that each Gibbs sample is "worth" more than each MH sample.

A simple method to reduce the autocorrelation is to use **thinning**, in which we keep every n'th sample. This does not increase the efficiency of the underlying sampler, but it does save space, since it avoids storing highly correlated samples.

We can estimate the information content of a set of samples by computing the **effective sample size** (**ESS**) S_{eff}, defined by

$$S_{\text{eff}} \triangleq \frac{\text{Var}_{MC}(f)}{\text{Var}_{MCMC}(\overline{f})} \tag{24.82}$$

From Figure 24.12, it is clear that the effective sample size of the Gibbs sampler is higher than that of the other samplers (in this example).

24.4.5 How many chains?

A natural question to ask is: how many chains should we run? We could either run one long chain to ensure convergence, and then collect samples spaced far apart, or we could run many short chains, but that wastes the burnin time. In practice it is common to run a medium number of chains (say 3) of medium length (say 100,000 steps), and to take samples from each after discarding the first half of the samples. If we initialize at a local mode, we may be able to use all the samples, and not wait for burn-in.

Model	Goal	Method	Reference
Probit	MAP	Gradient	Section 9.4.1
Probit	MAP	EM	Section 11.4.6
Probit	Post	EP	(Nickisch and Rasmussen 2008)
Probit	Post	Gibbs+	Exercise 24.6
Probit	Post	Gibbs with ARS	(Dellaportas and Smith 1993)
Probit	Post	MH using IRLS proposal	(Gamerman 1997)
Logit	MAP	Gradient	Section 8.3.4
Logit	Post	Gibbs+ with Student	(Fruhwirth-Schnatter and Fruhwirth 2010)
Logit	Post	Gibbs+ with KS	(Holmes and Held 2006)

Table 24.1 Summary of some possible algorithms for estimation and inference for binary classification problems using Gaussian priors. Abbreviations: Aux. = auxiliary variable sampling, ARS = adaptive rejection sampling, EP = expectation propagation, Gibbs+ = Gibbs sampling with auxiliary variables, IRLS = iterative reweighted least squares, KS = Kolmogorov Smirnov, MAP = maximum a posteriori, MH = Metropolis Hastings, Post = posterior.

24.5 Auxiliary variable MCMC *

Sometimes we can dramatically improve the efficiency of sampling by introducing dummy **auxiliary variables**, in order to reduce correlation between the original variables. If the original variables are denoted by $\mathbf{x}$, and the auxiliary variables by $\mathbf{z}$, we require that $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})$, and that $p(\mathbf{x}, \mathbf{z})$ is easier to sample from than just $p(\mathbf{x})$. If we meet these two conditions, we can sample in the enlarged model, and then throw away the sampled $\mathbf{z}$ values, thereby recovering samples from $p(\mathbf{x})$. We give some examples below.

24.5.1 Auxiliary variable sampling for logistic regression

In Section 9.4.2, we discussed the latent variable interpretation of probit regression. Recall that this had the form

$$z_i \triangleq \mathbf{w}^T \mathbf{x}_i + \epsilon_i \tag{24.83}$$

$$\epsilon_i \sim \mathcal{N}(0, 1) \tag{24.84}$$

$$y_i = 1 = \mathbb{I}(z_i \geq 0) \tag{24.85}$$

We exploited this representation in Section 11.4.6, where we used EM to find an ML estimate. It is straightforward to convert this into an auxiliary variable Gibbs sampler (Exercise 24.6), since $p(\mathbf{w}|\mathcal{D})$ is Gaussian and $p(z_i|\mathbf{x}_i, y_i, \mathbf{w})$ is truncated Gaussian, both of which are easy to sample from.

Now let us discuss how to derive an auxiliary variable Gibbs sampler for logistic regression. Let ϵ_i follow a **logistic distribution**, with pdf

$$p_{\text{Logistic}}(\epsilon) = \frac{e^{-\epsilon}}{(1 + e^{-\epsilon})^2} \tag{24.86}$$

with mean $\mathbb{E}[\epsilon] = 0$ and variance $\text{var}[\epsilon] = \pi^2/3$. The cdf has the form $F(\epsilon) = \text{sigm}(\epsilon)$, which is the logistic function. Since $y_i = 1$ iff $\mathbf{w}^T\mathbf{x}_i + \epsilon > 0$, we have, by symmetry, that

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \int_{-\mathbf{w}^T\mathbf{x}_i}^{\infty} f(\epsilon)d\epsilon = \int_{-\infty}^{\mathbf{w}^T\mathbf{x}_i} f(\epsilon)d\epsilon = F(\mathbf{w}^T\mathbf{x}_i) = \text{sigm}(\mathbf{w}^T\mathbf{x}_i) \quad (24.87)$$

as required.

We can derive an auxiliary variable Gibbs sampler by sampling from $p(\mathbf{z}|\mathbf{w}, \mathcal{D})$ and $p(\mathbf{w}|\mathbf{z}, \mathcal{D})$. Unfortunately, sampling directly from $p(\mathbf{w}|\mathbf{z}, \mathcal{D})$ is not possible. One approach is to define $\epsilon_i \sim \mathcal{N}(0, \lambda_i)$, where $\lambda_i = (2\psi_i)^2$ and $\psi_i \sim$ KS, the Kolmogorov Smirnov distribution, and then to sample $\mathbf{w}$, $\mathbf{z}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\psi}$ (Holmes and Held 2006).

A simpler approach is to approximate the logistic distribution by the Student distribution (Albert and Chib 1993). Specifically, we will make the approximation $\epsilon_i \sim \mathcal{T}(0, 1, \nu)$, where $\nu \approx 8$. We can now use the scale mixture of Gaussians representation of the Student to simplify inference. In particular, we write

$$\lambda_i \sim \text{Ga}(\nu/2, \nu/2) \quad (24.88)$$
$$\epsilon_i \sim \mathcal{N}(0, \lambda_i^{-1}) \quad (24.89)$$
$$z_i \triangleq \mathbf{w}^T\mathbf{x}_i + \epsilon_i \quad (24.90)$$
$$y_i = 1|z_i = \mathbb{I}(z_i \geq 0) \quad (24.91)$$

All of the full conditionals now have a simple form; see Exercise 24.7 for the details.

Note that if we set $\nu = 1$, then $z_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, 1)$, which is equivalent to probit regression (see Section 9.4). Rather than choosing between probit or logit regression, we can simply estimate the ν parameter. There is no convenient conjugate prior, but we can consider a finite range of possible values and evaluate the posterior as follows:

$$p(\nu|\boldsymbol{\lambda}) \propto p(\nu) \prod_{i=1}^{N} \frac{1}{\Gamma(\nu/2)(\nu/2)^{\nu/2}} \lambda_i^{\nu/2-1} e^{-\nu\lambda_i/2} \quad (24.92)$$

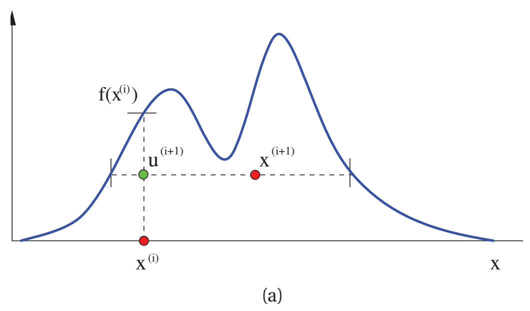
Furthermore, if we define $\mathbf{V}_0 = v_0\mathbf{I}$, we can sample v_0 as well. For example, suppose we use a IG(δ_1, δ_2) prior for v_0. The posterior is given by $p(v_0|\mathbf{w}) = \text{IG}(\delta_1 + \frac{1}{2}D, \delta_2 + \frac{1}{2}\sum_{j=1}^{D} w_j^2)$. This can be interleaved with the other Gibbs sampling steps, and provides an appealing Bayesian alternative to cross validation for setting the strength of the regularizer.

See Table 24.1 for a summary of various algorithms for fitting probit and logit models. Many of these methods can also be extended to the multinomial logistic regression case. For details, see (Scott 2009; Fruhwirth-Schnatter and Fruhwirth 2010).

24.5.2 Slice sampling

Consider sampling from a univariate, but multimodal, distribution $\tilde{p}(x)$. We can sometimes improve the ability to make large moves by adding an auxiliary variable u. We define the joint distribution as follows:

$$\hat{p}(x, u) = \begin{cases} 1/Z_p & \text{if } 0 \leq u \leq \tilde{p}(x) \\ 0 & \text{otherwise} \end{cases} \quad (24.93)$$



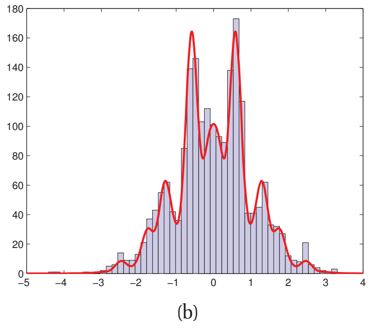
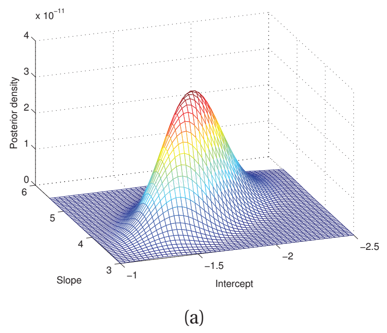


Figure 24.14 (a) Illustration of the principle behind slice sampling. Given a previous sample x^i, we sample u^{i+1} uniformly on $[0, f(x^i)]$, where f is the target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. Source: Figure 15 of (Andrieu et al. 2003) . Used with kind permission of Nando de Freitas. (b) Slice sampling in action. Figure generated by `sliceSamplingDemo1d`.



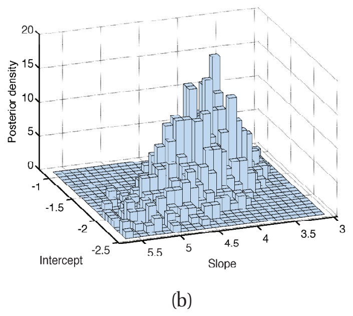


Figure 24.15 Binomial regression for 1d data. (a) Grid approximation to posterior. (b) Slice sampling approximation. Figure generated by `sliceSamplingDemo2d`.

where $Z_p = \int \tilde{p}(x)dx$. The marginal distribution over x is given by

$$\int \hat{p}(x, u)du = \int_0^{\tilde{p}(x)} \frac{1}{Z_p} du = \frac{\tilde{p}(x)}{Z_p} = p(x) \tag{24.94}$$

so we can sample from $p(x)$ by sampling from $\hat{p}(x, u)$ and then ignoring u. The full conditionals have the form

$$p(u|x) = U_{[0,\tilde{p}(x)]}(u) \tag{24.95}$$

$$p(x|u) = U_A(x) \tag{24.96}$$

where $A = \{x : \tilde{p}(x) \geq u\}$ is the set of points on or above the chosen height u. This corresponds to a slice through the distribution, hence the term **slice sampling** (Neal 2003a). See Figure 24.14(a).

In practice, it can be difficult to identify the set A. So we can use the following approach: construct an interval $x_{min} \leq x \leq x_{max}$ around the current point x^s of some width. We then test to see if each end point lies within the slice. If it does, we keep extending in that direction until it lies outside the slice. This is called **stepping out**. A candidate value x' is then chosen uniformly from this region. If it lies within the slice, it is kept, so $x^{s+1} = x'$. Otherwise we shrink the region such that x' forms one end and such that the region still contains x^s. Then another sample is drawn. We continue in this way until a sample is accepted.

To apply the method to multivariate distributions, we can sample one extra auxiliary variable for each dimension. The advantage of slice sampling over Gibbs is that it does not need a specification of the full-conditionals, just the unnormalized joint. The advantage of slice sampling over MH is that it does not need a user-specified proposal distribution (although it does require a specification of the width of the stepping out interval).

Figure 24.14(b) illustrates the algorithm in action on a synthetic 1d problem. Figure 24.15 illustrates its behavior on a slightly harder problem, namely binomial logistic regression. The model has the form

$$y_i \sim \text{Bin}(n_i, \text{logit}(\beta_1 + \beta_2 x_i)) \tag{24.97}$$

We use a vague Gaussian prior for the β_j's. Figure 24.15(a) shows a grid-based approximation to the posterior, and Figure 24.15(b) shows a sample-based approximation. In this example, the grid is faster to compute, but for any problem with more than 2 dimensions, the grid approach is infeasible.

24.5.3 Swendsen Wang

Consider an Ising model of the following form:

$$p(\mathbf{x}) = \frac{1}{Z}\prod_e f_e(\mathbf{x}_e) \tag{24.98}$$

where $\mathbf{x}_e = (x_i, x_j)$ for edge $e = (i, j)$, $x_i \in \{+1, -1\}$, and the edge factor f_e is defined by $\begin{pmatrix} e^J & e^{-J} \\ e^{-J} & e^J \end{pmatrix}$, where J is the edge strength. Gibbs sampling in such models can be slow when J is large in absolute value, because neighboring states can be highly correlated. The **Swendsen Wang** algorithm (Swendsen and Wang 1987) is a 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. [5] These are called **bond variables**, and will be denoted by $\mathbf{z}$. We then define an extended model $p(\mathbf{x}, \mathbf{z})$ of the form

$$p(\mathbf{x}, \mathbf{z}) = \frac{1}{Z'}\prod_e g_e(\mathbf{x}_e, z_e) \tag{24.99}$$

where $z_e \in \{0, 1\}$, and we define the new factor as follows: $g_e(\mathbf{x}_e, z_e = 0) = \begin{pmatrix} e^{-J} & e^{-J} \\ e^{-J} & e^{-J} \end{pmatrix}$, and $g_e(\mathbf{x}_e, z_e = 1) = \begin{pmatrix} e^J - e^{-J} & 0 \\ 0 & e^J - e^{-J} \end{pmatrix}$. It is clear that $\sum_{z_e=0}^{1} g_e(\mathbf{x}_e, z_e) = f_e(\mathbf{x}_e)$,

5. Our presentation of the method is based on some notes by David Mackay, available from `http://www.inference.phy.cam.ac.uk/mackay/itila/swendsen.pdf`.

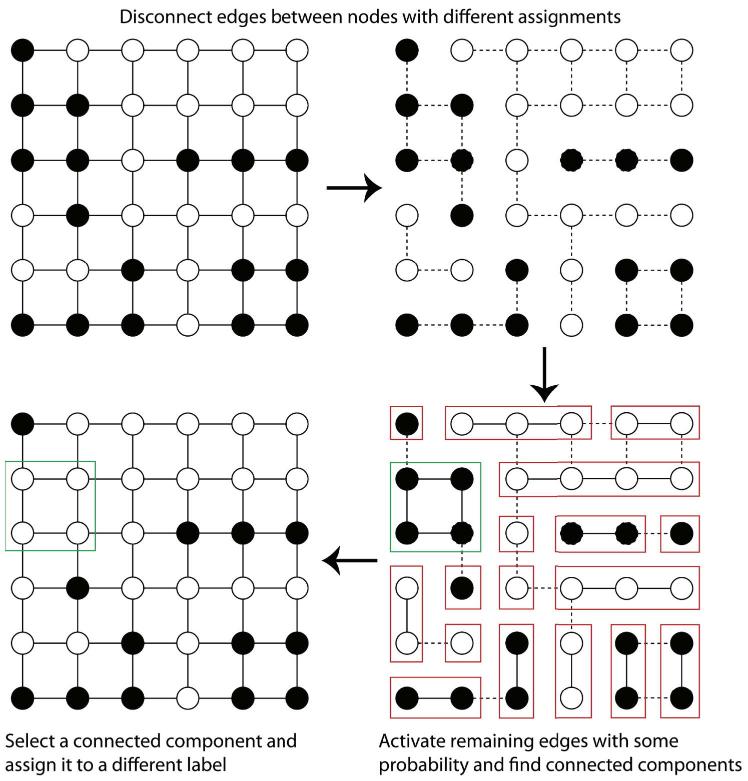

Disconnect edges between nodes with different assignments
Activate remaining edges with some probability and find connected components
Select a connected component and assign it to a different label

Figure 24.16 Illustration of the Swendsen Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

and hence that $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})$. So if we can sample from this extended model, we can just throw away the $\mathbf{z}$ samples and get valid $\mathbf{x}$ samples from the original distribution.

Fortunately, it is easy to apply Gibbs sampling to this extended model. The full conditional $p(\mathbf{z}|\mathbf{x})$ factorizes over the edges, since the bond variables are conditionally independent given the node variables. Furthermore, the full conditional $p(z_e|\mathbf{x}_e)$ 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 z_e to 1 with probability $p = 1 - e^{-2J}$, otherwise we set it to 0. In Figure 24.16 (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 24.16 (bottom right), the bonds that are randomly turned on are represented by solid edges.

To sample $p(\mathbf{x}|\mathbf{z})$, 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, since the off-diagonal terms in the $g_e(\mathbf{x}_e, z_e = 1)$ factor are 0. Pick a state ± 1 uniformly at random, and force all the variables in this component to adopt this new state. This is illustrated in Figure 24.16 (bottom left), where the green square denotes the selected connected component, and we choose to force all nodes within in to enter the white state.

The validity of this algorithm is left as an exercise, as is the extension to handle local evidence and non-stationary potentials.

It should be intuitively clear that Swendsen Wang makes much larger moves through the state space than Gibbs sampling. In fact, SW mixes much faster than Gibbs sampling on 2d lattice Ising models for a variety of values of the coupling parameter, provided $J > 0$. 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 exponentiall 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, computation in this regime is provably hard for any algorithm (Jerrum and Sinclair 1993, 1996).

24.5.4 Hybrid/Hamiltonian MCMC *

In this section, we briefly mention a way to perform MCMC sampling for continuous state spaces, for which we can compute the gradient of the (unnormalized) log-posterior. This is the case in neural network models, for example.

The basic idea is to think of the parameters as a particle in space, and to create auxiliary variables which represent the "momentum" of this particle. We then update this parameter/ momentum pair according to certain rules (see e.g., (Duane et al. 1987; Neal 1993; MacKay 2003; Neal 2010) for details). The resulting method is called **hybrid MCMC** or **Hamiltonian MCMC**. The two main parameters that the user must specify are how many **leapfrog steps** to take when updating the position/ momentum, and how big to make these steps. Performance can be quite sensitive to these parameters (although see (Hoffman and Gelman 2011) for a recent way to set them automatically). This method can be combined with stochastic gradient descent (Section 8.5.2) in order to handle large datasets, as explained in (Ahn et al. 2012).

Recently, a more powerful extension of this method has been developed, that exploits second-order gradient information. See (Girolami et al. 2010) for details.

24.6 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.

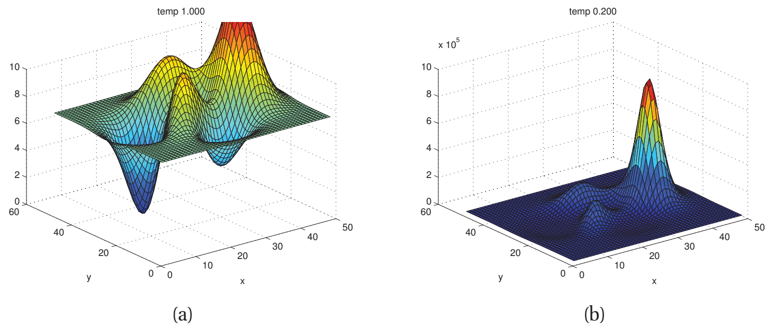


Figure 24.17 An energy surface at different temperatures. Note the different vertical scales. (a) $T = 1$. (b) $T = 0.5$. Figure generated by `saDemoPeaks`.

24.6.1 Simulated annealing

Simulated annealing (Kirkpatrick et al. 1983) is a stochastic algorithm that attempts to find the global optimum of a black-box function $f(\mathbf{x})$. It is closely related to the Metropolis-Hastings algorithm for generating samples from a probability distribution, which we discussed in Section 24.3. SA can be used for both discrete and continuous optimization.

The method is inspired by statistical physics. The key quantity is the **Boltzmann distribution**, which specifies that the probability of being in any particular state $\mathbf{x}$ is given by

$$p(\mathbf{x}) \propto \exp(-f(\mathbf{x})/T) \tag{24.100}$$

where $f(\mathbf{x})$ is the "energy" of the system and T is the computational temperature. As the temperature approaches 0 (so the system is cooled), the system spends more and more time in its minimum energy (most probable) state.

Figure 24.17 gives an example of a 2d function at different temperatures. At high temperatures, $T \gg 1$, the surface is approximately flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools, the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is possible to "track" the largest peak, and thus find the global optimum. This is an example of a **continuation method**.

We can generate an algorithm from this as follows. At each step, sample a new state according to some proposal distribution $\mathbf{x}' \sim q(\cdot|\mathbf{x}_k)$. For real-valued parameters, this is often simply a random walk proposal, $\mathbf{x}' = \mathbf{x}_k + \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. For discrete optimization, other kinds of local moves must be defined.

Having proposed a new state, we compute

$$\alpha = \exp\left((f(\mathbf{x}) - f(\mathbf{x}'))/T\right) \tag{24.101}$$

We then accept the new state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}'$) with probability $\min(1, \alpha)$, otherwise we stay in the current state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}_k$). This means that if the new state has lower energy (is more probable), we will definitely accept it, but it it has higher energy (is less probable), we might still accept, depending on the current temperature. Thus the algorithm allows "down-hill" moves in probability space (up-hill in energy space), but less frequently as the temperature drops.