Assignment			$\max_C$
a^0	b^0	d^0	300,000
a^0	b^0	d^1	300,000
a^0	b^1	d^0	5,000,000
a^0	b^1	d^1	500
a^1	b^0	d^0	100
a^1	b^0	d^1	1,000,000
a^1	b^1	d^0	100,000
a^1	b^1	d^1	100,000

$\beta_1(A, B, D)$

Assignment		$\max_{A,C}$
b^0	d^0	300,000
b^0	d^1	1,000,000
b^1	d^0	5,000,000
b^1	d^1	100,000

$\mu_{1,2}(B, D)$

Assignment			$\max_A$
b^0	c^0	d^0	300,000
b^0	c^0	d^1	1,000,000
b^0	c^1	d^0	300,000
b^0	c^1	d^1	100
b^1	c^0	d^0	500
b^1	c^0	d^1	100,000
b^1	c^1	d^0	5,000,000
b^1	c^1	d^1	100,000

$\beta_2(B, C, D)$

Figure 13.3 The max-marginals for the Misconception **example.** Listed are the beliefs for the two cliques and the sepset.

are exactly equivalent. Thus, we can show that the analogue to equation (10.9) holds also for max-product:

$$\mu_{i,j}(\boldsymbol{S}_{i,j}) = \delta_{j \to i}(\boldsymbol{S}_{i,j}) \cdot \delta_{i \to j}(\boldsymbol{S}_{i,j}). \tag{13.11}$$

In particular, this equivalence holds at convergence, so that a clique's max-marginal over a sepset can be computed from the max-product messages.

13.3.2 Message Passing as Reparameterization

reparameterization

clique tree measure

Somewhat surprisingly, as for the sum-product case, we can view the max-product message passing steps as *reparameterizing* the original distribution, in a way that leaves the distribution invariant. More precisely, we view a set of beliefs β_i and sepset messages $\mu_{i,j}$ in a max-product clique tree as defining a *measure* using equation (10.11), precisely as for sum-product trees:

$$Q_{\mathcal{T}} = \frac{\prod_{i \in \mathcal{V}_{\mathcal{T}}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_{\mathcal{T}}} \mu_{i,j}(\boldsymbol{S}_{i,j})}. \tag{13.12}$$

When we begin a run of max-product belief propagation, the initial potentials are simply the initial potentials in Φ, and the messages are all $\mathbf{1}$, so that $Q_{\mathcal{T}}$ is precisely $\tilde{P}_\Phi$. Examining the proof of corollary 10.3, we can see that it does not depend on the definition of the messages in terms of summing out the beliefs, but only on the way in which the messages are then used to update the receiving beliefs. Therefore, the proof of the theorem holds unchanged for max-product message passing, proving the following result:

Proposition 13.2 *In an execution of max-product message passing (whether belief propagation or belief update) in a clique tree, equation (13.12) holds throughout the algorithm.*

We can now directly conclude the following result:

Theorem 13.5 *Let $\{\beta_i\}$ and $\{\mu_{i,j}\}$ be the max-calibrated set of beliefs obtained from executing max-product message passing, and let $Q_\mathcal{T}$ be the distribution induced by these beliefs. Then $Q_\mathcal{T}$ is a representation of the distribution $\tilde{P}_\Phi$ that also satisfies the max-product calibration constraints of equation (13.10).*

Example 13.9 *Continuing with example 13.8, it is straightforward to confirm that the original measure $\tilde{P}_\Phi$ can be reconstructed directly from the max-marginals and the sepset message. For example, consider the entry $\tilde{P}_\Phi(a^1, b^0, c^1, d^0) = 100$. According to equation (10.10), the clique tree measure is:*

$$\frac{\beta_1(a^1, b^0, d^0)\beta_2(b^0, c^1, d^0)}{\mu_{1,2}(b^0, d^0)} = \frac{100 \cdot 300,000}{300,000} = 100,$$

as required. The equivalence for other entries can be verified similarly. ■

Comparing this computation to example 10.6, we see that the sum-product clique tree and the max-product clique tree both induce reparameterizations of the original measure $\tilde{P}_\Phi$, but these two reparameterizations are different, since they must satisfy different constraints.

13.3.3 Decoding Max-Marginals

Given the max-marginals, can we find the actual MAP assignment? In the case of variable elimination, we had the max-marginal only for a single variable X_i (the last to be eliminated). Therefore, although we could identify the assignment for X_i in the MAP assignment, we had to perform a traceback procedure to compute the assignments to the other variables. Now the situation appears different: we have max-marginals for all of the variables in the network. Can we use this property to simplify this process?

One obvious solution is to use the max-marginal for each variable X_i to compute its own optimal assignment, and thereby compose a full joint assignment to all variables. However, this simplistic approach may not always work.

Example 13.10 *Consider a simple XOR-like distribution $P(X_1, X_2)$ that gives probability 0.1 to the assignments where $X_1 = X_2$ and 0.4 to the assignments where $X_1 \neq X_2$. In this case, for each assignment to X_1, there is a corresponding assignment to X_2 whose probability is 0.4. Thus, the max-marginal of X_1 is the symmetric factor $(0.4, 0.4)$, and similarly for X_2. Indeed, we can choose either of the two values for X_1 and complete it to a MAP assignment, and similarly for X_2. However, if we choose the values for X_1 and X_2 in an* inconsistent *way, we may get an assignment whose probability is much lower. Thus, our joint assignment cannot be chosen by separately optimizing the individual max-marginals.* ■

Recall that we defined a set of node beliefs to be unambiguous if each belief has a unique maximal value. This condition prevents symmetric cases like the one in the preceding example. Indeed, it is not difficult to show the following result:

Proposition 13.3 *The following two conditions are equivalent:*

- *The set of node beliefs $\{\mathit{MaxMarg}_{\tilde{P}_\Phi}(X_i) : X_i \in \mathcal{X}\}$ is unambiguous, with*

$$x_i^* = \arg\max_{x_i} \mathit{MaxMarg}_{\tilde{P}_\Phi}(X_i)$$

the unique optimizing value for X_i;

- *$\tilde{P}_\Phi$ has a unique MAP assignment $(x_1^*, \ldots, x_n^*)$.*

See exercise 13.8. For generic probability measures, the assumption of unambiguity is not overly stringent, since we can always break ties by introducing a slight random perturbation into all of the factors, making all of the elements in the joint distribution have slightly different probabilities. However, if the distribution has special structure — deterministic relationships or shared parameters — that we want to preserve, this type of ambiguity may be unavoidable.

Thus, if there are no ties in any of the calibrated node beliefs, we can find the unique MAP assignment by locally optimizing the assignment to each variable separately.

If there are ties in the node beliefs, our task can be reformulated as follows:

Definition 13.3 local optimality

Let $\beta_i(\boldsymbol{C}_i)$ be a belief in a max-calibrated clique tree. We say that an assignment ξ^* has the local optimality property *if, for each clique $\boldsymbol{C}_i$ in the tree, we have that*

$$\xi^*\langle \boldsymbol{C}_i \rangle \in \arg\max_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i), \tag{13.13}$$

that is, the assignment to $\boldsymbol{C}_i$ in ξ^* optimizes the $\boldsymbol{C}_i$ belief. The task of finding a locally optimal assignment ξ^* given a max-calibrated set of beliefs is called the decoding *task.* ■

decoding

traceback

Solving the decoding task in the ambiguous case can be done using a *traceback* procedure as in algorithm 13.1. However, local optimality provides us with a simple, local test for *verifying* whether a given assignment is the MAP assignment:

Theorem 13.6

Let $\beta_i(\boldsymbol{C}_i)$ be a set of max-calibrated beliefs in a clique tree $\mathcal{T}$, with $\mu_{i,j}$ the associated sepset beliefs. Let $Q_\mathcal{T}$ be the clique tree measure defined as in equation (13.12). Then an assignment ξ^* satisfies the local optimality property relative to the beliefs $\{\beta_i(\boldsymbol{C}_i)\}_{i \in \mathcal{V}_\mathcal{T}}$ if and only if it is the global MAP assignment relative to $Q_\mathcal{T}$.

Proof The proof of the "if" direction follows directly from our previous results. We have that $Q_\mathcal{T}$ is max-calibrated, and hence is a fixed point of the max-product algorithm. (In other words, if we run max-product inference on the distribution defined by $Q_\mathcal{T}$, we would get precisely the beliefs $\beta_i(\boldsymbol{C}_i)$.) Thus, these beliefs are max-marginals of $Q_\mathcal{T}$. If ξ^* is the MAP assignment to $Q_\mathcal{T}$, it must maximize each one of its max-marginals, proving the desired result.

The proof of the only if direction requires the following lemma, which plays an even more significant role in later analyses.

Lemma 13.1

Let ϕ be a factor over scope $\boldsymbol{Y}$ and ψ be a factor over scope $\boldsymbol{Z} \subset \boldsymbol{Y}$ such that ψ is a max-marginal of ϕ over $\boldsymbol{Z}$; that is, for any $\boldsymbol{z}$:

$$\psi(\boldsymbol{z}) = \max_{\boldsymbol{y} \sim \boldsymbol{z}} \phi(\boldsymbol{y}).$$

Let $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} \phi(\boldsymbol{y})$. Then $\boldsymbol{y}^*$ is also an optimal assignment for the factor ϕ/ψ, where, as usual, we take $\psi(\boldsymbol{y}^*) = \psi(\boldsymbol{y}^*\langle \boldsymbol{Z} \rangle)$.

Proof Recall that, due to the properties of max-marginalization, each entry $\psi(\boldsymbol{z})$ arises from some entry $\phi(\boldsymbol{y})$ such that $\boldsymbol{y} \sim \boldsymbol{z}$. Because $\boldsymbol{y}^*$ achieves the optimal value in ϕ, and ψ is the

max-marginal of ϕ, we have that $\boldsymbol{z}^*$ achieves the optimal value in ψ. Hence, $\phi(\boldsymbol{y}^*) = \psi(\boldsymbol{z}^*)$, so that $\left(\frac{\phi}{\psi}\right)(\boldsymbol{y}^*) = 1$. Now, consider any other assignment $\boldsymbol{y}$ and the assignment $\boldsymbol{z} = \boldsymbol{y}\langle \boldsymbol{Z} \rangle$. Either the value of $\boldsymbol{z}$ is obtained from $\boldsymbol{y}$, or it is obtained from some other $\boldsymbol{y}'$ whose value is larger. In the first case, we have that $\phi(\boldsymbol{y}) = \psi(\boldsymbol{z})$, so that $\left(\frac{\phi}{\psi}\right)(\boldsymbol{y}) = 1$. In the second case, we have that $\phi(\boldsymbol{y}) < \psi(\boldsymbol{z})$ and $\left(\frac{\phi}{\psi}\right)(\boldsymbol{y}) < 1$. In either case,

$$\left(\frac{\phi}{\psi}\right)(\boldsymbol{y}) \leq \left(\frac{\phi}{\psi}\right)(\boldsymbol{y}^*),$$

as required. ■

To prove the only-if direction, we first rewrite the clique tree distribution of equation (13.12) in a directed way. We select a root clique $\boldsymbol{C}_r$; for each clique $i \neq r$, let $\pi(i)$ be the parent clique of i in this rooted tree. We then assign each sepset $\boldsymbol{S}_{i,\pi(i)}$ to the *child* clique i. Note that, because each clique has at most one parent, each clique is assigned at most one sepset. Thus, we obtain the following rewrite of equation (13.12):

$$\beta_r(\boldsymbol{C}_r) \prod_{i \neq r} \frac{\beta_i(\boldsymbol{C}_i)}{\mu_{i,\pi(i)}(\boldsymbol{S}_{i,\pi(i)})}. \tag{13.14}$$

Now, let ξ^* be an assignment that satisfies the local optimality property. By assumption, it optimizes every one of the beliefs. Thus, the conditions of lemma 13.1 hold for each of the ratios in this product, and for the first term involving the root clique. Thus, ξ^* also optimizes each one of the terms in this product, and therefore it optimizes the product as a whole. It must therefore be the MAP assignment. ■

As we will see, these concepts and related results have important implications in some of our later derivations.

13.4 Max-Product Belief Propagation in Loopy Cluster Graphs

In section 11.3 we applied the sum-product message passing using the clique tree algorithm to a loopy cluster graph, obtaining an approximate inference algorithm. In the same way, we can generalize max-product message passing to the case of cluster graphs. The algorithms that we present in this section are directly analogous to their sum-product counterparts in section 11.3. However, as we discuss, the guarantees that we can provide are much stronger in this case.

13.4.1 Standard Max-Product Message Passing

As for the case of clique trees, the algorithm divides into two phases: computing the beliefs using message passing and using those beliefs to identify a single joint assignment.

13.4.1.1 Message Passing Algorithm

The message passing algorithm is straightforward: it is precisely the same as the algorithm of algorithm 11.1, except that we use the procedure of algorithm 13.2 in place of the SP-Message

procedure. As for sum-product, there are no guarantees that this algorithm will converge. Indeed, in practice, it tends to converge somewhat less often than the sum-product algorithm, perhaps because the averaging effect of the summation operation tends to smooth out messages, and reduce oscillations. Many of the same ideas that we discussed in box 11.B can be used to improve convergence in this algorithm as well.

At convergence, the result will be a set of calibrated clusters: As for sum-product, if the clusters are not calibrated, convergence has not been achieved, and the algorithm will continue iterating. However, the resulting beliefs will not generally be the exact max-marginals; these resulting beliefs are often called *pseudo-max-marginals*.

pseudo-max-marginal

As we saw in section 11.3.3.1 for sum-product, the distribution invariance property that holds for clique trees is a consequence only of the message passing procedure, and does not depend on the assumption that the cluster graph is a tree. The same argument holds here; thus, proposition 13.2 can be used to show that max-product message passing in a cluster graph is also simply reparameterizing the distribution:

Corollary 13.3

In an execution of max-product message passing (whether belief propagation or belief update) in a cluster graph, the invariant equation (10.10) holds initially, and after every message passing step.

13.4.1.2 Decoding the Pseudo-Max-Marginals

Given a set of pseudo-max-marginals, we now have to solve the decoding problem in order to identify a joint assignment. In general, we cannot expect this assignment to be the exact MAP, but we can hope for some reasonable approximation. But how do we identify such an assignment? It turns out that our ability to do so depends strongly on whether there exists some assignment that satisfies the local optimality property of definition 13.3 for the max-calibrated beliefs in the cluster graph. Unlike in the case of clique trees, such a joint assignment does not necessarily exist:

Example 13.11

Consider a cluster graph with the three clusters $\{A, B\}, \{B, C\}, \{A, C\}$ and the beliefs

	a^1	a^0
b^1	1	2
b^0	2	1

	b^1	b^0
c^1	1	2
c^0	2	1

	a^1	a^0
c^1	1	2
c^0	2	1

These beliefs are max-calibrated, in that all messages are $(2, 2)$. However, there is no joint assignment that maximizes all of the cluster beliefs simultaneously. For example, if we select a^0, b^1, we maximize the value in the A, B belief. We can now select c^0 to maximize the value in the B, C belief. However, we now have a nonmaximizing assignment a^0, c^0 in the A, C belief. No matter which assignment of values we select in this example, we do not obtain a single joint assignment that maximizes all three beliefs. ■

frustrated loop

Loops such as this are often called *frustrated*.

In other cases, a locally optimal joint assignment does exist. In particular, **when all the node beliefs are all unambiguous, it is not difficult to show that all of the cluster beliefs also have a unique maximizing assignment, and that these local cluster-maximizing assignments are necessarily consistent with each other** (exercise 13.9). However, there are also other cases where the node beliefs are ambiguous, and yet a locally optimal joint assignment exists:

Example 13.12

Consider a cluster graph of the same structure as in example 13.11, but with the beliefs:

	a^1	a^0
b^1	2	1
b^0	1	2

	b^1	b^0
c^1	2	1
c^0	1	2

	a^1	a^0
c^1	2	1
c^0	1	2

In this case, the beliefs are ambiguous, yet a locally optimal joint assignment exists (both a^1, b^1, c^1 and a^0, b^0, c^0 are locally optimal). ■

In general, the decoding problem in a loopy cluster graph is not a trivial task. Recall that, in clique trees, we could simply choose any of the maximizing assignments for the beliefs at a clique, and be assured that it could be extended into a joint MAP assignment. Here, as illustrated by example 13.11, we may make a choice for one cluster that cannot be extended into a consistent joint assignment. In that example, of course, there is no assignment that works. However, it is not difficult to construct examples where one choice of locally optimal assignments would give rise to a consistent joint assignment, whereas another would not (exercise 13.10).

How do we find a locally optimal joint assignment, if one exists? Recall from the definition that an assignment is locally optimal if and only if it selects one of the optimizing assignments in every single cluster. Thus, we can essentially label the assignments in each cluster as either "legal" if they optimize the belief or "illegal" if they do not. We now must search for an assignment to $\mathcal{X}$ that results in a legal value for each cluster. This problem is precisely a *constraint satisfaction problem (CSP)*, where the constraints are derived from the local optimality condition. More precisely, a constraint satisfaction problem can be defined in terms of a Markov network (or factor graph) where all of the entries in the beliefs are either 0 or 1. The CSP problem is now one of finding an assignment whose (unnormalized) measure is 1, if one exists, and otherwise reporting failure. In other words, the CSP problem is simply that of finding the MAP assignment in this model with $\{0, 1\}$-valued beliefs. The field of CSP algorithms is a large one, and a detailed survey is outside the scope of the book; see section 13.9 for some background reading. We note, however, that the CSP problem is itself $\mathcal{NP}$-hard, and therefore we have no guarantees that a locally optimal assignment, even if one exists, can be found efficiently.

constraint satisfaction problem

Thus, given a max-product calibrated cluster graph, we can convert it to a discrete-valued CSP by simply taking the belief in each cluster, changing each assignment that locally optimizes the belief to 1 and all other assignments to 0. We then run some CSP solution method. If the outcome is an assignment that achieves 1 in every belief, this assignment is guaranteed to be a locally optimal assignment. Otherwise, there is no locally optimal assignment. In this case, we must resort to the use of alternative solution methods. One heuristic in this latter situation is to use information obtained from the max-product propagation to construct a partial assignment. For example, assume that a variable X_i is unambiguous in the calibrated cluster graph, so that the only value that locally optimizes its node marginal is x_i. In this case, we may

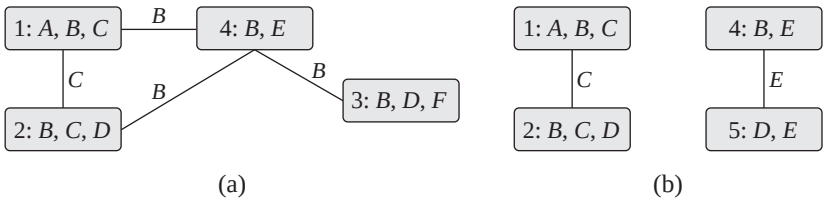


Figure 13.4 Two induced subgraphs derived from figure 11.3a. (a) Graph over $\{B, C\}$; (b) Graph over $\{C, E\}$.

decide to restrict attention only to assignments where $X_i = x_i$. In many real-world problems, a large fraction of the variables in the network are unambiguous in the calibrated max-product cluster graph. Thus, this heuristic can greatly simplify the model, potentially even allowing exact methods (such as clique tree inference) to be used for the resulting restricted model. We note, however, that the resulting assignment would not necessarily satisfy the local optimality condition, and all of the guarantees we will present hold only under that assumption.

13.4.1.3 Strong Local Maximum

What type of guarantee can we provide for a decoded assignment from the pseudo-max-marginals produced by the max-product belief propagation algorithm? It is certainly not the case that this assignment is the MAP assignment; nor is it even the case that we can guarantee that the probability of this assignment is "close" in any sense to that of the true MAP assignment.

strong local maximum

However, if we can construct a locally optimal assignment ξ^* relative to the beliefs produced by max-product BP, we can prove that ξ^* is a *strong local maximum*, in the following sense: For certain subsets of variables $\boldsymbol{Y} \subset \mathcal{X}$, there is no assignment ξ' that is higher-scoring than ξ^* and differs from it only in the assignment to $\boldsymbol{Y}$. These subsets $\boldsymbol{Y}$ are those that induce any disjoint union of subgraphs each of which contains at most a single loop (including trees, which contain no loops).

Definition 13.4

induced subgraph

Let $\mathcal{U}$ be a cluster graph over $\mathcal{X}$, and $\boldsymbol{Y} \subset \mathcal{X}$ be some set of variables. We define the induced subgraph *$\mathcal{U}_{\boldsymbol{Y}}$ to be the subgraph of clusters and sepsets in $\mathcal{U}$ that contain some variable in $\boldsymbol{Y}$.* ■

This definition is most easily understood in the context of a pairwise Markov network, where the cluster graph is simply the set of edges in the MRF and the sepsets are the individual variables. In this case, the induced subgraph for a set $\boldsymbol{Y}$ is simply the set of nodes corresponding to $\boldsymbol{Y}$ and any edges that contain them. In a more general cluster graph, the result is somewhat more complex:

Example 13.13

Consider the cluster graph of figure 11.3a. Figure 13.4a shows the induced subgraph over $\{B, C\}$; this subgraph contains at exactly one loop, which is connected to an additional cluster. Figure 13.4b shows the induced subgraph over $\{C, E\}$; this subgraph is a union of two disjoint trees. ■

We can now state the following important theorem:

Theorem 13.7

Let $\mathcal{U}$ be a max-product calibrated cluster graph for $\tilde{P}_\Phi$, and let ξ^* be a locally optimal assignment for $\mathcal{U}$. Let $\boldsymbol{Z}$ be any set of variables for which $\mathcal{U}_{\boldsymbol{Z}}$ is a collection of disjoint subgraphs each of which contains at most a single loop. Then for any assignment ξ' which is the same as ξ^* except for the assignment to the variables in $\boldsymbol{Z}$, we have that

$$\tilde{P}_\Phi(\xi') \leq \tilde{P}_\Phi(\xi^*). \tag{13.15}$$

Proof We prove the theorem under the assumption that $\mathcal{U}_{\boldsymbol{Z}}$ is a single tree, leaving the rest of the proof as an exercise (exercise 13.12). Owing to the recalibration property, we can rewrite the joint probability $\tilde{P}_\Phi$ as in equation (13.12). We can partition the terms in this expression into two groups: those that involve variables in $\boldsymbol{Z}$ and those that do not. Let $\boldsymbol{Y} = \mathcal{X} - \boldsymbol{Z}$ and $\boldsymbol{y}^*$ be the locally optimal assignment to $\boldsymbol{Y}$. We now consider the unnormalized measure obtained over $\boldsymbol{Z}$ when we restrict the distribution to the event $\boldsymbol{Y} = \boldsymbol{y}^*$ (as in definition 4.5). Since we set $\boldsymbol{Y} = \boldsymbol{y}^*$, the terms corresponding to beliefs that do not involve $\boldsymbol{Z}$ are constant, and hence they do not affect the comparison between ξ' and ξ^*.

We can now define $\tilde{P}'_{\boldsymbol{y}^*}(\boldsymbol{Z})$ to be the measure obtained by restricting equation (13.12) only to the terms in the beliefs (at both clusters and sepsets) that involve variables in $\boldsymbol{Z}$. It follows that an assignment $\boldsymbol{z}$ optimizes $\tilde{P}_\Phi(\boldsymbol{z}, \boldsymbol{y}^*)$ if and only if it optimizes $\tilde{P}'_{\boldsymbol{y}^*}$. This measure precisely corresponds to a clique tree whose structure is $\mathcal{U}_{\boldsymbol{Z}}$ and whose beliefs are the beliefs in our original calibrated cluster graph $\mathcal{U}$, but restricted to $\boldsymbol{Y} = \boldsymbol{y}^*$. Let $\mathcal{T}_{\boldsymbol{y}^*}$ represent this clique tree and its associated beliefs.

Because $\mathcal{U}$ is max-product calibrated, so is its subgraph $\mathcal{T}_{\boldsymbol{y}^*}$. Moreover, if an assignment $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ is optimal for some belief β_i, then $\boldsymbol{z}^*$ is also optimal for the restricted belief $\beta_i[\boldsymbol{Y} = \boldsymbol{y}^*]$. We therefore have a max-product calibrated clique tree $\mathcal{T}_{\boldsymbol{y}^*}$ and $\boldsymbol{z}^*$ is a locally optimal assignment for it. Because this is a clique tree, local optimality implies MAP, and so $\boldsymbol{z}^*$ must be a MAP assignment in this clique tree. As a consequence, there is no assignment $\boldsymbol{z}'$ that has a higher probability in $\tilde{P}'_{\boldsymbol{y}^*}$, proving the desired result. ■

To illustrate the power of this theorem, consider the following example:

Example 13.14

Consider the 4×4 grid network in figure 11.4, and assume that we use the pairwise cluster graph construction of figure 11.6 (shown there for a 3×3 grid). This result implies that the MAP solution found by max-product belief propagation has higher probability than any assignment obtained by changing the assignment to any of the following subsets of variables $\boldsymbol{Y}$:

- *a set of variables in any single row, such as $\boldsymbol{Y} = \{A_{1,1}, A_{1,2}, A_{1,3}, A_{1,4}\}$;*
- *a set of variables in any single column;*
- *a "comb" structure such as the variables in row 1, column 2 and column 4;*
- *a single loop, such as $\boldsymbol{Y} = \{A_{1,1}, A_{1,2}, A_{2,2}, A_{2,1}\}$;*
- *a collection of disconnected subsets of the preceding form, for example: the union of the variables in rows 1 and 3; or the loop above union with the L-structure consisting of the variables in row 4 and the variables in column 4.* ■

This result is a powerful one, inasmuch as it shows that the solution obtained from max-product belief propagation is robust against large perturbations. Thus, although

one can construct examples where max-product belief propagation obtains the wrong solutions, these solutions are strong local maxima, and therefore they often have high probability.

13.4.2 Max-Product BP with Counting Numbers ⋆

The preceding algorithm performs max-product message passing that is analogous to the sum-product message passing with the Bethe free-energy approximation. We can also construct analogues of the various generalizations of sum-product message passing, as defined in section 11.3.7. We can derive max-product variants based both on the region-graph methods, which allow us to introduce larger clusters, and based on the notion of alternative counting numbers. From an algorithmic perspective, the transformation of sum-product to max-product algorithms is straightforward: we simply replace summation with maximization. The key question is the extent to which we can provide a formal justification for these methods.

Recall that, in our discussion of sum-product algorithms, we derived the belief propagation algorithms in two different ways. The first was simply by taking the message passing algorithm on clique trees and running it on loopy cluster graphs, ignoring the presence of loops. The second derivation was obtained by a variational analysis, where the algorithm arose naturally as the fixed points of an approximate energy functional. This view was compelling both because it suggested some theoretical justification for the algorithm and, even more important, because it immediately gave rise to a variety of generalizations, obtained from different approximations to the energy functional, different methods for optimizing the objective, and more.

For the case of max-product, our discussion so far follows the first approach, viewing the message passing algorithm as a simple generalization of the max-product clique tree algorithm. Given the similarity between the sum-product and max-product algorithms presented so far, one may assume that we can analogously provide a variational justification for max-product, for example, as optimizing the same energy functional, but with max-calibration rather than sum-calibration constraints on adjacent clusters. For example, in a variational derivation of the max-product clique tree algorithm, we would replace the sum-calibration constraint of equation (11.7) with the analogous max-calibration constraint of equation (13.10). Although plausible, this analogy turns out to be incorrect. The key problem is that, whereas the sum-marginalization constraint of equation (11.7) is a simple linear equality, the constraint of equation (13.10) is not. Indeed, the max function involved in the constraint is not even smoothly differentiable, so that the framework of Lagrange multipliers cannot be applied.

However, as we now show, we can provide an optimization-based derivation and more formal justification for max-product BP with convex counting numbers. For these variants, we can even show conditions under which these algorithms are guaranteed to produce the correct MAP assignment. We begin this section by describing the basic algorithm, and proving the key optimality result: that any locally optimal assignment for convex max-product BP is guaranteed to be the MAP assignment. Then, in section 13.5, we provide an alternative view of this approach in terms of its relationship to two other classes of algorithms. This perspective will shed additional insight on the properties of this algorithm and on the cases in which it provides a useful guarantee.

Algorithm 13.3 Calibration using max-product BP in a Bethe-structured cluster graph

Procedure Generalized-MP-BP (
  Φ, // Set of factors
  $\mathbf{R}$, // Set of regions
  $\{\kappa_r\}_{r\in\mathbf{R}}$, $\{\kappa_i\}_{X_i\in\mathcal{X}}$ // Counting numbers
)

$\rho_i \leftarrow 1/\kappa_i$
$\rho_r \leftarrow 1/\kappa_r$
Initialize-CGraph
while region graph is not max-calibrated
  Select $\boldsymbol{C}_r$ and $X_i \in \boldsymbol{C}_r$
    $\delta_{i\to r}(X_i) \leftarrow \left[\left(\prod_{r'\neq r}\delta_{i\to r'}(X_i)\right)^{\rho_i}\left(\max_{\boldsymbol{C}_r - X_i}\psi_r(\boldsymbol{C}_r)\left(\prod_{X_j\in\boldsymbol{C}_r, j\neq i}\delta_{j\to r}\right)^{\rho_r}\right)\right]^{-\frac{1}{\rho_i+\rho_r}}$
  for each region $r \in \mathbf{R}\cup\{1,\ldots,n\}$
    $\beta_r(\boldsymbol{C}_r) \leftarrow \left(\psi_r(\boldsymbol{C}_r)\prod_{X_i\in\boldsymbol{C}_r}\delta_{i\to r}(X_i)\right)^{\rho_r}$
return $\{\beta_r\}_{r\in\mathbf{R}}$

13.4.2.1 Max-Product with Counting Numbers

We begin with a reminder of the notion of belief propagation with counting numbers. For concreteness, we also provide the max-product variant of a message passing algorithm for this case, although (as we mentioned) the max-product variant can be obtained from the sum-product algorithm using a simple syntactic substitution.

counting numbers

In section 11.3.7, we defined a set of sum-product message passing algorithms; these algorithms were defined in terms of a set of *counting numbers* that specify the extent to which entropy terms for different subsets of variables are counted in the entropy approximation used in the energy functional. For a given set of counting numbers, one can derive a message passing algorithm by using the fixed point equations obtained by differentiating the Lagrangian for the energy functional, with its sum-product calibration constraints. The standard belief propagation algorithm is obtained from the Bethe energy approximation; other sets of counting numbers give rise to other message passing algorithms.

Bethe cluster graphs

As we discussed, one can take these sum-product message passing algorithms (for example, those in exercise 11.17 and exercise 11.19) and convert them to produce a max-product variant by simply replacing each summation operation as maximization. For concreteness, in algorithm 13.3, we repeat the algorithm of exercise 11.17, instantiated to the max-product setting. Recall that this algorithm applies only to *Bethe cluster graphs*, that is, graphs that have two levels of regions: "large" regions r containing multiple variables with counting numbers κ_r, and singleton regions containing individual variables X_i with counting numbers κ_i; all factors in Φ are assigned only to large regions, so that $\psi_i = 1$ for all i.

reparameterization

A critical observation is that, **like the sum-product algorithms, and like the max-product clique tree algorithm (see theorem 13.5), these new message passing algorithms are a *reparameterization* of the original distribution.** In other words, their fixed points are a different representation of the same distribution, in terms of a set of max-calibrated beliefs. This property, which is stated for sum-product in theorem 11.6, asserts that, at fixed points of the message passing algorithm, we have that:

$$\tilde{P}_\Phi(\mathcal{X}) = \prod_r (\beta_r)^{\kappa_r}. \tag{13.16}$$

The proof of this equality (see exercise 11.16 and exercise 11.19) is a consequence only of the way in which we define region beliefs in terms of the messages. Therefore, the reparameterization property applies equally to fixed points of the max-product algorithms. It is this property that will be critical in our derivation.

13.4.2.2 Optimality Guarantee

As in the case of standard max-product belief propagation algorithms, given a set of max-product calibrated beliefs that reparameterize the distribution, we now search for an assignment that is locally optimal for this set of beliefs. However, as we now show, under certain assumptions, such an assignment is guaranteed to be the MAP assignment.

Although more general variants of this theorem exist, we focus on the case of a Bethe-structured region graph, as described. Here, we also assume that our large regions in $\mathbf{R}$ have counting number 1. We assume also that factors in the network are assigned only to large regions, so that $\psi_i = 1$ for all i. Finally, in a property that is critical to the upcoming derivation, we assume that the counting numbers κ_r are convex, as defined in definition 11.4. Recall that a vector of counting numbers κ_r is *convex* if there exist *nonnegative* numbers ν_r, ν_i, and $\nu_{r,i}$ such that:

convex counting numbers

$$\begin{aligned} \kappa_r &= \nu_r + \textstyle\sum_{i\,:\,X_i \in \boldsymbol{C}_r} \nu_{r,i} \quad \text{for all } r \\ \kappa_i &= \nu_i - \textstyle\sum_{r\,:\,X_i \in \boldsymbol{C}_r} \nu_{r,i} \quad \text{for all } i. \end{aligned}$$

This is the same assumption used to guarantee that the region-graph energy functional in equation (11.27) is a concave function. Although here we have no energy functional, the purpose of this assumption is similar: As we will see, it allows us to redistribute the terms in the reparameterization of the probability distribution, so as to guarantee that all terms have a positive coefficient.

From these assumptions, we can now prove the following theorem:

Theorem 13.8 *Let P_Φ be a distribution, and consider a Bethe-structured region graph with large regions and singleton regions, where the counting numbers are convex. Assume that we have a set of max-calibrated beliefs $\beta_r(\boldsymbol{C}_r)$ and $\beta_i(X_i)$ such that equation (13.16) holds. If there exists an assignment ξ^* that is locally optimal relative to each of the beliefs β_r, then ξ^* is the optimal MAP assignment.*

Proof Applying equation (13.16) to our Bethe-structured graph, we have that:

$$\tilde{P}_\Phi(\mathcal{X}) = \prod_{r \in \mathbf{R}} \beta_r \prod_i \beta_i^{\kappa_i}.$$

Owing to the convexity of the counting numbers, we can rewrite the right-hand side as:

$$\prod_r (\beta_r)^{\nu_r} \prod_i (\beta_i)^{\nu_i} \prod_{i,r \,:\, X_i \in \boldsymbol{C}_r} \left(\frac{\beta_r}{\beta_i} \right)^{\nu_{r,i}}.$$

Owing to the nonnegativity of the coefficients ν, we have that:

$$\begin{aligned} \max_\xi \tilde{P}_\Phi(\xi) &= \max_\xi \prod_r (\beta_r(\boldsymbol{c}_r))^{\nu_r} \prod_i (\beta_i(x_i))^{\nu_i} \prod_{i,r \,:\, X_i \in \boldsymbol{C}_r} \left(\frac{\beta_r}{\beta_i}(\boldsymbol{c}_r) \right)^{\nu_{r,i}} \\ &\leq \prod_r (\max_{\boldsymbol{c}_r} \beta_r(\boldsymbol{c}_r))^{\nu_r} \prod_i (\max_{x_i} \beta_i(x_i))^{\nu_i} \prod_{i,r \,:\, X_i \in \boldsymbol{C}_r} \left(\max_{\boldsymbol{c}_r} \frac{\beta_r}{\beta_i}(\boldsymbol{c}_r) \right)^{\nu_{r,i}}. \end{aligned}$$

We have now reduced this expression to a product of terms, each raised to the power of a positive exponent. Some of these terms are factors in the max-product calibrated network, and others are ratios of factors and their max-product marginal over an individual variable. The proof now is exactly the same as the proof of theorem 13.6. Let ξ^* be an assignment that satisfies the local optimality property. By assumption, it optimizes every one of the region beliefs. Because the ratios involve a factor and its max-marginal, the conditions of lemma 13.1 hold for each of the ratios in this expression. Thus, ξ^* optimizes each one of the terms in this product, and therefore it optimizes the product as a whole. It therefore optimizes $\tilde{P}_\Phi(\xi)$, and must therefore be the MAP assignment. ■

We can also derive the following useful corollary, which allows us, in certain cases, to characterize parts of the MAP solution even if the local optimality property does not hold:

Corollary 13.4

Under the setting of theorem 13.8, if a variable X_i takes a particular value x_i^* in all locally optimal assignments ξ^* then $x_i^{map} = x_i^*$ in the MAP assignment. More generally, if there is some set S_i such that, in any locally optimal assignment ξ^* we have that $x_i^* \in S_i$, then $x_i^{map} \in S_i$.

At first glance, the application of this result seems deceptively easy. After all, in order to be locally optimal, an assignment must assign to X_i one of the values that maximizes its individual node marginal. Thus, it appears that we can easily extract, for each X_i, some set S_i (perhaps an overestimate) to which corollary 13.4 applies. Unfortunately, when we use this procedure, we cannot guarantee that x_i^{map} is actually in the set S_i. The corollary applies only if there exists a locally optimal assignment to the entire set of beliefs. If no such assignment exists, the set of locally maximizing values in X_i's node belief may have no relation to the true MAP assignment.

13.4.3 Discussion

In this section, we have shown that max-product message passing algorithms, if they converge, provide a max-calibrated reparameterization of the distribution $\tilde{P}_\Phi$. This reparameterization essentially converts the global optimization problem of finding a single joint MAP assignment to a local optimization problem: finding a set of locally optimal assignments to the individual cliques that are consistent with each other. Importantly, we can show that this locally consistent assignment, if it exists, satisfies strong optimality properties: In the case of the standard (Bethe-approximation) reparameterization,

the joint assignment satisfies strong local optimality; in the case of reparameterizations based on convex counting numbers, it is actually guaranteed to be the MAP assignment.

Although these guarantees are very satisfying, their usefulness relies on several important questions that we have not yet addressed. The first two relate to the max-product calibrated reparameterization of the distribution: does one exist, and can we find it? First, for a given set of counting numbers, does there always exist a max-calibrated reparameterization of $\tilde{P}_\Phi$ in terms of these counting numbers? Somewhat surprisingly, as we show in section 13.5.3, the answer to that question is yes for convex counting numbers; it turns out to hold also for the Bethe counting numbers, although we do not show this result. Second, we must ask whether we can always find such a reparameterization. We know that if the max-product message passing algorithm converges, it must converge to such a fixed point. But unfortunately, there is no guarantee that the algorithm will converge. In practice, standard max-product message passing often does not converge. For certain specific choices of convex counting numbers (see, for example, box 13.A), one can design algorithms that are guaranteed to be convergent.

However, even if we find an appropriate reparameterization, we are still left with the problem of extracting a joint assignment that satisfies the local optimality property. Indeed, such as assignment may not even exist. In section 13.5.3.3, we present a necessary condition for the existence of such an assignment.

It is currently not known how the choice of counting numbers affects either of these two issues: our ability to find effectively a max-product calibrated reparameterization, and our ability to use the result to find a locally consistent joint assignment. Empirically, preliminary results suggest that nonconvex counting numbers (such as those obtained from the Bethe approximation) converge less often than the the convex variants, but converge more quickly when they do converge. The different convex variants converge at different rates, but tend to converge to fixed points that have a similar set of ambiguities in the beliefs. Moreover, in cases where convex max-product BP converges whereas standard max-product does not, the resulting beliefs often contain many ambiguities (beliefs with equal values), making it difficult to determine whether the local optimality property holds, and to identify such an assignment if it exists.

---

tree-reweighted belief propagation

LP relaxation

Box 13.A — Concept: Tree-Reweighted Belief Propagation. *One algorithm that is worthy of special mention, both because of historical precedence and because of its popularity, is the* tree-reweighted belief propagation *algorithm (often known as* TRW*). This algorithm was the first message passing algorithm to use convex counting numbers; it was also the context in which message passing algorithms were first shown to be related to the* linear program relaxation *of the MAP optimization problem that we discuss in section 13.5. This algorithm, developed in the context of a pairwise Markov network, utilizes the same approach as in the TRW variant of sum-product message passing: It defines a probability distribution over trees $\mathcal{T}$ in the network, so that each edge in the pairwise network appears in at least one tree, and it then defines the counting numbers to be the edge and negative node appearance probabilities, as defined in equation (11.26). Note that, unlike the algorithms of section 13.4.2.1, here the factors do not have a counting number of* 1*, so that the algorithms we presented there require some modification. Briefly, the max-product TRW*

algorithm uses the following update rule:

$$\delta_{i\to j} = \max_{x_i} \left[\left(\psi_i(x_i) \prod_{k \in \mathrm{Nb}_i} \delta_{k\to i}(x_i) \right)^{\frac{\kappa_{i,j}}{\kappa_i}} \frac{1}{\delta_{j\to i}(x_i)} \psi_{i,j}(x_i, x_j) \right]. \tag{13.17}$$

One particular variant of the TRW algorithm, called TRW-S, provides some particularly satisfying guarantees. Assume that we order the nodes in the network in some fixed ordering $X_1, \ldots, X_n$, and consider a set of trees each of which is a subsequence of this ordering, that is, of the form $X_{i_1}, \ldots, X_{i_k}$ for $i_1 < \ldots i_k$. We now pass messages in the network by repeating two phases, where in one phase we pass messages from X_1 towards X_n, and in the other from X_n towards X_1. With this message passing scheme, it is possible to guarantee that the algorithm continuously increases the dual objective, and hence it is convergent.

13.5 MAP as a Linear Optimization Problem ⋆

A very important and useful insight on the MAP problem is derived from viewing it directly as an optimization problem. This perspective allows us to draw upon the vast literature on optimization algorithms and apply some of these ideas to the specific case of MAP inference. Somewhat surprisingly, some of the algorithms that we describe elsewhere in this chapter turn out to be related to the optimization perspective; the insights obtained from understanding the connections can provide the basis for a theoretical analysis of these methods, and they can suggest improvements.

For the purposes of this section, we assume that the distribution specified in the MRF is positive, so that all of the entries in all of the factors are positive. This assumption turns out to be critical for some of the derivations in this section, and facilitates many others.

13.5.1 The Integer Program Formulation

integer linear program

The basic MAP problem can be viewed as an *integer linear program* — an optimization problem (see appendix A.4.1) over a set of integer valued variables, where both the objective and the constraints are linear. To define a linear optimization problem, we must first turn all of our products into summations. This transformation gives rise to the following *max-sum* form:

max-sum

$$\arg\max_{\xi} \log \tilde{P}_\Phi(\xi) = \arg\max_{\xi} \sum_{r \in \mathbf{R}} \log(\phi_r(\boldsymbol{c}_r)), \tag{13.18}$$

where $\Phi = \{\phi_r \ : \ r \in \mathbf{R}\}$, and $\boldsymbol{C}_r$ is the scope of ϕ_r.

For $r \in \mathbf{R}$, we define $n_r = |Val(\boldsymbol{C}_r)|$. For any joint assignment ξ, if $\xi\langle \boldsymbol{C}_r \rangle = \boldsymbol{c}_r^j$, the factor $\log(\phi_r)$ makes a contribution to the objective of $\log(\phi_r(\boldsymbol{c}_r^j))$, a quantity that we denote as η_r^j.

optimization variables

We introduce *optimization variables* $q(\boldsymbol{x}_r^j)$, where $r \in \mathbf{R}$ enumerates the different factors, and $j = 1, \ldots, n_r$ enumerates the different possible assignments to the variables $\boldsymbol{C}_r$ that comprise the factor $\boldsymbol{C}_r$. These variables take binary values, so that $q(\boldsymbol{x}_r^j) = 1$ if and only if $\boldsymbol{C}_r = \boldsymbol{c}_r^j$, and 0 otherwise. It is important to distinguish the optimization variables from the random

variables in our original graphical model; here we have an optimization variable $q(\boldsymbol{x}_r^j)$ for each joint assignment $\boldsymbol{c}_r^j$ to the model variables $\boldsymbol{C}_r$.

Let $\boldsymbol{q}$ denote a vector of the optimization variables $\{q(\boldsymbol{x}_r^j) : r \in \mathbf{R};\ j = 1, \ldots, n_r\}$, and $\boldsymbol{\eta}$ denote a vector of the coefficient η_r^j sorted in the same order. Both of these are vectors of dimension $N = \sum_{k=1}^{K} n_r$. With this interpretation, the MAP objective can be rewritten as:

$$\textbf{maximize}_{\boldsymbol{q}} \sum_{r \in \mathbf{R}} \sum_{j=1}^{n_r} \eta_r^j q(\boldsymbol{x}_r^j), \tag{13.19}$$

or, in shorthand, $\textbf{maximize}_{\boldsymbol{q}} \boldsymbol{\eta}^T \boldsymbol{q}$.

Example 13.15

Assume that we have a pairwise MRF shaped like a triangle A—B—C—A, so that we have three factors over pairs of connected random variables: $\phi_1(A, B), \phi_2(B, C), \phi_3(A, C)$. Assume that A, B are binary-valued, whereas C takes three values. Here, we would have the optimization variables $q(\boldsymbol{x}_1^1), \ldots, q(\boldsymbol{x}_1^4), q(\boldsymbol{x}_2^1), \ldots, q(\boldsymbol{x}_2^6), q(\boldsymbol{x}_3^1), \ldots, q(\boldsymbol{x}_3^6)$. We assume that the values of the variables are enumerated lexicographically, so that $q(\boldsymbol{x}_3^4)$, for example, corresponds to a^2, c^1. ■

We can view our MAP inference problem as optimizing this linear objective over the space of assignments to $\boldsymbol{q} \in \{0, 1\}^N$ that correspond to legal assignments to $\mathcal{X}$. What constraints on $\boldsymbol{q}$ do we need to impose in order to guarantee that it corresponds to some assignment to $\mathcal{X}$? Most obviously, we need to ensure that, in each factor, only a single assignment is selected. Thus, in our example, we cannot have both $q(\boldsymbol{x}_1^1) = 1$ and $q(\boldsymbol{x}_1^2) = 1$. Slightly subtler are the cross-factor consistency constraints: If two factors share a variable, we need to ensure that the assignment to this variable according to $\boldsymbol{q}$ is consistent in those two factors. In our example, for instance, if we have that $q(\boldsymbol{x}_1^1) = 1$, so that $B = b^1$, we would need to have $q(\boldsymbol{x}_2^1) = 1$, $q(\boldsymbol{x}_2^2) = 1$, or $q(\boldsymbol{x}_2^3) = 1$.

There are several ways of encoding these consistency constraints. First, we require that we restrict attention to integer solutions:

$$q(\boldsymbol{x}_r^j) \in \{0, 1\} \qquad \text{For all } r \in \mathbf{R};\ j \in \{1, \ldots, n_r\}. \tag{13.20}$$

We can now utilize two linear equalities to enforce the consistency of these integer solutions. The first constraint enforces the mutual exclusivity within a factor:

$$\sum_{j=1}^{n_r} q(\boldsymbol{x}_r^j) = 1 \qquad \text{For all } r \in \mathbf{R}. \tag{13.21}$$

The second constraint implies that factors in our MRF agree on the variables in the intersection of their scopes:

$$\sum_{j\ :\ \boldsymbol{c}_r^j \sim \boldsymbol{s}_{r,r'}} q(\boldsymbol{x}_r^j) = \sum_{l\ :\ \boldsymbol{c}_{r'}^l \sim \boldsymbol{s}_{r,r'}} q(\boldsymbol{x}_{r'}^l) \tag{13.22}$$

For all $r, r' \in \mathbf{R}$ and all $\boldsymbol{s}_{r,r'} \in Val(\boldsymbol{C}_r \cap \boldsymbol{C}_{r'})$.

Note that this constraint is vacuous for pairs of clusters whose intersection is empty, since there are no assignments $\boldsymbol{s}_{r,r'} \in Val(\boldsymbol{C}_r \cap \boldsymbol{C}_{r'})$.

Example 13.16

Returning to example 13.15, the mutual exclusivity constraints for ϕ_1 would assert that $\sum_{j=1}^{4} q(\boldsymbol{x}_1^j) = 1$. Altogether, we would have three such constraints — one for each factor. The consistency constraints associated with $\phi_1(A, B)$ and $\phi_2(B, C)$ assert that:

$$\begin{aligned} q(\boldsymbol{x}_1^1) + q(\boldsymbol{x}_1^3) &= q(\boldsymbol{x}_2^1) + q(\boldsymbol{x}_2^2) + q(\boldsymbol{x}_2^3) \\ q(\boldsymbol{x}_1^2) + q(\boldsymbol{x}_1^4) &= q(\boldsymbol{x}_2^4) + q(\boldsymbol{x}_2^5) + q(\boldsymbol{x}_2^6), \end{aligned}$$

where the first constraint ensures consistency when $B = b^1$ and the second when $B = b^2$. Overall, we would have three such constraints for $\phi_2(B, C), \phi_3(A, C)$, corresponding to the three values of C, and two constraints for $\phi_1(A, B), \phi_3(A, C)$, corresponding to the two values of A.

Together, these constraints imply that there is a one-to-one mapping between possible assignments to the $q(\boldsymbol{x}_r^j)$ optimization variables and legal assignments to A, B, C. ■

In general, equation (13.20), equation (13.21), and equation (13.22) together imply that the assignment $q(\boldsymbol{x}_r^j)$'s correspond to a single legal assignment:

Proposition 13.4

Any assignment to the optimization variables $\boldsymbol{q}$ that satisfies equation (13.20), equation (13.21), and equation (13.22) corresponds to a single legal assignment to $X_1, \ldots, X_n$.

The proof is left as an exercise (see exercise 13.13).

Thus, we have now reformulated the MAP task as an integer linear program, where we optimize the linear objective of equation (13.19) subject to the constraints equation (13.20), equation (13.21), and equation (13.22). We note that the problem of solving integer linear programs is itself $\mathcal{NP}$-hard, so that (not surprisingly) we have not avoided the basic hardness of the MAP problem. However, there are several techniques that have been developed for this class of problems, which can be usefully applied to integer programs arising from MAP problems. One of the most useful is described in the next section.

13.5.2 Linear Programming Relaxation

LP relaxation

linear program

One of the methods most often used for tackling integer linear programs is the method of *linear program relaxation*. In this approach, we turn a discrete, combinatorial optimization problem into a continuous problem. This problem is a *linear program* (LP), which can be solved in polynomial time, and for which a range of very efficient algorithms exist. One can then use the solutions to this LP to obtain approximate solutions to the MAP problem. To perform this relaxation, we substitute the constraint equation (13.20) with a *relaxed constraint*:

$$q(\boldsymbol{x}_r^j) \geq 0 \quad \text{For all } r \in \mathbf{R},\ j \in \{1, \ldots, n_r\}. \tag{13.23}$$

This constraint and equation (13.21) together imply that each $q(\boldsymbol{x}_j^r) \in [0, 1]$; thus, we have relaxed the combinatorial constraint into a continuous one. This relaxation gives rise to the following *linear program* (LP):

linear program

MAP-LP:

Find $\{q(\boldsymbol{x}_r^j) \ : \ r \in \mathbf{R}; j = 1, \ldots, n_r\}$

maximizing $\boldsymbol{\eta}^\top \boldsymbol{q}$

subject to

$$\sum_{j=1}^{n_r} q(\boldsymbol{x}_r^j) = 1 \qquad r \in \mathbf{R}$$

$$\sum_{j \ : \ \boldsymbol{c}_r^j \sim \boldsymbol{s}_{r,r'}} q(\boldsymbol{x}_r^j) = \sum_{l \ : \ \boldsymbol{c}_{r'}^l \sim \boldsymbol{s}_{r,r'}} q(\boldsymbol{x}_{r'}^l) \qquad r, r' \in \mathbf{R}, \ \boldsymbol{s}_{r,r'} \in Val(\boldsymbol{C}_r \cap \boldsymbol{C}_{r'})$$

$$\boldsymbol{q} \geq \mathbf{0}$$

This linear program is a relaxation of our original integer program, since every assignment to $\boldsymbol{q}$ that satisfies the constraints of the integer problem also satisfies the constraints of the linear program, but not the other way around. Thus, the optimal value of the objective of the relaxed version will be no less than the value of the (same) objective in the exact version, and it can be greater when the optimal value is achieved at an assignment to $\boldsymbol{q}$ that does not correspond to a legal assignment ξ.

pseudo-marginals

local consistency polytope

A closer examination shows that the space of assignments to $\boldsymbol{q}$ that satisfies the constraints of MAP-LP corresponds exactly to the locally consistent *pseudo-marginals* for our cluster graph $\mathcal{U}$, which comprise the *local consistency polytope* $Local[\mathcal{U}]$, defined in equation (11.16). To see this equivalence, we note that equation (13.23) and equation (13.21) imply that any assignment to $\boldsymbol{q}$ defines a set of locally normalized distributions over the clusters in the cluster graph — nonnegative factors that sum to 1; by equation (13.22), these factors must be sum-calibrated. Thus, there is a one-to-one mapping between consistent pseudo-marginals and possible solutions to the LP.

We can use this observation to answer the following important question: Given a non-integer solution to the relaxed LP, how can we derive a concrete assignment? One obvious approach is a greedy assignment process, which assigns values to the variables X_i one at a time. For each variable, and for each possible assignment x_i, it considers the set of reduced pseudo-marginals that would result by setting $X_i = x_i$. We can now compute the energy term (or, equivalently, the LP objective) for each such assignment, and select the value x_i that gives the maximum value. We then permanently reduce each of the pseudo-marginals with the assignment $X_i = x_i$, and continue. We note that, at the point when we assign X_i, some of the variables have already been assigned, whereas others are still undetermined. At the end of the process, all of the variables have been assigned a specific value, and we have a single joint assignment.

marginal polytope

To understand the result obtained by this algorithm, recall that $Local[\mathcal{U}]$ is a superset of the *marginal polytope* $Marg[\mathcal{U}]$ — the space of legal distributions that factorize over $\mathcal{U}$ (see equation (11.15)). Because our objective in equation (13.19) is linear, it has the same optimum over the marginal polytope as over the original space of $\{0, 1\}$ solutions: The value of the objective at a point corresponding to a distribution $P(\mathcal{X})$ is the expectation of its value at the assignments ξ that receive positive probability in P; therefore, one cannot achieve a higher value of the objective with respect to P than with respect to the highest-value assignment ξ. Thus, if we could perform our optimization over the continuous space $Marg[\mathcal{U}]$, we would find the optimal solution to our MAP objective. However, as we have already discussed, the marginal

polytope is a complex object, which can be specified only using exponentially many constraints. Thus, we cannot feasibly perform this optimization.

By contrast, the optimization problem obtained by this relaxed version has a linear objective with linear constraints, and both involve a number of terms which is linear in the size of the cluster graph. Thus, this linear program admits a range of efficient solutions, including ones with polynomial time guarantees. We can thus apply off-the-shelf methods for solving such problems. Of course, the result is often fractional, in which case it is clearly not an optimal solution to the MAP problem.

The LP formulation has advantages and disadvantages. **By formulating our problem as a linear program, we obtain a very flexible framework for solving it; in particular, we can easily incorporate additional constraints into the LP, which reduce the space of possible assignments to q, eliminating some solutions that do not correspond to actual distributions over $\mathcal{X}$.** (See section 13.9 for some references.) On the other hand, as we discussed in example 11.4, the optimization problems defined over this space of constraints are very large, making standard optimization methods very expensive. Of course, the LP has special structure: For example, when viewed as a matrix, the equality constraints in this LP all have a particular block structure that corresponds to the structure of adjacent clusters; moreover, when the MRF is not densely connected, the constraint matrix is also sparse. However, standard LP solvers may not be ideally suited for exploiting this special structure. Thus, empirical evidence suggests that the more specialized solution methods for the MAP problems are often more effective than using off-the-shelf LP solvers. As we now discuss, the convex message passing algorithms described in section 13.4.2 can be viewed as specialized solution methods to the *dual* of this LP. More recent work explicitly aims to solve this dual using general-purpose optimization techniques that do take advantage of the structure; see section 13.9 for some references.

13.5.3 Low-Temperature Limits

In this section, we show how we can use a limit process to understand the connection between the relaxed MAP-LP and both sum-product and max-product algorithms with convex counting numbers. As we show, this connection provides significant new insight on all three algorithms.

13.5.3.1 LP as Sum-Product Limit

More precisely, recall that the energy functional is defined as:

$$F[P_\Phi, Q] = \sum_{\phi_r \in \Phi} \mathbf{E}_{\boldsymbol{C}_r \sim Q}[\log \phi_r(\boldsymbol{C}_r)] + \tilde{\boldsymbol{H}}_Q(\mathcal{X}),$$

where $\tilde{\boldsymbol{H}}_Q(\mathcal{X})$ is some (exact or approximate) version of the entropy of Q. Consider the first term in this expression, also called the energy term. Let $q(\boldsymbol{x}_r^j)$ denote the cluster marginal $\beta_r(\boldsymbol{c}_r^j)$. Then we can rewrite the energy term as:

$$\sum_{r \in \mathbf{R}} \sum_{j=1}^{n_r} q(\boldsymbol{x}_r^j) \log(\phi_r(\boldsymbol{c}_r^j),$$

which is precisely the objective in our LP relaxation of the MAP problem. Thus, the energy functional is simply a sum of two terms: the LP relaxation objective, and an entropy term. In

the energy functional, both of these terms receive equal weight. Now, however, consider an alternative objective, called the *temperature-weighted energy function*. This objective is defined in terms of a *temperature* parameter $T > 0$:

temperature-weighted energy function

temperature

$$\tilde{F}^{(T)}[P_\Phi, Q] = \sum_{\phi_r \in \Phi} \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}[\log \phi_r(\boldsymbol{C}_r)] + T\tilde{\boldsymbol{H}}_Q(\mathcal{X}). \tag{13.24}$$

As usual in our derivations, we consider the task of maximizing this objective subject to the sum-marginalization constraints, that is, that $Q \in \mathit{Local}[\mathcal{U}]$.

The temperature-weighted energy functional reweights the importance of the two terms in the objective. Since $T \longrightarrow 0$, we will place a greater emphasis on the linear energy term (the first term), which is precisely the objective of the relaxed LP. Thus, since $T \longrightarrow 0$, the objective $\tilde{F}^{(T)}[P_\Phi, Q]$ tends to the LP objective. Can we then infer that the fixed points of the objective (say, those obtained from a message passing algorithm) are necessarily optima of the LP? The answer to this question is positive for concave versions of the entropy, and negative otherwise.

In particular, assume that $\tilde{\boldsymbol{H}}_Q(\mathcal{X})$ is a weighted entropy $\tilde{\boldsymbol{H}}^{\boldsymbol{\kappa}}_{\boldsymbol{Q}}(\mathcal{X})$, such that $\boldsymbol{\kappa}$ is a convex set of counting numbers, as in equation (11.20). From the assumption on convexity of the counting numbers and the positivity of the distribution, it follows that the function $\tilde{F}^{(T)}[P_\Phi, Q]$ is strongly convex in the distribution Q. The space $\mathit{Local}[\mathcal{U}]$ is a convex space. Thus, there is a unique global minimum $Q^{*(T)}$ for every T, and that optimum changes continuously in T. Standard results now imply that the limit of $Q^{*(T)}$ is optimal for the limiting problem, which is precisely the LP.

On the other hand, this result does not hold for nonconvex entropies, such as the one obtained by the Bethe approximation, where the objective can have several distinct optima. In this case, there are examples where a sequence of optima obtained for different values of T converges to a point that is not a solution to the LP. Thus, for the remainder of this section, we assume that $\tilde{\boldsymbol{H}}_Q(\mathcal{X})$ is derived from a convex set of counting numbers.

13.5.3.2 Max-Product as Sum-Product Limit

What do we gain from this perspective? It does not appear practical to use this characterization as a constructive solution method. For one thing, we do not want to solve multiple optimization problems, for different values of T. For another, the optimization problem becomes close to degenerate as T grows small, making the problem hard to solve. However, if we consider the dual problem to each of the optimization problems of this sequence, we can analytically characterize the limit of these duals. Surprisingly, this limit turns out to be a fixed point of the max-product belief propagation algorithm.

We first note that the temperature-weighted energy functional is virtually identical in its form to the original functional. Indeed, we can formalize this intuition if we divide the objective through by T; since $T > 0$, this step does not change the optima. The resulting objective has the form:

$$\begin{aligned} \frac{1}{T}\sum_{\phi_r \in \Phi} \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}[\log \phi_r(\boldsymbol{C}_r)] + \boldsymbol{H}_Q(\mathcal{X}) &= \sum_{\phi_r \in \Phi} \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}\left[\frac{1}{T}(\log \phi_r(\boldsymbol{C}_r))\right] + \tilde{\boldsymbol{H}}_Q(\mathcal{X}) \\ &= \sum_{\phi_r \in \Phi} \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}\left[\log \phi_r^{1/T}\right] + \tilde{\boldsymbol{H}}_Q(\mathcal{X}). \end{aligned} \tag{13.25}$$

This objective has precisely the same form as the standard approximate energy functional, but for a different set of factors: the original factors, raised to the power of $1/T$. This set of factors defines a new unnormalized density:

$$\tilde{P}_\Phi^{(T)}(\mathcal{X}) = (\tilde{P}_\Phi(\mathcal{X}))^{1/T}.$$

Because our entropy is concave, and using our assumption that the distribution is positive, the approximate free energy $\tilde{F}[P_\Phi, Q]$ is strictly convex and hence has a unique global minimum $Q^{(T)}$ for each temperature T. We can now consider the Lagrangian dual of this new objective, and characterize this unique optimum via its dual parameterization $Q^{(T)}$. In particular, as we have previously shown, $Q^{(T)}$ is a reparameterization of the distribution $\tilde{P}_\Phi^{(T)}(\mathcal{X})$:

$$\tilde{P}_\Phi^{(T)} = \prod_{r \in \mathbf{R}} \beta_r^{(T)} \prod_i (\beta_i^{(T)})^{\kappa_i} = \prod_{r \in \mathbf{R}^+} (\beta_r^{(T)})^{\kappa_i}, \tag{13.26}$$

where, for simplicity of notation, we define $\mathbf{R}^+ = \mathbf{R} \cup \mathcal{X}$ and $\kappa_r = 1$ for $r \in \mathbf{R}$.

Our goal is now to understand what happens to $Q^{(T)}$ as we take $T \longrightarrow 0$. We first reformulate these beliefs by defining, for every region $r \in \mathbf{R}^+$:

$$\bar{\beta}_r^{(T)} = \max_{\boldsymbol{x}'_r} \beta_r^{(T)}(\boldsymbol{x}'_r) \tag{13.27}$$

$$\tilde{\beta}_r^{(T)}(\boldsymbol{c}_r) = \left(\frac{\beta_r^{(T)}(\boldsymbol{c}_r)}{\bar{\beta}_r^{(T)}} \right)^T. \tag{13.28}$$

The entries in the new beliefs $\tilde{\boldsymbol{\beta}}^{(T)} = \{\tilde{\beta}_r^{(T)}(\boldsymbol{C}_r)\}$ take values between 0 and 1, with the maximal entry in each belief always having the value 1.

We now define the limiting value of these beliefs:

$$\tilde{\beta}_r^{(0)}(\boldsymbol{c}_r) = \lim_{T \longrightarrow 0} \tilde{\beta}_r^{(T)}(\boldsymbol{c}_r). \tag{13.29}$$

Because the optimum changes continuously in T, and because the beliefs take values in a convex space (all are in the range $[0, 1]$), the limit is well defined. Our goal is to show that the limit beliefs $\tilde{\boldsymbol{\beta}}^{(0)}$ are a fixed point of the max-product belief propagation algorithm for the model $\tilde{P}_\Phi$. We show this result in two parts. We first show that the limit is max-calibrated, and then that it provides a reparameterization of our distribution $\tilde{P}_\Phi$.

Proposition 13.5 *The limiting beliefs $\tilde{\boldsymbol{\beta}}^{(0)}$ are max-calibrated.*

Proof We wish to show that for any region r, any $X_i \in \boldsymbol{C}_r$, and any $x_i \in Val(X_i)$, we have:

$$\max_{\boldsymbol{c}_r \sim x_i} \tilde{\beta}_r^{(0)}(\boldsymbol{c}_r) = \tilde{\beta}_i^{(0)}(x_i). \tag{13.30}$$

Consider the left-hand side of this equality.

$$\begin{aligned}
\max_{\boldsymbol{c}_r \sim x_i} \tilde{\beta}_r^{(0)}(\boldsymbol{c}_r) &= \max_{\boldsymbol{c}_r \sim x_i} \left[\lim_{T \longrightarrow 0} \tilde{\beta}_r^{(T)}(\boldsymbol{c}_r) \right] \\
(i) &= \lim_{T \longrightarrow 0} \left[\sum_{\boldsymbol{c}_r \sim x_i} (\tilde{\beta}_r^{(T)}(\boldsymbol{c}_r))^{1/T} \right]^T \\
&= \lim_{T \longrightarrow 0} \left[\sum_{\boldsymbol{c}_r \sim x_i} \left(\frac{\beta_r^{(T)}(\boldsymbol{c}_r)}{\bar{\beta}_r^{(T)}} \right) \right]^T \\
&= \lim_{T \longrightarrow 0} \left[\frac{1}{\bar{\beta}_r^{(T)}} \left(\sum_{\boldsymbol{c}_r \sim x_i} \beta_r^{(T)}(\boldsymbol{c}_r) \right) \right]^T \\
(ii) &= \lim_{T \longrightarrow 0} \left[\frac{1}{\bar{\beta}_r^{(T)}} \beta_i^{(T)}(x_i) \right]^T \\
(iii) &= \lim_{T \longrightarrow 0} \left[\frac{\bar{\beta}_i^{(T)}}{\bar{\beta}_r^{(T)}} \frac{\beta_i^{(T)}(x_i)}{\bar{\beta}_i^{(T)}} \right]^T \\
(iv) &= \lim_{T \longrightarrow 0} \left[\frac{\bar{\beta}_i^{(T)}}{\bar{\beta}_r^{(T)}} (\tilde{\beta}_i^{(T)}(x_i))^{1/T} \right]^T \\
&= \lim_{T \longrightarrow 0} \left[\left(\frac{\bar{\beta}_i^{(T)}}{\bar{\beta}_r^{(T)}} \right)^T \tilde{\beta}_i^{(T)}(x_i) \right] \\
(v) &= \lim_{T \longrightarrow 0} \tilde{\beta}_i^{(T)}(x_i)] \quad = \tilde{\beta}_i^{(0)}(x_i),
\end{aligned}$$

as required. In this derivation, the step marked (i) is a general relationship between maximization and summation; see lemma 13.2. The step marked (ii) is a consequence of the sum-marginalization property of the region beliefs $\beta_r^{(T)}(\boldsymbol{C}_r)$ relative to the individual node belief. The step marked (iii) is simply multiplying and dividing by the same expression. The step marked (iv) is derived directly by substituting the definition of $\tilde{\beta}_i^{(T)}(x_i)$. The step marked (v) is a consequence of the fact that, because of sum-marginalization, $\bar{\beta}_i^{(T)}/\bar{\beta}_r^{(T)}$ (for $X_i \in \boldsymbol{C}_r$) is bounded in the range $[1, |Val(\boldsymbol{C}_r - \{X_i\})|]$ for any $T > 0$, and therefore its limit, since $T \longrightarrow 0$ is 1.

It remains to prove the following lemma:

Lemma 13.2

For $i = 1, \ldots, k$, let $a_i(T)$ be a continuous function of T for $T > 0$. Then

$$\max_i \lim_{T \longrightarrow 0} a_i(T) = \lim_{T \longrightarrow 0} \left(\sum_i (a_i(T))^{1/T} \right)^T. \tag{13.31}$$

Proof Because the functions are continuous, we have that, for some T_0, there exists some j such that, for any $T < T_0$, $a_j(T) \geq a_i(T)$ for all $i \neq j$; assume, for simplicity, that this j

is unique. (The proof of the more general case is similar.) Let $a_j^* = \lim_{T \longrightarrow 0} a_j(T)$. The left-hand side of equation (13.31) is then clearly a_j^*. The expression on the right-hand side can be rewritten:

$$\lim_{T \longrightarrow 0} a_j(T) \left(\sum_i \left(\frac{a_i(T)}{a_j(T)} \right)^{1/T} \right)^T = a_j^* \left[\lim_{T \longrightarrow 0} \left(\sum_i \left(\frac{a_i(T)}{a_j(T)} \right)^T \right)^{1/T} \right] = a_j^*.$$

The first equality follows from the fact that the $a_j(T)$ sequence is convergent. The second follows from the fact that, because $a_j(T) > a_i(T)$ for all $i \neq j$ and all $T < T_0$, the ratio $a_i(T)/a_j(T)$ is bounded in $[0, 1]$, with $a_j(T)/a_j(T) = 1$; therefore the limit is simply 1. ■

The proof of this lemma concludes the proof of the theorem. ■

We now wish to show the second important fact:

Theorem 13.9

The limit $\tilde{\boldsymbol{\beta}}^{(0)}$ is a proportional reparameterization of $\tilde{P}_\Phi$, that is:

$$\tilde{P}_\Phi(\mathcal{X}) \propto \prod_{r \in \mathbf{R}} (\tilde{\beta}_r^{(0)}(\boldsymbol{c}_r))^{\kappa_r}.$$

PROOF Due to equation (13.26), we have that

$$\tilde{P}_\Phi^{(T)}(\mathcal{X}) = \prod_{r \in \mathbf{R}} (\beta_r^{(T)}(\boldsymbol{c}_r))^{\kappa_r}.$$

We can raise each side to the power T, and obtain that:

$$\tilde{P}_\Phi(\mathcal{X}) = \left(\prod_{r \in \mathbf{R}} (\beta_r^{(T)}(\boldsymbol{c}_r))^{\kappa_r} \right)^T.$$

We can divide each side by

$$\left(\prod_{r \in \mathbf{R}^+} (\bar{\beta}_r^{(T)})^{\kappa_r} \right)^T,$$

to obtain the equality

$$\frac{\tilde{P}_\Phi(\mathcal{X})}{\left(\prod_{r \in \mathbf{R}^+} (\bar{\beta}_r^{(T)})^{\kappa_r} \right)^T} = \prod_r (\tilde{\beta}_r^{(T)}(\boldsymbol{c}_r))^{\kappa_r}.$$

This equality holds for every value of $T > 0$. Moreover, as we argued, the right-hand side is bounded in $[0, 1]$, and hence so is the left-hand side. As a consequence, we have an equality of two bounded continuous functions of T, so that they must also be equal at the limit $T \longrightarrow 0$. It follows that the limiting beliefs $\tilde{\boldsymbol{\beta}}^{(0)}$ are proportional to a reparameterization of $\tilde{P}_\Phi$. ■

13.5.3.3 Discussion

Overall, the analysis in this section reveals interesting connections between three separate algorithms: the linear program relaxation of the MAP problem, the low-temperature limit of sum-product belief propagation with convex counting numbers, and the max-product reparameterization with (the same) convex counting numbers. These connections hold for any set of convex counting numbers and any (positive) distribution $\tilde{P}_\Phi$.

Specifically, we have characterized the solution to the relaxed LP as the limit of a sequence of optimization problems, each defined by a temperature-weighted convex energy functional. Each of these optimization problems can be solved using an algorithm such as convex sum-product BP, which (assuming convergence) produces optimal beliefs for that problem. We have also shown that the beliefs obtained in this sequence can be reformulated to converge to a new set of beliefs that are max-product calibrated. These beliefs are fixed points of the convex max-product BP algorithm. Thus, we can hope to use max-product BP to find these limiting beliefs.

Our earlier results show that the fixed points of convex max-product BP, if they admit a locally optimal assignment, are guaranteed to produce the MAP assignment. We can now make use of the results in this section to shed new light on this analysis.

Theorem 13.10 *Assume that we have a set of max-calibrated beliefs $\beta_r(\boldsymbol{C}_r)$ and $\beta_i(X_i)$ such that equation (13.16) holds. Assume furthermore that ξ^* is a locally consistent joint assignment relative to these beliefs. Then the* MAP-LP *relaxation is tight.*

Proof We first observe that

$$\begin{aligned}\max_{\xi} \log \tilde{P}_\Phi(\xi) &= \max_{Q \in \mathit{Marg}[\mathcal{U}]} \boldsymbol{E}_{\xi \sim Q}\left[\log \tilde{P}_\Phi(\xi)\right] \\ &\leq \max_{\boldsymbol{Q} \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_{\xi \sim \boldsymbol{Q}}\left[\log \tilde{P}_\Phi(\xi)\right], \end{aligned} \tag{13.32}$$

which is equal to the value of MAP-LP. Note that we are abusing notation in the expectation used in the last expression, since $\boldsymbol{Q} \in \mathit{Local}[\mathcal{U}]$ is not a distribution but a set of pseudo-marginals. However, because $\log \tilde{P}_\Phi(\xi)$ factors according to the structure of the clusters in the pseudo-marginals, we can use a set of pseudo-marginals to compute the expectation.

Next, we note that for any set of functions $f_r(\boldsymbol{C}_r)$ whose scopes align with the clusters $\boldsymbol{C}_r$, we have that:

$$\begin{aligned}\max_{Q \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}\left[\sum_r f_r(\boldsymbol{C}_r)\right] &= \max_{Q \in \mathit{Local}[\mathcal{U}]} \sum_r \boldsymbol{E}_{\boldsymbol{C}_r \sim Q}[f_r(\boldsymbol{C}_r)] \\ &\leq \sum_r \max_{\boldsymbol{C}_r} f_r(\boldsymbol{C}_r),\end{aligned}$$

because an expectation is smaller than the max.

We can now apply this derivation to the reformulation of $\tilde{P}_\Phi$ that we get from the reparameterization:

$$\max_{Q \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_Q\left[\log \tilde{P}_\Phi(\xi)\right] = \max_{Q \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_Q\left[\begin{array}{l}\sum_r \nu_r \log(\beta_r(\boldsymbol{c}_r)) + \sum_i \nu_i \log \beta_i(x_i) \\ + \sum_{i,r\,:\,X_i \in \boldsymbol{C}_r} \nu_{r,i} \left(\log \beta_r(\boldsymbol{c}_r) - \log \beta_i(x_i)\right)\end{array}\right].$$

From the preceding derivation, it follows that:

$$\leq \sum_r \max_{\boldsymbol{c}_r} \nu_r \log(\beta_r(\boldsymbol{c}_r)) + \sum_i \max_{x_i} \nu_i \log \beta_i(x_i) + \sum_{i,r \,:\, X_i \in \boldsymbol{C}_r} \max_{\boldsymbol{c}_r; x_i = \boldsymbol{c}_r\langle X_i \rangle} \nu_{r,i} \left(\log \beta_r(\boldsymbol{c}_r) - \log \beta_i(x_i)\right).$$

And from the positivity of the counting numbers, we get

$$= \sum_r \nu_r \max_{\boldsymbol{c}_r} \log(\beta_r(\boldsymbol{c}_r)) + \sum_i \nu_i \max_{x_i} \log \beta_i(x_i) + \sum_{i,r \,:\, X_i \in \boldsymbol{C}_r} \nu_{r,i} \max_{\boldsymbol{c}_r; x_i = \boldsymbol{c}_r\langle X_i \rangle} \left(\log \beta_r(\boldsymbol{c}_r) - \log \beta_i(x_i)\right).$$

Now, due to lemma 13.1 (reformulated for log-factors), we have that ξ^* optimizes each of the maximization expressions, so that we conclude:

$$\begin{aligned} &= \sum_r \nu_r \log(\beta_r(\boldsymbol{c}_r^*)) + \sum_i \nu_i \log \beta_i(x_i^*) \\ &\quad + \sum_{i,r \,:\, X_i \in \boldsymbol{C}_r} \nu_{r,i} \left(\log \beta_r(\boldsymbol{c}_r^*) - \log \beta_i(x_i^*)\right) \\ &= \log \tilde{P}_\Phi(\xi^*). \end{aligned}$$

Putting this conclusion together with equation (13.32), we obtain:

$$\begin{aligned} \max_\xi \log \tilde{P}_\Phi(\xi) &\leq \max_{Q \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_{\xi \sim Q}\left[\log \tilde{P}_\Phi(\xi)\right] \\ &\leq \log \tilde{P}_\Phi(\xi^*). \end{aligned}$$

Because the right-hand side is clearly $\leq$ the left-hand side, the entire inequality holds as an equality, proving that

$$\max_\xi \log \tilde{P}_\Phi(\xi) = \max_{Q \in \mathit{Local}[\mathcal{U}]} \boldsymbol{E}_{\xi \sim Q}\left[\log \tilde{P}_\Phi(\xi)\right],$$

that is, the value of the integer program optimization is the same as that of the relaxed LP. ■

This last fact has important repercussions. In particular, it shows that **convex max-product BP can be decoded only if the LP is tight; otherwise, there is no locally optimal joint assignment, and no decoding is possible. It follows that convex max-product BP provides provably useful results only in cases where** MAP-LP **itself provides the optimal answer to the MAP problem. We note that a similar conclusion does *not* hold for nonconvex variants such as those based on the standard Bethe counting numbers; in particular, standard max-product BP is not an upper bound to** MAP-LP**, and therefore it can return solutions in the interior of the polytope of** MAP-LP**. As a consequence, it may be decodable**

even when the LP is not tight; in that case, the returned joint assignment may be the MAP, or it may be a suboptimal assignment.

This result leaves several intriguing open questions. First, we note that this result shows a connection between the results of max-product and the LP only when the LP is tight. It is an open question whether we can show a general connection between the max-product beliefs and the dual of the original LP. A second question is whether we can construct better techniques that directly solve the LP or its dual; indeed, recent work (see section 13.9) explores a range of other techniques for this task. A third question is whether this technique provides a useful heuristic: Even if the reparameterization we derive does not have a locally consistent joint assignment, we can still use it to construct an assignment using various heuristic methods, such as selecting for each variable X_i the assignment $x_i^* = \arg\max_{x_i} \beta_i(x_i)$. While there are no guarantees about this solution, it may still work well in practice.

13.6 Using Graph Cuts for MAP

In this section, we discuss the important class of *metric* and *semi-metric* MRFs, which we defined in box 4.D. This class has received considerable attention, largely owing to its importance in computer- vision applications. We show how this class of networks, although possibly very densely connected, can admit an optimal or close-to-optimal solution, by virtue of structure in the potentials.

13.6.1 Inference Using Graph Cuts

The basic graph construction is defined for pairwise MRFs consisting solely of binary-valued variables ($\mathcal{V} = \{0, 1\}$). Although this case has restricted applicability, it forms the basis for the general case. As we now show, **the MAP problem for a certain class of binary-valued MRFs can be solved *optimally* using a very simple and efficient *graph-cut* algorithm. Perhaps the most surprising aspect of this reduction is that this algorithm is guaranteed to return the optimal solution in polynomial time, regardless of the structural complexity of the underlying graph. This result stands in contrast to most of the other results presented in this book, where polynomial-time solutions were obtainable only for graphs of low tree width. Equally noteworthy is the fact that a similar result does not hold for sum-product computations over this class of graphs; thus, we have an example of a class of networks where sum-product inference and MAP inference have very different computational properties.**

We first define the min-cut problem for a graph, and then show how the MAP problem can be reduced to it. The min-cut problem is defined by a set of vertices $\mathcal{Z}$, plus two distinguished nodes generally known as s and t. We have a set of directed edges $\mathcal{E}$ over $\mathcal{Z} \cup \{s, t\}$, where each edge $(z_1, z_2) \in \mathcal{E}$ is associated with a nonnegative cost $cost(z_1, z_2)$. A *graph cut* is a disjoint partition of $\mathcal{Z}$ into $\mathcal{Z}_s \cup \mathcal{Z}_t$ such that $s \in \mathcal{Z}_s$ and $t \in \mathcal{Z}_t$. The *cost* of the cut is:

graph cut

$$cost(\mathcal{Z}_s, \mathcal{Z}_t) = \sum_{z_1 \in \mathcal{Z}_s, z_2 \in \mathcal{Z}_t} cost(z_1, z_2).$$

In words, the cost is the total sum of the edges that cross from the $\mathcal{Z}_s$ side of the partition to the $\mathcal{Z}_t$ side. The *minimal cut* is the partition $\mathcal{Z}_s, \mathcal{Z}_t$ that achieves the minimal cost. While

presenting a min-cut algorithm is outside the scope of this book, such algorithms are standard, have polynomial-time complexity, and are very fast in practice.

How do we reduce the MAP problem to one of computing cuts on a graph? Intuitively, we need to design our graph so that a cut corresponds to an assignment to $\mathcal{X}$, and its cost to the value of the assignment. The construction follows straightforwardly from this intuition. Our vertices (other than s, t) represent the variables in our MRF. We use the s side of the cut to represent the label 0, and the t side to represent the label 1. Thus, we map a cut $\mathcal{C} = (\mathcal{Z}_s, \mathcal{Z}_t)$ to the following assignment $\xi^{\mathcal{C}}$:

$$x_i^{\mathcal{C}} = 0 \quad \text{if and only if} \quad z_i \in \mathcal{Z}_s.$$

We begin by demonstrating the construction on the simple case of the generalized Ising model of equation (4.6). Note that energy functions are invariant to additive changes in all of the components, since these just serve to move all entries in $E(x_1, \ldots, x_n)$ by some additive factor, leaving their relative order invariant. Thus, we can assume, without loss of generality, that all components of the energy function are nonnegative. Moreover, we can assume that, for every node i, either $\epsilon_i(1) = 0$ or $\epsilon_i(0) = 0$. We now construct the graph as follows:

- If $\epsilon_i(1) = 0$, we introduce an edge $z_i \to t$, with cost $\epsilon_i(0)$.
- If $\epsilon_i(0) = 0$, we introduce an edge $s \to z_i$, with cost $\epsilon_i(1)$.
- For each pair of variables X_i, X_j that are connected by an edge in the MRF, we introduce both an edge (z_i, z_j) and the edge (z_j, z_i), both with cost $\lambda_{i,j} \geq 0$.

Now, consider the cost of a cut $(\mathcal{Z}_s, \mathcal{Z}_t)$. If $z_i \in \mathcal{Z}_s$, then X_i is assigned a value of 0. In this case, z_i and t are on opposite sides of the cut, and so we will get a contribution of $\epsilon_i(0)$ to the cost of the cut; this contribution is precisely the X_i node energy of the assignment $X_i = 0$, as we would want. The analogous argument applies when $z_i \in \mathcal{Z}_t$. We now consider the edge potential. The edge (z_i, z_j) only makes a contribution to the cut if we place z_i and z_j on opposite sides of the cut; in this case, the contribution is $\lambda_{i,j}$. Conversely, the pair X_i, X_j makes a contribution of $\lambda_{i,j}$ to the energy function if $X_i \neq X_j$, and otherwise it contributes 0. Thus, the contribution of the edge to the cut is precisely the same as the contribution of the node pair to the energy function. Overall, we have shown that the cost of the cut is precisely the same as the energy of the corresponding assignment. Thus, the min-cut algorithm is guaranteed to find the assignment to $\mathcal{X}$ that minimizes the energy function, that is, ξ^{map}.

Example 13.17

Consider a simple example where we have four variables X_1, X_2, X_3, X_4 connected in a loop with the edges X_1—X_2, X_2—X_3, X_3—X_4, X_1—X_4. Assume we have the following energies, where we list only components that are nonzero:

$$\begin{array}{llll} \epsilon_1(0) = 7 & \epsilon_2(1) = 2 & \epsilon_3(1) = 1 & \epsilon_4(1) = 6 \\ \lambda_{1,2} = 6 & \lambda_{2,3} = 6 & \lambda_{3,4} = 2 & \lambda_{1,4} = 1. \end{array}$$

The graph construction and the minimum cut for this example are shown in figure 13.5.

Going by the node potentials alone, the optimal assignment is $X_1 = 1$, $X_2 = 0$, $X_3 = 0$, $X_4 = 0$. However, we also have interaction potentials that encourage agreement between neighboring nodes. In particular, there are fairly strong potentials that induce $X_1 = X_2$ and $X_2 = X_3$. Thus, the node-optimal assignment achieves a penalty of 7 from the contributions of $\lambda_{1,2}$ and $\lambda_{1,4}$.

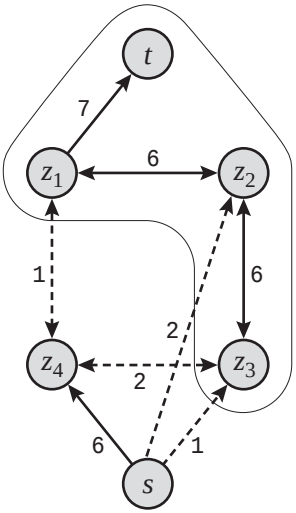


Figure 13.5 Example graph construction for applying min-cut to the binary MAP problem, based on example 13.17. Numbers on the edges represent their weight. The cut is represented by the set of nodes in $\mathcal{Z}_t$. Dashed edges are ones that participate in the cut; note that only one of the two directions of a bidirected edge contributes to the weight of the cut, which is 6 in this example.

Conversely, the assignment where X_2 and X_3 agree with X_1 gets a penalty of only 6 from the X_2 and X_3 node contributions and from the weaker edge potentials $\lambda_{3,4}$ and $\lambda_{1,4}$. Thus, the overall MAP assignment has $X_1 = 1$, $X_2 = 1$, $X_3 = 1$, $X_4 = 0$. ■

As we mentioned, the MAP problem in such graphs reduces to a minimum cut problem regardless of the network connectivity. Thus, this approach allows us to find MAP solution for a class of MRFs for which probability computations are intractable.

We can easily extend this construction beyond the generalized Ising model:

Definition 13.5 submodular energy function

A pairwise energy $\epsilon_{i,j}(\cdot,\cdot)$ is said to be submodular *if*

$$\epsilon_{i,j}(1,1) + \epsilon_{i,j}(0,0) \leq \epsilon_{i,j}(1,0) + \epsilon_{i,j}(0,1). \tag{13.33}$$

■

The graph construction for submodular energies, which is shown in detail in algorithm 13.4, is a little more elaborate. It first normalizes each edge potential by subtracting $\epsilon_{i,j}(0,0)$ from all entries; this operation subtracts a constant amount from the energies of all assignments, corresponding to a constant multiple in probability space, which only changes the (in this case irrelevant) partition function. It then moves as much mass as possible to the individual node potentials for i and j. These steps leave a single pairwise term that defines an energy only for the assignment $v_i = 0, v_j = 1$:

$$\epsilon'_{i,j}(0,1) = \epsilon_{i,j}(1,0) + \epsilon_{i,j}(0,1) - \epsilon_{i,j}(0,0) - \epsilon_{i,j}(1,1).$$

Algorithm 13.4 Graph-cut algorithm for MAP in pairwise binary MRFs with submodular potentials

Procedure MinCut-MAP (
  ϵ // Singleton and pairwise submodular energy factors
)
  // Define the energy function
  for all i
    $\epsilon'_i \leftarrow \epsilon_i$
  Initialize $\epsilon'_{i,j}$ to 0 for all i, j
  for all pairs $i < j$
    $\epsilon'_i(1) \leftarrow \epsilon'_i(1) + (\epsilon_{i,j}(1,0) - \epsilon_{i,j}(0,0))$
    $\epsilon'_j(1) \leftarrow \epsilon'_j(1) + (\epsilon_{i,j}(1,1) - \epsilon_{i,j}(1,0))$
    $\epsilon'_{i,j}(0,1) \leftarrow \epsilon_{i,j}(1,0) + \epsilon_{i,j}(0,1) - \epsilon_{i,j}(0,0) - \epsilon_{i,j}(1,1)$

  // Construct the graph
  for all i
    if $\epsilon'_i(1) > \epsilon'_i(0)$ **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s, z_i)\}$
      $cost(s, z_i) \leftarrow \epsilon'_i(1) - \epsilon'_i(0)$
    else
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(z_i, t)\}$
      $cost(z_i, t) \leftarrow \epsilon'_i(0) - \epsilon'_i(1)$
    for all pairs $i < j$ such that $\epsilon'_{i,j}(0,1) > 0$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(z_i, z_j)\}$
      $cost(z_i, z_j) \leftarrow \epsilon'_{i,j}(0,1)$

  $\boldsymbol{t} \leftarrow$ MinCut$(\{z_1, \ldots, z_n\}, \mathcal{E})$
    // MinCut returns $t_i = 1$ iff $z_i \in \mathcal{Z}_t$
  return $\boldsymbol{t}$

Because of submodularity, this term satisfies $\epsilon'_{i,j}(0,1) \geq 0$. The algorithm executes this transformation for every pairwise potential i, j. The resulting energy function can easily be converted into a graph using essentially the same construction that we used earlier; the only slight difference is that for our new energy function $\epsilon'_{i,j}(v_i, v_j)$ we need to introduce only the edge (z_i, z_j), with cost $\epsilon'_{i,j}(0,1)$; we do not introduce the opposite edge (z_j, z_i). We now use the same mapping between s-t cuts in the graph and assignment to the variables $X_1, \ldots, X_n$. It is not difficult to verify that the cost of an s-t cut $\mathcal{C}$ in the resulting graph is precisely $E(\xi^{\mathcal{C}}) + \text{Const}$ (see exercise 13.14). Thus, finding the minimum cut in this graph directly gives us the cost-minimizing assignment ξ^{map}.

Note that for pairwise submodular energy, there is an LP relaxation of the MAP integer optimization, which is tight. Thus, this result provides another example where having a tight LP relaxation allows us to find the optimal MAP assignment.

13.6.2 Nonbinary Variables

In the case of nonbinary variables, we can no longer use a graph construction to solve the MRF optimally. Indeed, the problem of optimizing the energy function, even if it is submodular, is $\mathcal{NP}$-hard in this case. Here, a very useful technique is to take greedy hill-climbing steps, but where each step involves a globally optimal solution to a simplified problem. Two types of steps have been utilized extensively: *alpha-expansion* and *alpha-beta swap*. As we will show, under appropriate conditions on the energy function, both the alpha-expansion step and the alpha-beta-swap steps can be performed optimally by applying the min-cut procedure to an appropriately constructed MRF. Thus, the search procedure can take a global step in the space.

alpha-expansion

The *alpha-expansion* considers a particular value v; the step simultaneously considers all of the variables X_i in the MRF, and allows each of them to take one of two values: it can keep its current value x_i, or change its value to v. Thus, the step expands the set of variables that take the label v; the label v is often denoted α in the literature; hence the name alpha-expansion.

The alpha-expansion algorithm is shown in algorithm 13.5. It consists of repeated applications of alpha-expansion steps, for different labels v. Each alpha-expansion step is defined relative to our current assignment $\boldsymbol{x}$ and a target label v. Our goal is to select, for each variable X_i whose current label x_i is other than v, whether in the new assignment $\boldsymbol{x}'$ its new label will remain x_i or move to v. We do so using a new MRF that has binary variables T_i for each variable X_i; we then define a new assignment $\boldsymbol{x}'$ so that $x'_i = x_i$ if $T_i = t_i^0$, and $x'_i = v$ if $T_i = t_i^1$.

restricted energy function

We define a new *restricted energy function* E' using the following set of potentials:

$$\begin{array}{lcl} \epsilon'_i(t_i^0) & = & \epsilon_i(x_i) \\ \epsilon'_i(t_i^1) & = & \epsilon_i(v) \\ \epsilon'_{i,j}(t_i^0, t_j^0) & = & \epsilon_{i,j}(x_i, x_j) \\ \epsilon'_{i,j}(t_i^0, t_j^1) & = & \epsilon_{i,j}(x_i, v) \\ \epsilon'_{i,j}(t_i^1, t_j^0) & = & \epsilon_{i,j}(v, x_j) \\ \epsilon'_{i,j}(t_i^1, t_j^1) & = & \epsilon_{i,j}(v, v) \end{array} \tag{13.34}$$

It is straightforward to see that for any assignment $\boldsymbol{t}$, $E'(\boldsymbol{t}) = E(\boldsymbol{x}')$. Thus, finding the optimal $\boldsymbol{t}$ corresponds to finding the optimal $\boldsymbol{x}'$ in the restricted space of v-expansions of $\boldsymbol{x}$.

In order to optimize $\boldsymbol{t}$ using graph cuts, the new energy E' needs to be submodular, as in equation (13.33). Plugging in the definition of the new potentials, we get the following constraint:

$$\epsilon_{i,j}(x_i, x_j) + \epsilon_{i,j}(v, v) \leq \epsilon_{i,j}(x_i, v) + \epsilon_{i,j}(v, x_j).$$

Now, if we have an MRF defined by some distance function μ, then $\epsilon_{i,j}(v, v) = 0$ by reflexivity, and the remaining inequality is a direct consequence of the triangle inequality. Thus, we can apply the alpha-expansion procedure to any metric MRF.

alpha-beta swap

The second type of step is the *alpha-beta swap*. Here, we consider two labels: v_1 and v_2. The step allows each variable whose current label is v_1 to keep its value or change it to v_2, and conversely for variables currently labeled v_2. Like the alpha-expansion step, the alpha-beta swap over a given assignment $\boldsymbol{x}$ can be defined easily by constructing a new energy function, over which min-cut can be performed. The details are left as an exercise (exercise 13.15). We note that the alpha-beta-swap operation requires only that the energy function be a semimetric (that is, the triangle inequality is not required).

These two steps allow us to use the min-cut procedure as a subroutine in solving the MAP problem in metric or semimetric MRFs with nonbinary variables.

Algorithm 13.5 Alpha-expansion algorithm

```
Procedure Alpha-Expansion (
    ϵ,    // Singleton and pairwise energies
    x     // Some initial assignment
)
1   repeat
2     change ← false
3     for k = 1, ..., K
4       t ← Alpha-Expand(ϵ, x, v_k)
5       for i = 1, ..., n
6         if t_i = 1 then
7           x_i ← v_k     // If t_i = 0, x_i doesn't change
8           change ← true
9   until change = false
10  return (x)

Procedure Alpha-Expand (
    ϵ,
    x     // Current assignment
    v     // Expansion label
)
1   Define ϵ' as in equation (13.34)
2   return MinCut-MAP(ϵ')
```

Box 13.B — Case Study: Energy Minimization in Computer Vision. *Over the past few years, MRFs have become a standard tool for addressing a range of low-level vision tasks, some of which we reviewed in box 4.B. As we discussed, the pairwise potentials in these models are often aimed at penalizing discrepancies between the values of adjacent pixels, and hence they often naturally satisfy the submodularity assumption that are necessary for the application of graph cut methods. Also very popular is the TRW-S variant of the convex belief propagation algorithms, described in box 13.A. Standard belief propagation has also been used in multiple applications.*

Vision problems pose some significant challenges. Although the grid structures associated with images are not dense, they are very large, and they contain many tight loops, which can pose difficulties for convergence of the message passing algorithm. Moreover, in some tasks, such as stereo reconstruction*, the value space of the variables is a discretization of a continuous space, and therefore many values are required to get a reasonable approximation. As a consequence, the representation of the pairwise potentials can get very large, leading to memory problems.*

stereo reconstruction

A number of fairly comprehensive empirical studies have been done comparing the various methods on a suite of computer-vision benchmark problems. By and large, it seems that for the grid-structured networks that we described, graph-cut methods with the alpha-expansion step and TRW-S are fairly comparable, with the graph-cut methods dominating in running time; both significantly

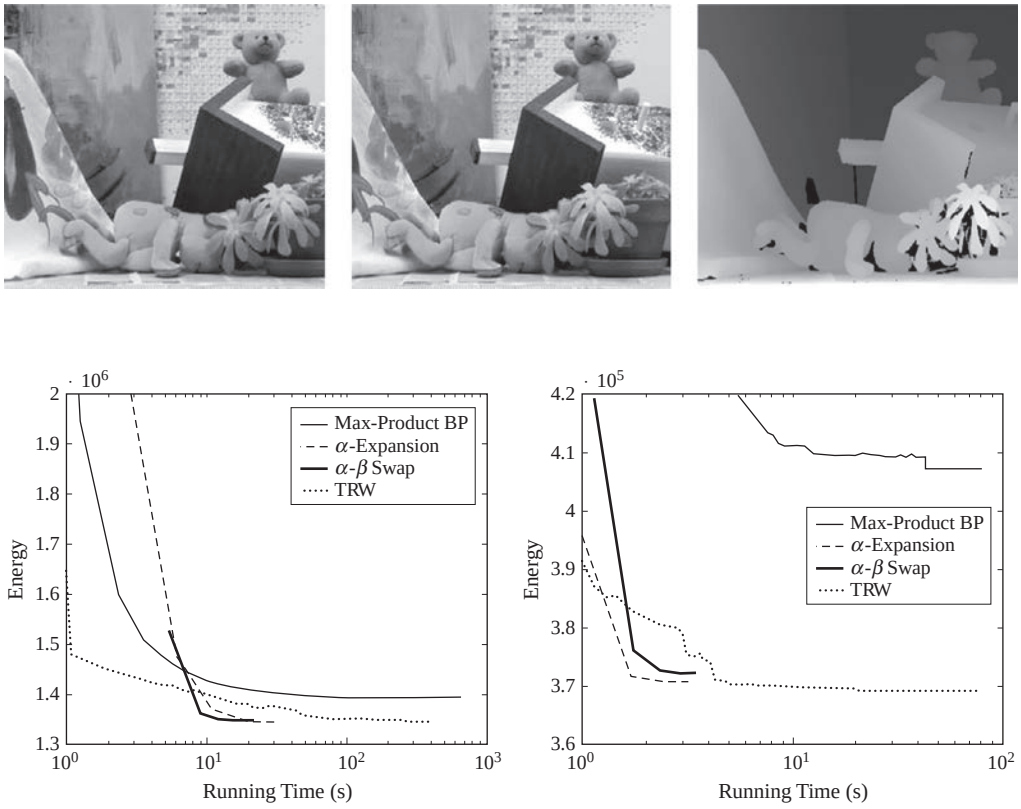


Figure 13.B.1 — MAP inference for stereo reconstruction The top row contains a pair of stereo images for a problem known as Teddy and the target output (darker pixels denote a larger z value); the images are taken from Scharstein and Szeliski (2003). The bottom row shows the best energy obtained as a function of time by several different MAP algorithms:max-product BP, the TRW variant of convex BP, min-cut with alpha-expansion, and min-cut with alpha-beta swap. The left image is for Teddy, and the right is for a different stereo problem called Tsukuba.

outperform the other methods. Figure 13.B.1 shows some sample results on stereo-reconstruction problems; here, the energies are close to submodular, allowing the application of a range of different methods.

The fact that convex BP is solving the dual problem to the relaxed LP allows it to provide a lower bound on the energy of the true MAP assignment. Moreover, as we discussed, it can sometimes provide optimality guarantees on the inferred solution. Thus, it is sometimes possible to compare the results of these methods to the true global optimum of the energy function. Somewhat surprisingly, it appears that both methods come very close to achieving optimal energies on a large fraction of these benchmark problems, suggesting that the problem of energy minimization for these MRFs is

essentially solved.

In contrast to this optimistic viewpoint is the observation that the energy minimizing configuration is often significantly worse than the "target" assignment (for example, the true depth disparity in a stereo reconstruction problem). In other words, the ground truth often has a worse energy (lower probability) than the assignment that optimizes the energy function. This finding suggests that a key problem is that of designing better energy functions, which better capture the structure of our target assignments. This topic has been the focus of much recent work. In many cases, the resulting energies involve nonlocal interactions between the pixels, and are therefore significantly more complex. Some evidence suggests that as the graph becomes more dense and less local, belief propagation methods start to degrade. Conversely, as the potentials become less submodular, the graph-cut methods become less applicable. Thus, the design of new energy-minimization methods that are applicable to these richer energy functions is a topic of significant current interest.

13.7 Local Search Algorithms ⋆

A final class of methods that have been applied to MAP and marginal MAP queries are methods that search over the space of assignments. The task of searching for a high-weight (or low-cost) assignment of values to a set of variables is a central one in many applications, and it has received attention in a number of communities. Methods for addressing this task come in many flavors.

systematic search

Some of those methods are *systematic*: They search the space so as to ensure that assignments that are not considered are not optimal, and thereby guarantee an optimal solution. Such methods generally search over the space of partial assignments, starting with the empty assignment, and assigning variables one at a time. One such method, known as *branch-and-bound*, is described in appendix A.4.3.

branch-and-bound

Other methods are nonsystematic, and they come without performance guarantees. Here, many of the methods search over the space of full assignments, usually by making local changes to the assignment so as to improve its score. These *local search* methods generally provide no guarantees of optimality. Appendix A.4.2 describes some of the techniques that are most commonly applied in practice.

local search

The application of search techniques to the MAP problem is a fairly straightforward process: The *search space* is defined by the possible assignments ξ to $\mathcal{X}$, and $\log \tilde{P}(\xi)$ is the score; we omit details. Although generally less powerful than the methods we described earlier, these methods do have some advantages. For example, the *beam search* method of appendix A.4.2 provides a useful alternative in cases where the complete model is too large to fit into memory; see exercise 15.10. We also note that branch-and-bound does provide a simple method for finding the K most likely assignment; see exercise 13.18. This algorithm requires at least as much computation time as the clique tree–based algorithm, but significantly less space.

search space

beam search

These methods have much greater applicability in the context of *marginal MAP* problem, where most other methods are not (currently) applicable. Here, we search over the space of assignments $\boldsymbol{y}$ to the max-variables $\boldsymbol{Y}$. Here, we conduct the search so that we can fix some or all of the max-variables to have a concrete assignment. As we show, this allows us to remove the constraint on the variable elimination ordering, allowing an unrestricted ordering to be used.

marginal MAP

Here, we search over the space of assignments $\boldsymbol{y}$ for those that maximize

$$\text{score}(\boldsymbol{y}) = \sum_{\boldsymbol{W}} \tilde{P}_\Phi(\boldsymbol{y}, \boldsymbol{W}). \tag{13.35}$$

search operator

Several search procedures are appropriate in this setting. In one approach, we use some local search algorithm, as in appendix A.4.2. As usual in local search, the algorithm begins with some complete assignment $\boldsymbol{y}_0$ to $\boldsymbol{Y}$. We then consider applying different *search operators* to $\boldsymbol{y}$; for each such operator o, we produce a new partial assignment $\boldsymbol{y}' = o(\boldsymbol{y})$ as a successor to the current state, which is evaluated by computing $\text{score}(\boldsymbol{y}')$. Importantly, since we now have a complete assignment to the max-variables $\boldsymbol{y}' = o(\boldsymbol{y})$, the resulting score is simply a sum-product expression, and it can be computed by standard sum-product elimination of the variables $\boldsymbol{W}$, with no constraints on the variable ordering. The tree-width in these cases is usually much smaller than in the constrained case; for example, in the network of figure 13.2, the network for a fixed assignment $\boldsymbol{y}'$ is simply a chain, and the computation of the score can therefore be done in time linear in n.

tabu search

While we can consider a variety of search operators, the most straightforward are operators of the form $do(Y_i = y_i^j)$, which set a variable $Y_i \in \boldsymbol{Y}$ to the value y_i^j. We can now apply any greedy local-search algorithm, such as those described in appendix A.4.2. Empirical evidence suggests that greedy hill climbing with *tabu search* performs very well on this task, especially if initialized intelligently. In particular, one simple yet good heuristic is to calibrate the clique tree with no assignment to the max-variables; we then compute, for each Y_i its unnormalized probability $\tilde{P}_\Phi(Y_i)$ (which can be extracted from any clique containing Y_i), and initialize $y_i = \arg\max_{Y_i} \tilde{P}_\Phi(Y_i)$.

While simple in principle, a naive implementation of this algorithm can be quite costly. Let $k = |\boldsymbol{Y}|$ and assume for simplicity that $|Val(Y_i)| = d$ for all $Y_i \in \boldsymbol{Y}$. Each step of the search requires $k \times (d-1)$ evaluations of score, each of which involves a run of probabilistic inference over the network. Even for simple networks, this cost can often be prohibitive.

dynamic programming

Fortunately, we can greatly improve the computational performance of this algorithm using the same type of *dynamic programming* tricks that we used in other parts of this book. Most important is the observation that **we can compute the score of all of the operators in our search using a single run of clique tree propagation, in the clique tree corresponding to an unconstrained elimination ordering.** Let $\mathcal{T}$ be an *unconstrained* clique tree over $\mathcal{X} = \boldsymbol{Y} \cup \boldsymbol{W}$, initialized with the original potentials of $\tilde{P}_\Phi$. Let $\boldsymbol{y}$ be our current assignment to $\boldsymbol{Y}$. For any Y_i, let $\boldsymbol{Y}_{-i} = \boldsymbol{Y} - \{Y_i\}$ and $\boldsymbol{y}_{-i}$ be the assignment in $\boldsymbol{y}$ to $\boldsymbol{Y}_{-i}$. We can use the algorithm developed in exercise 10.12 to compute $\tilde{P}_\Phi(Y_i, \boldsymbol{y}_{-i})$ for every $Y_i \in \boldsymbol{Y}$. Recall that this algorithm requires only a single clique tree calibration that computes all of the messages; with those messages, each clique that contains a variable Y_i can locally compute $\tilde{P}_\Phi(Y_i, \boldsymbol{y}_{-i})$ in time that is linear in the size of the clique. This idea reduces the cost of each step by a factor of $O(kd)$, an enormous saving. For example, in the network of figure 13.2, we can use a clique tree whose cliques are of the form X_i, Y_{i+1}, X_{i+1}, with sepsets X_i between cliques. Here, the maximum clique size is 3, and the computation requires time linear in k.

We can also use search methods other than local hill climbing. One alternative is to utilize a systematic search procedure that is guaranteed to find the exact solution. Particularly well suited to this task is the branch-and-bound search described in appendix A.4.3. Recall that branch-and-bound systematically explores partial assignments to the variables $\boldsymbol{Y}$; it only discards a partial

assignment $\boldsymbol{y}'$ if it already has a complete solution $\boldsymbol{y}$ that is provably better than the best possible solution that one can obtain by extending $\boldsymbol{y}'$ to a complete assignment. This pruning relies on having a way of estimating the upper bound on a partial assignment $\boldsymbol{y}'$. In our setting, such an upper bound can be obtained by using variable elimination, ignoring the constraint on the ordering whereby all summations occur before all maximizations. An algorithm based on these ideas is developed further in exercise 13.20.

13.8 Summary

In this chapter, we have considered the problem of finding the MAP assignment and described a number of methods for addressing it. The MAP problem has a broad range of applications, in computer vision, computational biology, speech recognition, and more. Although the use of MAP inference loses us the ability to measure our confidence (or uncertainty) in our conclusions, there are good reasons nevertheless for using a single MAP assignment rather than using the marginal probabilities of the different variables. One is the preference for obtaining a single coherent joint assignment, whereas a set of individual marginals may not make sense as a whole. The second is that there are inference methods that are applicable to the MAP problem and not to the task of computing probabilities, so that the former may be tractable even when the latter is not.

The methods we discussed fall into several major categories. The variable elimination method is very similar to the approaches we discussed in chapter 9, where we replace summation with maximization. The only slight extension is the traceback procedure, which allows us to identify the MAP assignment once the variable elimination process is complete.

Although one can view the max-product clique tree algorithm as a dynamic programming extension of variable elimination, it is more illuminating to view it as a method for reparameterizing the distribution to produce a max-calibrated set of beliefs. With this reparameterization, we can convert the global optimization problem — finding a coherent joint assignment — to a local optimization problem — finding a set of local assignments each of which optimizes its (calibrated) belief. Importantly, the same view also characterizes the cluster-graph-based belief propagation algorithms. The properties of max-calibrated beliefs allow us to prove strong (local or global) optimality properties for the results of these different message passing algorithms. In particular, for message passing with convex counting numbers we can sometimes construct an assignment that is the true MAP.

A seemingly very different class of methods is based on considering the integer program that directly encodes our optimization problem, and then constructing a relaxation as a linear program. Somewhat surprisingly, there is a deep connection between the convex max-product BP algorithm and the linear program relaxation. In particular, the solution to the dual problem of this LP is a fixed point of any convex max-product BP algorithm; thus, these algorithms can be viewed as a computational method for solving this dual problem. The use of these message passing methods offers a trade-off: they are space-efficient and easy to implement, but they may not converge to the optimum of the dual problem.

Importantly, the fixed point of a convex BP algorithm can be used to provide a MAP assignment only if the MAP LP is a tight relaxation of the integer MAP optimization problem. Thus, it appears that the LP relaxation is the fundamental construct in the application and analysis of

the convex BP algorithms. This conclusion motivates two recent lines of work in MAP inference: One line attempts to construct tighter relaxations to the MAP optimization problem; importantly, since the same relaxation is used for both the free energy optimization in section 11.3.6 and for the MAP relaxations, progress made on improved relaxations for one task is directly useful for the other. The second line of work attempts to solve the LP or its dual using techniques other than message passing. While the problems are convex and hence can in principle be solved directly using standard techniques, the size of the problems makes the cost of this simple approach prohibitive in many practical applications. However, the rich and well-developed theory of convex optimization provides a wealth of potential tools, and some are already being adapted to take advantage of the structure of the MAP problem. It is likely that eventually these algorithms will replace convex BP as the method of choice for solving the dual. See section 13.9 for some references along those lines.

A different class of algorithms is based on reducing the MAP problem in pairwise, binary MRFs to one of finding the minimum cut in a graph. Although seemingly restrictive, this procedure forms a basic building block for solving a much broader class of MRFs. These methods provide an effective solution method only for MRFs where the potentials satisfy (or almost satisfy) the submodularity property. Conversely, their complexity depends fairly little on the complexity of the graph (the number of edges); as such, they allow certain MRFs to be solved efficiently that are not tractable to any other method. **Empirically, for energies that are close to submodular, the methods based on graph cuts are significantly faster than those based on message passing.** We note that in this case, also, there is an interesting connection to the linear programming view: The case that admits an optimal solution using minimum cut (pairwise, binary MRFs whose potentials are submodular) are also ones where there is a tight LP relaxation to the MAP problem. Thus, one can view the minimum-cut algorithm as a specialized method for exploiting special structure in the LP for solving it more efficiently.

In contrast to the huge volume of work on the MAP problem, relatively little work has been done on the marginal MAP problem. This lack is, in some sense, not surprising: the intrinsic difficulty of the problem is daunting and eliminates any hope of a general-purpose solution. Nevertheless, it would be interesting to see whether some of the recent algorithmic techniques developed for the MAP problem could be extended to apply to the marginal MAP case, leading to new solutions to the marginal MAP problem for at least a subset of MRFs.

13.9 Relevant Literature

We begin by reminding the reader, before tackling the literature, that there is a conflict of terminologies here: In some papers, the MAP problem is called MPE, whereas the marginal MAP problem is called simply MAP.

The problem of finding the MAP assignment in a probabilistic model was first addressed by Viterbi (1967), in the context of hidden Markov models; this algorithm came to be called the *Viterbi algorithm*. A generalization to other singly connected Bayesian networks was first proposed by Pearl (1988). The clique tree algorithm for this problem was described by Lauritzen and Spiegelhalter (1988). Shimony (1994) showed that the MAP problem is $\mathcal{NP}$-hard in general networks.

Viterbi algorithm

The problem of finding a MAP assignment to an MRF is equivalent (up to a negative-logarithm

transformation) to the task of minimizing an energy function that is defined as a sum of terms, each involving a small number of variables. There is a considerably body of literature on the *energy minimization* problem, in both continuous and discrete space. Extensive work on energy minimization in MRFs has been done in the computer-vision community, where the locality of the spatial structure naturally defines a highly structured, often pairwise, MRF.

energy minimization

Early work on the energy minimization task focused on hill-climbing techniques, such as simple coordinate ascent (known under the name *iterated conditional modes* (Besag 1986)) or simulated annealing (Barnard 1989). Many other search methods for the MAP problem have been proposed, including systematic approaches such as branch-and-bound (Santos 1991; Marinescu et al. 2003).

iterated conditional modes

The interest in max-product belief propagation on a loopy graph first arose in the context of turbo-decoding. The first general-purpose theoretical analysis for this approach was provided by Weiss and Freeman (2001b), who showed optimality properties of an assignment derived from an unambiguous set of beliefs reached at convergence of max-product BP. In particular, they showed that the assignment is the global optimum for networks involving only a single loop, and a strong local optimum (robust to changes in the assignments for a disjoint collection of single loops and trees) in general.

Wainwright, Jaakkola, and Willsky (2004) first proposed the view of message passing as reparameterizing the distribution so as to get the local beliefs to correspond to max-marginals. In subsequent work, Wainwright, Jaakkola, and Willsky (2005) developed the first convexified message passing algorithm for the MAP problem. The algorithm, known as TRW, used an approximation of the energy function based on a convex combination of trees. This paper was the first to show lemma 13.1. It also showed that if a fixed point of the TRW algorithm satisfied a stronger property than local optimality, it provided the MAP assignment. However, the TRW algorithm did not monotonically improve its objective, and indeed the algorithm was generally not convergent. Kolmogorov (2006) defined TRW-S, a variant of TRW that passes message asynchronously, in a particular order. TRW-S is guaranteed to increase the objective monotonically, and hence is convergent. However, TRW-S is not guaranteed to converge to the global optimum of the dual objective, since it can get stuck in local optima.

The connections between max-product BP, the lower-temperature limit of sum-product BP, and the linear programming relaxation were studied by Weiss, Yanover, and Meltzer (2007). They also showed results on the optimality of partial assignments extracted from unambiguous beliefs derived from convex BP fixed points, extending earlier results of Kolmogorov and Wainwright (2005) for TRW-S.

Max flow techniques to solve submodular binary problems were originally developed by Boros, Hammer and collaborators (Hammer 1965; Boros and Hammer 2002). These techniques were popularized in the vision-MRF community by Greig, Porteous, and Seheult (1989), who were the first to apply these techniques to images. Ishikawa (2003) extended this work to the nonbinary case, but assuming that the interaction between variables is convex. Boykov, Veksler, and Zabih (2001) were the first to propose the alpha-expansion and alpha-beta swap steps, which allow the application of graph-cut methods to nonbinary problems; they also prove certain guarantees regarding the energy of the assignment found by these global steps, relative to the energy of the optimal MAP assignment. Kolmogorov and Zabih (2004) generalized and analyzed the graph constructions used in these methods, using techniques similar to those described by Boros and Hammer (2002). Recent work extends the scope of the MRFs to which these techniques

can be applied, by introducing preprocessing steps that modify factors that do not satisfy the submodularity assumptions. For example, Rother et al. (2005) consider a method that truncates the potentials that do not conform to submodularity, as part of the iterative alpha-expansion algorithm, and they show that this approach, although not making optimal alpha-expansion steps, is still guaranteed to improve the objective at each iteration. We note that, for the case of metric potentials, belief propagation algorithms such as TRW also do well (see box 13.B); moreover, Felzenszwalb and Huttenlocher (2006) show how the computational cost of each message passing step can be reduced from $O(K^2)$ to $O(K)$, where K is the total number of labels, reducing the cost of these algorithms in this setting.

Szeliski et al. (2008) perform an in-depth empirical comparison of the performance of different methods on an ensemble of computer vision benchmark problems. Other empirical comparisons include Meltzer et al. (2005); Kolmogorov and Rother (2006); Yanover et al. (2006).

The LP relaxation for MRFs was first proposed by Schlesinger (1976), and then subsequently rediscovered independently by several researchers. Of these, the most relevant to our presentation is the work of Wainwright, Jaakkola, and Willsky (2005), who also established the first connection between the LP dual and message passing algorithms, and proposed the TRW algorithm. Various extensions were subsequently proposed by various authors, based on different relaxations that require more complex convex optimization algorithms (Muramatsu and Suzuki 2003; Kumar et al. 2006; Ravikumar and Lafferty 2006). Surprisingly, Kumar et al. (2007) subsequently showed that the simple LP relaxation was tighter (that is, better) relaxation than all of those more sophisticated methods.

A spate of recent works (Komodakis et al. 2007; Schlesinger and Giginyak 2007a,b; Sontag and Jaakkola 2007; Globerson and Jaakkola 2007b; Werner 2007; Sontag et al. 2008) make much deeper use of the linear programming relaxation of the MAP problem and of its dual. Globerson and Jaakkola (2007b); Komodakis et al. (2007) both demonstrate a message passing algorithm derived from this dual. The algorithm of Komodakis et al. is based on a dual decomposition algorithm, and is therefore guaranteed to converge to the optimum of the dual objective. Solving the LP relaxation or its dual does not generally give rise to the optimal MAP assignment. The work of Sontag and Jaakkola (2007); Sontag et al. (2008) shows how we can use the LP formulation to gradually add local constraints that hold for any set of pseudo-marginals defined by a real distribution. These constraints make the optimization space a tighter relaxation of the marginal polytope and thereby lead to improved approximations. Sontag et al. present empirical results that show that a small number of constraints often suffice to define the optimal MAP assignment.

Komodakis and colleagues 2005; 2007 also make use of LP duality in the context of graph cut methods, where it corresponds to the well-known duality between min-cut and max-flow. They use this approach to derive primal-dual methods that speed up and extend the alpha-expansion method in several ways.

Santos (1991, 1994) studied the question of finding the M most likely assignments. He presented an exact algorithm that uses the linear programming relaxation of the integer program, augmented with a branch-and-bound search that uses the LP as the bound. Nilsson (1998) provides an alternative algorithm that uses propagation in clique trees. Yanover and Weiss (2003) subsequently generalized this algorithm for the case of loopy BP.

Park and Darwiche extensively studied the marginal MAP problem, providing complexity results (Park 2002; Park and Darwiche 2001), local search algorithms (Park and Darwiche 2004a)

(including an efficient clique tree implementation), and a systematic branch-and-bound algorithm (Park and Darwiche 2003) based on the bound obtained by exchanging summation and maximization.

The study of constraint satisfaction problems, and related problems such as Boolean satisfiability (see appendix A.3.4) is the focus of a thriving research community, and much progress has been made. One recent overview can be found in the textbook of Dechter (2003). There has been a growing interest recently in relating CSP methods to belief propagation techniques, most notably the *survey propagation* (for example, (Maneva et al. 2007)).

survey propagation

13.10 Exercises

Exercise 13.1⋆⋆

Prove theorem 13.1.

Exercise 13.2⋆

Provide a structured variable elimination algorithm that solves the MAP task for networks with rule-based CPDs.

a. Modify the algorithm Rule-Sum-Product-Eliminate-Var in algorithm 9.7 to deal with the max-product task.
b. Show how we can perform the backward phase that constructs the most likely assignment to $\mathcal{X}$. Make sure you describe which information needs to be stored in the forward phase so as to enable the backward phase.

Exercise 13.3

Prove theorem 13.4.

Exercise 13.4

Show how to adapt Traceback-MAP of algorithm 13.1 to find the marginal MAP assignment, given the factors computed by a run of variable elimination for marginal MAP.

Exercise 13.5⋆

Consider the task of finding the second-most-likely assignment in a graphical model. Assume that we have produced a max-calibrated clique tree.

a. Assume that the probabilistic model is unambiguous. Show how we can find the second-best assignment using a single pass over the clique tree.
b. Now answer the same question in the case where the probabilistic model is ambiguous. Your method should use only the precomputed max-marginals.

Exercise 13.6⋆

Now, consider the task of finding the third-most-likely assignment in a graphical model. Finding the third-most-probable assignment is more complicated, since it cannot be computed from max-marginals alone.

a. We define the notion of *constrained max-marginal*: a max-marginal in a distribution that has some variable X_k constrained to take on only certain values. For $D_k \subset Val(X_k)$, we define the constrained max-marginal of X_i to be:

$$MaxMarg_{\tilde{P}_{X_k \in D_k}}(X_i = x_i) = \max_{\{\boldsymbol{x}: X_i = x_i, X_k \in D_k\}} \tilde{P}(\boldsymbol{x}).$$

Explain how to compute the preceding constrained max-marginals for all i and x_i using max-product message passing.

b. Find the third-most-probable assignment by using two sets of constrained max-marginals.

Exercise 13.7

Prove proposition 13.1.

Exercise 13.8

Prove proposition 13.3.

Exercise 13.9

Assume that max-product belief propagation converges to a set of calibrated beliefs $\beta_i(\boldsymbol{C}_i)$. Assume that each belief is unambiguous, so that it has a unique maximizing assignment $\boldsymbol{c}_i^*$. Prove that all of these locally optimizing assignments are consistent with each other, in that if $X_k = x_k^*$ in one assignment $\boldsymbol{c}_i^*$, then $X_k = x_k^*$ in every other assignment $\boldsymbol{c}_j^*$ for which $X_k \in \boldsymbol{C}_j$.

Exercise 13.10

Construct an example of a max-product calibrated cluster graph in which (at least) some beliefs have two locally optimal assignments, such that one local assignment can be extended into a globally consistent joint assignment (across all beliefs), and the other cannot.

Exercise 13.11⋆

Consider a cluster graph $\mathcal{U}$ that contains only a single loop, and assume that we have a set of max-product calibrated beliefs $\{\beta_i\}$ for $\mathcal{U}$ and an assignment ξ^* that is locally optimal relative to $\{\beta_i\}$. Prove that ξ^* is the MAP assignment relative to the distribution $P_{\mathcal{U}}$. (Hint: Use lemma 13.1 and a proof similar to that of theorem 13.6.)

Exercise 13.12

Using exercise 13.11, complete the proof of theorem 13.6. First prove the result for sets $\boldsymbol{Z}$ for which $\mathcal{U}_{\boldsymbol{Z}}$ contains only a single loop. Then prove the result for any $\boldsymbol{Z}$ for which $\mathcal{U}_{\boldsymbol{Z}}$ is a combination of disconnected trees and loops.

Exercise 13.13

Prove proposition 13.4.

Exercise 13.14

Show that the algorithm in algorithm 13.4 returns the correct MAP assignment. First show that for any cut $\mathcal{C} = \mathcal{Z}_s, \mathcal{Z}_t$, we have that

$$\textit{cost}(\mathcal{C}) = E(\xi^{\mathcal{C}}) + \text{Const.}$$

Conclude the desired result.

Exercise 13.15⋆

Show how the optimal alpha-beta swap step 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 the value of the t_i's defines an alpha-beta-swap transformation on the x_i's.
b. Define an energy function E' over the $\boldsymbol{T}$ variables such that $E'(\boldsymbol{t}) = E(\boldsymbol{x}')$.
c. Show that the energy function E' is submodular if the original energy function E is a semimetric.

Exercise 13.16⋆

truncation

As we discussed, many energy functions are not submodular. We now describe a method that allows min-cut methods to be applied to energy functions where most of the terms are submodular, but some small subset is not submodular. This method is based on the *truncation* of the nonsubmodular potentials, so as to make them submodular.

Algorithm 13.6 Efficient min-sum message passing for untruncated l-norm energies

```
    Procedure Msg-Truncated-l-Norm (
      c    // Parameters defining the pairwise factor
      h_i(x_i)    // Single-variable term in equation (13.36)
    )
1     for x_j = 1, ..., K − 1
2       r(x_j) ← min[h_i(x_j), r(x_j − 1) + c]
3     for x_j = K − 2, ..., 0
4       r(x_j) ← min[r(x_j), r(x_j + 1) + c]
5     return (r)
```

a. Let E be an energy function over binary-valued variables that contains some number of pairwise terms $\epsilon_{i,j}(v_i, v_j)$ that do not satisfy equation (13.33). Assume that we replace each such pairwise term $\epsilon_{i,j}$ with a term $\epsilon'_{i,j}$ that satisfies this inequality, by decreasing $\epsilon_{i,j}(0,0)$, by increasing $\epsilon_{i,j}(1,0)$ or $\epsilon_{i,j}(0,1)$, or both. The node energies remain unchanged. Let E' be the resulting energy. Show that if ξ^* optimizes E', then $E(\xi^*) \leq E(\mathbf{0})$

b. Describe how, in the multilabel case, this procedure can be used within the alpha-expansion algorithm to find a local optimum of the energy function.

Exercise 13.17⋆

Consider the task of passing a message over an edge X_i—X_j in a metric MRF; our goal is to make the message passing step more efficient by exploiting the metric structure. As usual in metric MRFs, we consider the problem in terms of energies; thus, our message computation takes the form:

$$\delta_{i \to j}(x_j) = \min_{x_i}(\epsilon_{i,j}(x_i, x_j) + h_i(x_i)), \tag{13.36}$$

where $h_i(x_i) = \epsilon_i(x_i) + \sum_{k \neq j} \delta_{i \to j}(x_k)$. In general, this computation requires $O(K^2)$ steps. However, we now consider two special cases where this computation can be done in $O(K)$ steps.

a. Assume that $\epsilon_{i,j}(x_i, x_j)$ is an Ising energy function, as in equation (4.6). Show how the message can be computed in $O(K)$ steps.

b. Now assume that both X_i, X_j take on values in $\{0, \ldots, K-1\}$. Assume that $\epsilon_{i,j}(x_i, x_j)$ is a nontruncated l-norm, as in equation (4.7) with $p = 1$ and $\text{dist}_{\max} = \infty$. Show that the algorithm in algorithm 13.6 computes the correct message in $O(K)$ steps.

c. Extend the algorithm of algorithm 13.6 to the case of a truncated l-norm (where $\text{dist}_{\max} < \infty$).

Exercise 13.18⋆

Consider the use of the branch-and-bound algorithm of appendix A.4.3 for finding the top K highest-probability assignments in an (unnormalized) distribution $\tilde{P}_\Phi$ defined by a set of factors Φ.

a. Consider a partial assignment $\boldsymbol{y}$ to some set of variables $\boldsymbol{Y}$. Provide both an upper and a lower bound to $\log \tilde{P}_\Phi(\boldsymbol{y})$.

b. Describe how to use your bounds in the context of a branch-and-bound algorithm to find the MAP assignment for $\tilde{P}_\Phi$. Can you use both the lower and upper bounds in your search?

c. Extend your algorithm to find the K highest probability joint assignments in $\tilde{P}_\Phi$. Hint: Your algorithm should find the assignments in order of decreasing probability, starting with the MAP. Be sure to reuse as much of your previous computations as possible as you continue the search for the next assignment.

Exercise 13.19

Show that, for any function f,

$$\max_x \sum_y f(x,y) \leq \sum_y \max_x f(x,y), \tag{13.37}$$

and provide necessary and sufficient conditions for when equation (13.37) holds as equality.

Exercise 13.20⋆

a. Use equation (13.37) to provide an efficient algorithm for computing an upper bound

$$\text{bound}(\boldsymbol{y}_{1\ldots i}) = \max_{y_{i+1},\ldots,y_n} \text{score}(\boldsymbol{y}_{1\ldots i}, y_{i+1}, \ldots, y_n),$$

where $\text{score}(\boldsymbol{y})$ is defined as in equation (13.35). Your computation of the bound should take no more than a run of variable elimination in an *unconstrained* elimination ordering over all of the network variables.

b. Use this bound to construct a branch-and-bound algorithm for the marginal-MAP problem.

Exercise 13.21⋆

In this question, we consider the application of conditioning to a marginal MAP query:

$$\arg\max_{\boldsymbol{Y}} \sum_{\boldsymbol{Z}} \prod_{\phi\in\Phi} \phi.$$

Let $\boldsymbol{U}$ be a set of conditioning variables.

a. Consider first the case of a simple MAP query, so that $\boldsymbol{Z} = \emptyset$ and $\boldsymbol{Y} = \mathcal{X}$. Show how you would adapt Conditioning in algorithm 9.5 to deal with the max-product rather than the sum-product task.

b. Now, consider a max-sum-product task. When is $\boldsymbol{U}$ a legal set of conditioning variables for this query? Justify your response. (Hint: Recall that the order of the operations we perform must respect the ordering constraint discussed in section 2.1.5, and that the elimination operations work from the outside in, and the conditioning operations from the inside out.)

c. Now, assuming that $\boldsymbol{U}$ is a legal set of conditioning variables, specify a conditioning algorithm that computes the value of the corresponding max-sum-product query, as in equation (13.8).

d. Extend your max-sum-product algorithm to compute the actual maximizing assignment to $\boldsymbol{Y}$, as in the MAP query. Your algorithm should work for any legal conditioning set $\boldsymbol{U}$.

14 *Inference in Hybrid Networks*

In our discussion of inference so far, we have focused on the case of discrete probabilistic models. However, many interesting domains also contain continuous variables such as temperature, location, or distance. In this chapter, we address the task of inference in graphical models that involve such variables.

For this chapter, let $\mathcal{X} = \Gamma \cup \Delta$, where Γ denotes the continuous variables and Δ the discrete variables. In cases where we wish to distinguish discrete and continuous variables, we use the convention that discrete variables are named with letters near the beginning of the alphabet (A, B, C), whereas continuous ones are named with letters near the end (X, Y, Z).

14.1 Introduction

14.1.1 Challenges

At an abstract level, the introduction of continuous variables in a graphical model is not difficult. As we saw in section 5.5, we can use a range of different representations for the CPDs or factors in our network. We now have a set of factors, over which we can perform the same operations that we utilize for inference in the discrete case: We can multiply factors, which in this case corresponds to multiplying the multidimensional continuous functions representing the factors; and we can marginalize out variables in a factor, which in this case is done using integration rather than summation. It is not difficult to show that, with these operations in hand, the sum-product inference algorithms that we used in the discrete case can be applied without change, and are guaranteed to lead to correct answers.

Unfortunately, a little more thought reveals that the correct implementation of these basic operations poses a range of challenges, whose solution is far from obvious.

The first challenge involves the representation of factors involving continuous variables. Unlike discrete variables, there is no universal representation of a factor over continuous variables, and so we must usually select a parametric family for each CPD or initial factor in our network. Even if we pick the same parametric family for each of our initial factor in the network, it may not be the case that multiplying factors or marginalizing a factor leaves it within the parametric family. If not, then it is not even clear how we would represent the intermediate results in our inference process. The situation becomes even more complex when factors in the original network call for the use of different parametric families. In this case, **it is generally unlikely that we can find a single parametric family that can correctly encode all of the intermediate factors**

in our network. In fact, in some cases — most notably networks involving both discrete and continuous variables — one can show that the intermediate factors cannot be represented using any fixed number of parameters; in fact, the representation size of those factors grows exponentially with the size of the network.

A second challenge involves the marginalization step, which now requires integration rather than summation. Integration introduces a new set of subtleties. First, not all functions are integrable: in some cases, the integral may be infinite or even ill defined. Second, even functions where the integral is well defined may not have a closed-form integral, requiring the use of a numerical integration method, which is usually approximate.

14.1.2 Discretization

discretization

An alternative approach to inference in hybrid models is to circumvent the entire problem of dealing with continuous factors: We simply convert all continuous variable to discrete ones by *discretizing* their domain into some finite set of intervals. Once all variables have been discretized, the result is a standard discrete probabilistic model, which we can handle using the standard inference techniques described in the preceding chapters.

How do we convert a hybrid CPDs into a table? Assume that we have a variable Y with a continuous parent X. Let A be the discrete variable that replaces X and B the discrete variable that replaces Y. Let $a \in Val(A)$ correspond to the interval $[x^1, x^2]$ for X, and $b \in Val(B)$ correspond to the interval $[y^1, y^2]$ for Y.

In principle, one approach for discretization is to define

$$P(b \mid a) = \int_{x^1}^{x^2} p(Y \in [y^1, y^2] \mid X = x) p(X = x \mid X \in [x^1, x^2]) dx.$$

This integral averages out the conditional probability that Y is in the interval $[y^1, y^2]$ given X, aggregating over the different values of x in the relevant interval for X. The distribution used in this formulation is the prior probability $p(X)$, which has the effect of weighting the average more toward more likely values of X. While plausible, this computation is expensive, since it requires that we perform inference in the model. Moreover, even if we perform our estimation relative to the prior $p(X)$, we have no guarantees of a good approximation, since our *posterior* over X may be quite different.

Therefore, for simplicity, we often use simpler approximations. In one, we simply select some particular value $x^* \in [x^1, x^2]$, and estimate $P(b \mid a)$ as the total probability mass of the interval $[y^1, y^2]$ given x^*: :

$$P(Y \in [y^1, y^2] \mid x^*) = \int_{y^1}^{y^2} p(y \mid x^*) dy.$$

For some density functions p, we can compute this interval in closed form. In others, we might have to resort to numerical integration methods. Alternatively, we can average the values over the interval $[x^1, x^2]$ using a predefined distribution, such as the uniform distribution over the interval.

Although discretization is used very often in practice, as we discussed in section 5.5, it has several significant limitations. The discretization is only an approximation of the true probability

distribution. In order to get answers that do not lose most of the information, our discretization scheme must have a fine resolution where the posterior probability mass lies. Unfortunately, before we actually perform the inference, we do not know the posterior distribution. Thus, we must often resort to a discretization that is fairly fine-grained over the entire space, leading to a very large domain for the resulting discrete variable.

This problem is particularly serious when we need to approximate distributions over more than a handful of discretized variables. As in any table-based CPD, the size of the resulting factor is exponential in the number of variables. When this number is large and the discretization is anything but trivial, the size of the factor can be huge. For example, if we need to represent a distribution over d continuous variables, each of which is discretized into m values, the total number of parameters required is $O(m^d)$. By contrast, if the distribution is a d-dimensional Gaussian, the number of parameters required is only $O(d^2)$. Thus, not only does the discretization process introduce approximations into our joint probability distribution, but we also often end up converting a polynomial parameterization into an exponential one.

Overall, discretization provides a trade-off between accuracy of the approximation and cost of computation. In certain cases, acceptable accuracy can be obtained at reasonable cost. However, in many practical applications, the computational cost required to obtain the accuracy needed for the task is prohibitive, making discretization a nonviable option.

14.1.3 Overview

Thus, we see that inference in continuous and hybrid models, although similar in principle to discrete inference, brings forth a new set of challenges. In this chapter, we discuss some of these issues and show how, in certain settings, these challenges can be addressed.

As for inference in discrete networks, the bulk of the work on inference in continuous and hybrid networks falls largely into two categories: Approaches that are based on message passing methods, and approaches that use one of the particle-based methods discussed in chapter 12. However, unlike the discrete case, even the message passing algorithms are rarely exact.

The message passing inference methods have largely revolved around the use of the Gaussian distribution. The easiest case is when the distribution is, in fact, a multivariate Gaussian. In this case, many of the challenges described before disappear. In particular, the intermediate factors in a Gaussian network can be described compactly using a simple parametric representation called the *canonical form*. This representation is closed under the basic operations using in inference: factor product, factor division, factor reduction, and marginalization. Thus, we can define a set of simple data structures that allow the inference process to be performed. Moreover, the integration operation required by marginalization is always well defined, and it is guaranteed to produce a finite integral under certain conditions; when it is well defined, it has a simple analytical solution.

As a consequence, a fairly straightforward modification of the discrete sum-product algorithms (whether variable elimination or clique tree) gives rise to an exact inference algorithm for Gaussian networks. A similar extension results in a Gaussian version of the loopy belief propagation algorithm; here, however, the conditions on integrability impose certain constraints about the form of the distribution. Importantly, under these conditions, loopy belief propagation for Gaussian distributions is guaranteed to return the correct means for the variables in the network, although it can underestimate the variances, leading to overconfident estimates.

There are two main extensions to the purely Gaussian case: non-Gaussian continuous densities, and hybrid models that involve both discrete and continuous variables. The most common method for dealing with these extensions is the same: we approximate intermediate factors in the computation as Gaussians; in effect, these algorithms are an instance of the expectation propagation algorithm discussed in section 11.4.4. As we discuss, Gaussians provide a good basis for the basic operations in these algorithms, including the key operation of approximate marginalization. Interestingly, there is one class of inference tasks where this general algorithm is guaranteed to produce exact answers to a certain subset of queries. This is the class of CLG networks in which we use a particular form of clique tree for the inference. Unfortunately, although of conceptual interest, this "exact" variant is rarely useful except in fairly small problems.

An alternative approach is to use an approximation method that makes no parametric assumptions about the distribution. Specifically, we can approximate the distribution as a set of particles, as described in chapter 12. As we will see, particle-based methods often provide the easiest approach to inference in a hybrid network. They make almost no assumptions about the form of the CPDs, and can approximate an arbitrarily complex posterior. Their primary disadvantage is, as usual, the fact that a very large number of particles might be required for a good approximation.

14.2 Variable Elimination in Gaussian Networks

The first class of networks we consider is the class of Gaussian networks, as described in chapter 7: These are networks where all of the variables are continuous, and all of the local factors encode linear dependencies. In the case of Bayesian networks, the CPDs take the form of linear Gaussians (as in definition 5.15). In the case of Markov networks, they can take the form of general log-quadratic form, as in equation (7.7).

As we showed in chapter 7, both of these representations are simply alternative parameterizations of a joint multivariate Gaussian distributions. This observation immediately suggests one approach to performing exact inference in this class of networks: We simply convert the LG network into the equivalent multivariate Gaussian, and perform the necessary operations — marginalization and conditioning — on that representation. Specifically, as we discussed, if we have a Gaussian distribution $p(\boldsymbol{X}, \boldsymbol{Y})$ represented as a mean vector and a covariance matrix, we can extract the marginal distribution $p(\boldsymbol{Y})$ simply by restricting attention to the elements of the mean and the covariance matrix that correspond to the variables in $\boldsymbol{Y}$. The operation of conditioning a Gaussian on evidence $\boldsymbol{Y} = \boldsymbol{y}$ is also easy: we simply instantiate the variables $\boldsymbol{Y}$ to their observed values $\boldsymbol{y}$ in the joint density function, and renormalize the resulting unnormalized measure over $\boldsymbol{X}$ to obtain a new Gaussian density.

This approach simply generates the joint distribution over the entire set of variables in the network, and then manipulates it directly. However, unlike the case of discrete distributions, the representation size of the joint density in the Gaussian case is quadratic, rather than exponential, in the number of variables. Thus, these operations are often feasible in a Gaussian network in cases that would not be feasible in a comparable discrete network.

Still, even quadratic cost might not be feasible in many cases, for example, when the network is over thousands of variables. Furthermore, this approach does not exploit any of the structure represented in the network distribution. An alternative approach to inference is to adapt the

message passing algorithms, such as variable elimination (or clique trees) for exact inference, or belief propagation for approximation inference, to the linear Gaussian case. We now describe this approach. We begin with describing the basic representation used for these message passing schemes, and then present these two classes of algorithms.

14.2.1 Canonical Forms

As we discussed, the key difference between inference in the continuous and the discrete case is that the factors can no longer be represented as tables. Naively, we might think that we can represent factors as Gaussians, but this is not the case. The reason is that linear Gaussian CPDs are generally not Gaussians, but are rather a conditional distribution. Thus, we need to find a more general representation for factors, that accommodates both Gaussian distributions and linear Gaussian models, as well as any combination of these models that might arise during the course of inference.

14.2.1.1 The Canonical Form Representation

The simplest representation used in this setting is the *canonical form*, which represents the intermediate result as a log-quadratic form $\exp(Q(\boldsymbol{x}))$ where Q is some quadratic function. In the inference setting, it is useful to make the components of this representation more explicit:

Definition 14.1 canonical form

A canonical form $\mathcal{C}(\boldsymbol{X}; K, \boldsymbol{h}, g)$ *(or* $\mathcal{C}(K, \boldsymbol{h}, g)$ *if we omit the explicit reference to* $\boldsymbol{X}$*) is defined as:*

$$\mathcal{C}(\boldsymbol{X}; K, \boldsymbol{h}, g) = \exp\left(-\frac{1}{2}\boldsymbol{X}^T K \boldsymbol{X} + \boldsymbol{h}^T \boldsymbol{X} + g\right). \tag{14.1}$$

■

We can represent every Gaussian as a canonical form. Rewriting equation (7.1), we obtain:

$$\begin{aligned} &\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \\ &\quad = \exp\left(-\frac{1}{2}\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu} - \log\left((2\pi)^{n/2}|\Sigma|^{1/2}\right)\right). \end{aligned}$$

Thus, $\mathcal{N}(\boldsymbol{\mu}; \Sigma) = \mathcal{C}(K, \boldsymbol{h}, g)$ where:

$$\begin{aligned} K &= \Sigma^{-1} \\ \boldsymbol{h} &= \Sigma^{-1}\boldsymbol{\mu} \\ g &= -\frac{1}{2}\boldsymbol{\mu}^T\Sigma^{-1}\boldsymbol{\mu} - \log\left((2\pi)^{n/2}|\Sigma|^{1/2}\right). \end{aligned}$$

However, canonical forms are more general than Gaussians: If K is not invertible, the canonical form is well defined, but it is not the inverse of a legal covariance matrix. In particular, we can easily represent linear Gaussian CPDs as canonical forms (exercise 14.1).

14.2.1.2 Operations on Canonical Forms

canonical form product

It is possible to perform various operations on canonical forms. The *product* of two canonical form factors over the same scope $\boldsymbol{X}$ is simply:

$$\mathcal{C}(K_1, \boldsymbol{h}_1, g_1) \cdot \mathcal{C}(K_2, \boldsymbol{h}_2, g_2) = \mathcal{C}(K_1 + K_2, \boldsymbol{h}_1 + \boldsymbol{h}_2, g_1 + g_2). \tag{14.2}$$

When we have two canonical factors over different scopes $\boldsymbol{X}$ and $\boldsymbol{Y}$, we simply extend the scope of both to make their scopes match and then perform the operation of equation (14.2). The extension of the scope is performed by simply adding zero entries to both the K matrices and the $\boldsymbol{h}$ vectors.

Example 14.1

Consider the following two canonical forms:

$$\begin{aligned}\phi_1(X,Y) &= \mathcal{C}\left(X, Y; \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, -3\right) \\ \phi_2(Y,Z) &= \mathcal{C}\left(Y, Z; \begin{bmatrix} 3 & -2 \\ -2 & 4 \end{bmatrix}, \begin{pmatrix} 5 \\ -1 \end{pmatrix}, 1\right).\end{aligned}$$

We can extend the scope of both of these by simply introducing zeros into the canonical form. For example, we can reformulate:

$$\phi_1(X,Y,Z) = \mathcal{C}\left(X, Y, Z; \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}, -3\right),$$

and similarly for $\phi_2(X,Y,Z)$. The two canonical forms now have the same scope, and can be multiplied using equation (14.2) to produce:

$$\mathcal{C}\left(X, Y, Z; \begin{bmatrix} 1 & -1 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}, \begin{pmatrix} 1 \\ 4 \\ -1 \end{pmatrix}, -2\right).$$

■

canonical form division

The *division* of canonical forms (which is required for message passing in the belief propagation algorithm) is defined analogously:

$$\frac{\mathcal{C}(K_1, \boldsymbol{h}_1, g_1)}{\mathcal{C}(K_2, \boldsymbol{h}_2, g_2)} = \mathcal{C}(K_1 - K_2, \boldsymbol{h}_1 - \boldsymbol{h}_2, g_1 - g_2). \tag{14.3}$$

vacuous canonical form

Note that the *vacuous canonical form*, which is the analogue of the "all 1" factor in the discrete case, is defined as $K = 0$, $\boldsymbol{h} = \mathbf{0}$, $g = 0$. Multiplying or dividing by this factor has no effect.

canonical form marginalization

Less obviously, we can *marginalize* a canonical form onto a subset of its variables. Let $\mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K, \boldsymbol{h}, g)$ be some canonical form over $\{\boldsymbol{X}, \boldsymbol{Y}\}$ where

$$K = \begin{bmatrix} K_{\boldsymbol{XX}} & K_{\boldsymbol{XY}} \\ K_{\boldsymbol{YX}} & K_{\boldsymbol{YY}} \end{bmatrix} \quad ; \quad h = \begin{pmatrix} h_{\boldsymbol{X}} \\ h_{\boldsymbol{Y}} \end{pmatrix}. \tag{14.4}$$

The marginalization of this function onto the variables $\boldsymbol{X}$ is, as usual, the integral over the variables $\boldsymbol{Y}$:

$$\int \mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K, \boldsymbol{h}, g)\, d\boldsymbol{Y}.$$

As we discussed, we have to guarantee that all of the integrals resulting from marginalization operations are well defined. In the case of canonical forms, the integral is finite if and only if $K_{\boldsymbol{YY}}$ is positive definite, or equivalently, that it is the inverse of a legal covariance matrix. In this case, the result of the integration operation is a canonical form $\mathcal{C}\left(\boldsymbol{X}; K', \boldsymbol{h}', g'\right)$ given by:

$$\begin{aligned} K' &= K_{\boldsymbol{XX}} - K_{\boldsymbol{XY}} K_{\boldsymbol{YY}}^{-1} K_{\boldsymbol{YX}} \\ h' &= \boldsymbol{h}_{\boldsymbol{X}} - K_{\boldsymbol{XY}} K_{\boldsymbol{YY}}^{-1} \boldsymbol{h}_{\boldsymbol{Y}} \\ g' &= g + \tfrac{1}{2}\left(\log|2\pi K_{\boldsymbol{YY}}^{-1}| + \boldsymbol{h}_{\boldsymbol{Y}}^T K_{\boldsymbol{YY}}^{-1} \boldsymbol{h}_{\boldsymbol{Y}}\right). \end{aligned} \tag{14.5}$$

canonical form reduction

Finally, it is possible to *reduce* a canonical form to a context representing evidence. Assume that the canonical form $\mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K, \boldsymbol{h}, g)$ is given by equation (14.4). Then setting $\boldsymbol{Y} = \boldsymbol{y}$ results in the canonical form $\mathcal{C}\left(\boldsymbol{X}; K', \boldsymbol{h}', g'\right)$ given by:

$$\begin{aligned} K' &= K_{\boldsymbol{XX}} \\ h' &= \boldsymbol{h}_{\boldsymbol{X}} - K_{\boldsymbol{XY}} \boldsymbol{y} \\ g' &= g + \boldsymbol{h}_{\boldsymbol{Y}}^T \boldsymbol{y} - \frac{1}{2} \boldsymbol{y}^T K_{\boldsymbol{YY}} \boldsymbol{y}. \end{aligned} \tag{14.6}$$

See exercise 14.3.

Importantly, all of the factor operations can be done in time that is polynomial in the scope of the factor. In particular, the product or division of factors requires quadratic time; factor marginalization, which requires matrix inversion, can be done naively in cubic time, and more efficiently using advanced methods.

14.2.2 Sum-Product Algorithms

sum-product

The operations described earlier are the basic building blocks for all of our *sum-product* exact inference algorithms: variable elimination and both types of clique tree algorithms. Thus, we can adapt these algorithms to apply to linear Gaussian networks, using canonical forms as our representation of factors. For example, in the Sum-Product-VE algorithm of algorithm 9.1, we simply implement the factor product operation as in equation (14.2), and replace the summation operation in Sum-Product-Eliminate-Var with an integration operation, implemented as in equation (14.5).

Care must be taken regarding the treatment of evidence. In discrete factors, when instantiating a variable $Z = z$, we could leave the variable Z in the factors involving it, simply zeroing the entries that are not consistent with $Z = z$. In the case of continuous variables, our representation of factors does not allow that option: when we instantiate $Z = z$, the variable Z is no longer part of the canonical form. Thus, it is necessary to reduce all the factors participating in the inference process to a scope that no longer contains Z. This reduction step is already part of the variable elimination algorithm of algorithm 9.2. It is straightforward to ensure that the clique tree algorithms of chapter 10 similarly reduce all clique and sepset potentials with the evidence prior to any message passing steps.

A more important problem that we must consider is that the marginalization operation may not be well defined for an arbitrary canonical form. In order to show the correctness of an inference algorithm, we must show that it executes a marginalization step only on canonical forms for which this operation is *well defined*. We prove this result in the context of the sum-

well-defined marginalization

product clique tree algorithm; the proof for the other cases follows in a straightforward way, due to the equivalence between the upward pass of the different message passing algorithms.

Proposition 14.1 *Whenever* SP-Message *is called, within the* CTree-SP-Upward *algorithm (algorithm 10.1) the marginalization operation is well-defined.*

PROOF Consider a call SP-Message(i, j), and let $\psi(\boldsymbol{C}_i)$ be the factor constructed in the clique prior to sending the message. Let $\mathcal{C}(\boldsymbol{C}_i; K, \boldsymbol{h}, g)$ be the canonical form associated with $\psi(\boldsymbol{C}_i)$. Let $\beta_i(\boldsymbol{C}_i) = \mathcal{C}(\boldsymbol{C}_i; K', \boldsymbol{h}', g')$ be the final clique potential that would be obtained at $\boldsymbol{C}_i$ in the case where $\boldsymbol{C}_i$ is the root of the clique tree computation. The only difference between these two potentials is that the latter also incorporates the message $\delta_{j \to i}$ from $\boldsymbol{C}_j$.

Let $\boldsymbol{Y} = \boldsymbol{C}_j - \boldsymbol{S}_{i,j}$ be the variables that are marginalized when the message is computed. By the running intersection property, none of the variables $\boldsymbol{Y}$ appear in the scope of the sepset $\boldsymbol{S}_{i,j}$. Thus, the message $\delta_{j \to i}$ does not mention any of the variables $\boldsymbol{Y}$. We can verify, by examining equation (14.2), that multiplying a canonical form by a factor that does not mention $\boldsymbol{Y}$ does not change the entries in the matrix K that are associated with the variables in $\boldsymbol{Y}$. It follows that $K_{\boldsymbol{Y}\boldsymbol{Y}} = K'_{\boldsymbol{Y}\boldsymbol{Y}}$, that is, the submatrices for $\boldsymbol{Y}$ in K and K' are the same. Because the final clique potential $\beta_i(\boldsymbol{C}_i)$ is its (unnormalized) marginal posterior, it is a normalizable Gaussian distribution, and hence the matrix K' is positive definite. As a consequence, the submatrix $K'_{\boldsymbol{Y}\boldsymbol{Y}}$ is also positive definite. It follows that $K_{\boldsymbol{Y}\boldsymbol{Y}}$ is positive definite, and therefore the marginalization operation is well defined. ■

It follows that we can adapt any of our exact inference algorithms to the case of linear Gaussian networks. The algorithms are essentially unchanged; only the representation of factors and the implementation of the basic factor operations are different. In particular, since all factor operations can be done in polynomial time, inference in linear Gaussian networks is linear in the number of cliques, and at most cubic in the size of the largest clique. By comparison, recall that the representation of table factors is, by itself, exponential in the scope, leading to the exponential complexity of inference in discrete networks.

It is interesting to compare the clique tree inference algorithm to the naive approach of simply generating the joint Gaussian distribution and marginalizing it. The exact inference algorithm requires multiple steps, each of which involves matrix product and inversion. By comparison, the joint distribution can be computed, as discussed in theorem 7.3, by a set of vector-matrix products, and the marginalization of a joint Gaussian over any subset of variables is trivial (as in lemma 7.1). Thus, in cases where the Gaussian has sufficiently low dimension, it may be less computationally intensive to use the naive approach for inference. Conversely, in cases where the distribution has high dimension and the network has reasonably low tree-width, the message passing algorithms can offer considerable savings.

14.2.3 Gaussian Belief Propagation

Gaussian belief propagation

The *Gaussian belief propagation* algorithm utilizes the information form, or canonical form, of the Gaussian distribution. As we discussed, a Gaussian network is encoded using a set of local quadratic potentials, as in equation (14.1). Reducing a canonical-form factor on evidence also results in a canonical-form factor (as in equation (14.6)), and so we can focus attention on a

representation that consists of a product of canonical-form factors. This product results in an overall quadratic form:

$$p(X_1, \ldots, X_n) \propto \exp\left(-\frac{1}{2}\boldsymbol{X}^T J \boldsymbol{X} + \boldsymbol{h}^T \boldsymbol{X}\right).$$

The measure p is normalizable and defines a legal Gaussian distribution if and only if J is positive definite. Note that we can obtain J by adding together the individual matrices K defined by the various potentials parameterizing the network.

In order to apply the belief propagation algorithm, we must define a cluster graph and assign the components of this parameterization to the different clusters in the graph. As in any belief propagation algorithm, we need the cluster graph to respect the family preservation property. In our setting, the only terms in the quadratic form involve single variables — the $h_i X_i$ terms — and pairs of variables X_i, X_j for which $J_{ij} \neq 0$. Thus, the minimal cluster graph that satisfies the family preservation requirement would contain a cluster for each edge X_i, X_j (pairs for which $J_{ij} \neq 0$). We choose to use a Bethe-structured cluster graph that has a cluster for each variable X_i and a cluster for each edge X_i, X_j. While it is certainly possible to define a belief propagation algorithm on a cluster graph with larger cliques, the standard application runs belief propagation directly on this pairwise network.

We note that the parameterization of the cluster graph is not uniquely defined. In particular, a term of the form $J_{ii}X_i^2$ can be partitioned in infinitely many ways among the node's own cluster and among the edges that contain X_i. Each of these partitions defines a different set of potentials in the cluster graph, and hence will induce a different execution of belief propagation. We describe the algorithm in terms of the simplest partition, where each diagonal term J_{ii} is assigned to the corresponding X_i cluster, and the off-diagonal terms J_{ij} are assigned to the X_i, X_j cluster.

With this decision, the belief propagation algorithm for Gaussian networks is simply derived from the standard message passing operations, implemented with the canonical-form operations for the factor product and marginalization steps. For concreteness, we now provide the precise message passing steps. The message from X_i to X_j has the form

$$\delta_{i \to j}(x_j) = \exp\left(-\frac{1}{2}J_{i \to j}x_j^2 + h_{i \to j}x_j\right). \tag{14.7}$$

We compute the coefficients in this expression via a two-stage process. The first step corresponds to the message sent from the X_i cluster to the X_i, X_j edge; in this step, X_i aggregates all of the information from its own local potential and the messages sent from its other incident edges:

$$\begin{array}{lcl} \hat{J}_{i \setminus j} & = & J_{ii} + \sum_{k \in \mathrm{Nb}_i - \{j\}} J_{k \to i} \\ \hat{h}_{i \setminus j} & = & h_i + \sum_{k \in \mathrm{Nb}_i - \{j\}} h_{k \to i}. \end{array} \tag{14.8}$$

In the second step, the X_i, X_j edge takes the message received from X_i and sends the appropriate message to X_j. The form of the message can be computed (with some algebraic manipulation) from the formulas for the conditional mean and conditional variance, that we used in theorem 7.4, giving rise to the following update equations:

$$\begin{array}{lcl} J_{i \to j} & = & -J_{ji}\hat{J}_{i \setminus j}^{-1} J_{ji} \\ h_{i \to j} & = & -J_{ji}\hat{J}_{i \setminus j}^{-1} \hat{h}_{i \setminus j}. \end{array} \tag{14.9}$$

These messages can be scheduled in various ways, either synchronously or asynchronously (see box 11.B).

If and when the message passing process has converged, we can compute the X_i-entries of the information form by combining the messages in the usual way:

$$\begin{aligned}\hat{J}_i &= J_{ii} + \sum_{k \in \mathrm{Nb}_i} J_{k \to i} \\ \hat{h}_i &= h_i + \sum_{k \in \mathrm{Nb}_i} h_{k \to i}.\end{aligned}$$

From this information-form representation, X_i's approximate mean $\hat{\mu}_i$ and covariance $\hat{\sigma}_i^2$ can be reconstructed as usual:

$$\begin{aligned}\hat{\mu}_i &= (\hat{J}_i)^{-1} \hat{h}_i \\ \hat{\sigma}_i^2 &= (\hat{J}_i)^{-1}\end{aligned}$$

One can now show the following result, whose proof we omit:

Theorem 14.1 *Let $\hat{\mu}_i, \hat{\sigma}_i^2$ be a set of fixed points of the message passing process defined in equation (14.8), equation (14.9). Then $\hat{\mu}_i$ is the correct posterior mean for the variable X_i in the distribution P.*

☞ **Thus, if the BP message passing process converges, the resulting beliefs encode the correct mean of the joint distribution. The estimated variances $\hat{\sigma}_i^2$ are generally not correct; rather, they are an underestimate of the true variances, so that the resulting posteriors are "overconfident."**

This correctness result is predicated on convergence. In general, this message passing process may or may not converge. Moreover, their convergence may depend on the order in which messages are sent. However, unlike the discrete case, one can provide a very detailed characterization of the convergence properties of this process, as well as sufficient conditions for convergence (see section 14.7 for some references). In particular, one can show that the pairwise normalizability condition, as in definition 7.3, suffices to guarantee the convergence of the belief propagation algorithm for any order of messages. Recall that this condition guarantees that each edge can be associated with a potential that is a normalized Gaussian distribution. As a consequence, when the Gaussian parameterizing the edge is multiplied with Gaussians encoding the incoming message, the result is also a well-normalized Gaussian.

We note that pairwise normalizability is sufficient, but not necessary, for the convergence of belief propagation.

Example 14.2 *Consider the Gaussian MRF shown in figure 14.1. This model defines a frustrated loop, since three of the edges in the loop are driving X_1, X_2 toward a positive correlation, but the edge between them is driving in the opposite direction. The larger the value of r, the worse the frustration. This model is diagonally dominant for any value of $r < 1/3$. It is pairwise normalizable for any $r < 0.39030$; however, it defines a valid Gaussian distribution for values of r up to 0.5.* ■

In practice, the Gaussian belief propagation algorithm often converges and provides an excellent alternative for reasoning in Gaussian distributions that are too large for exact techniques.

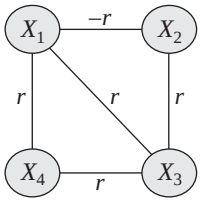


Figure 14.1 A Gaussian MRF used to illustrate convergence properties of Gaussian belief propagation. In this model, $J_{ii} = 1$, and $J_{ij} = r$ for all edges (i, j), except for $J_{12} = -r$.

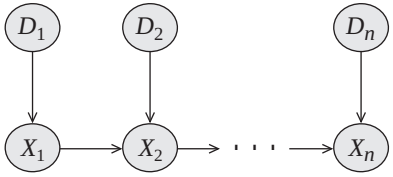


Figure 14.2 Simple CLG network used to demonstrate hardness of inference. $D_1, \ldots, D_n$ are discrete, and $X_1, \ldots, X_n$ are continuous.

14.3 Hybrid Networks

So far, we have dealt with models that involve only continuous variables. We now begin our discussion of hybrid networks — those that include both continuous and discrete variables. We focus the bulk of our discussion on conditional linear Gaussian (CLG) networks (definition 5.16), where there are no discrete variables with continuous parents, and where all the local probability models of continuous variables are conditional linear Gaussian CPDs. In the next section, we discuss inference for non-Gaussian dependencies, which will allow us to deal with non-CLG dependencies.

Even for this restricted class of networks, we can show that inference is very challenging. Indeed, we can show that inference in this class of networks is $\mathcal{NP}$-hard, even when the network structure is a polytree. We then show how the expectation propagation approach described in the previous section can be applied in this setting. Somewhat surprisingly, we show that this approach provides "exact" results in certain cases, albeit mostly ones of theoretical interest.

14.3.1 The Difficulties

As we discussed earlier, at an abstract level, variable elimination algorithms are all the same: They perform operations over factors to produce new factors. In the case of discrete models, factors can be represented as tables, and these operations can be performed effectively as table operations. In the case of Gaussian networks, the factors can be represented as canonical forms. As we now show, in the hybrid case, the representation of the intermediate factors can grow arbitrarily complex.

Consider the simple network shown in figure 14.2, where we assume that each D_i is a discrete binary variable, X_1 is a conditional Gaussian, and each X_i for $i > 1$ is a conditional linear

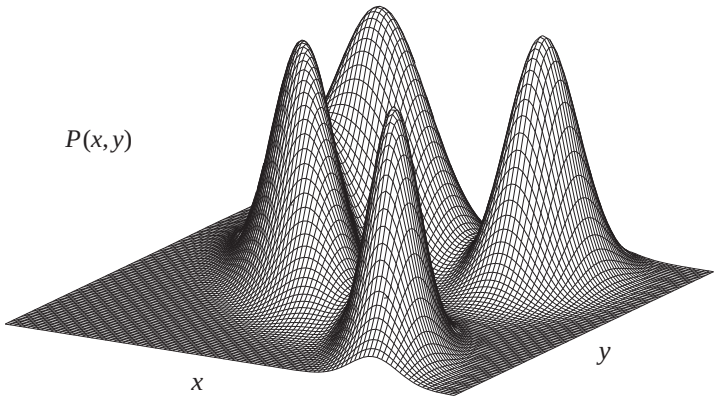


Figure 14.3 Joint marginal distribution $p(X_1, X_2)$ for a network as in figure 14.2

Gaussian (CLG) (see definition 5.15):

$$p(X_i \mid X_{i-1}, D_i) = \mathcal{N}\left(X_i \mid \alpha_{i,d_i} x_{i-1} + \beta_{i,d_i}; \sigma^2_{i,d_i}\right),$$

where for simplicity we take $\alpha_{1,d_1} = 0$, so the same formula applies for all i.

Assume that our goal is to compute $P(X_n)$. To do so, we marginalize the joint distribution:

$$p(X_n) = \sum_{D_1,\ldots,D_n} \int p(D_1, \ldots, D_n, X_1, \ldots, X_n) dX_1 \ldots dX_{n-1}.$$

Using the chain rule, the joint distribution is defined as:

$$p(D_1, \ldots, D_n, X_1, \ldots, X_n) = \prod_{i=1}^{n} P(D_i) p(X_1 \mid D_1) \prod_{i=2}^{n} p(X_i \mid D_i, X_{i-1}).$$

We can reorder the sums and integrals and push each of them in over factors that do not involve the variable to be marginalized. Thus, for example, we have that:

$$p(X_2) = \sum_{D_2} P(D_2) \int p(X_2 \mid X_1, D_2) \sum_{D_1} p(X_1 \mid D_1) P(D_1) dX_1.$$

Using the same variable elimination approach that we used in the discrete case, we first generate a factor over X_1 by multiplying $P(D_1)p(X_1 \mid D_1)$ and summing out D_1. This factor is then multiplied with $p(X_2 \mid X_1, D_2)$ to generate a factor over X_1, X_2, D_2. We can now eliminate X_1 by integrating the function that corresponds to this factor, to generate a factor over X_2, D_2. We can now sum out D_2 to get $p(X_2)$. The process of computing $p(X_n)$ is analogous.

Now, consider the marginal distribution $p(X_i)$ for $i = 1, \ldots, n$. For $i = 1$, this distribution is a mixture of two Gaussians, one corresponding to the value $D_1 = d_1^1$ and the other to $D_1 = d_1^0$. For $i = 2$, let us first consider the distribution $p(X_1, X_2)$. This distribution is a mixture of four

Gaussians, for the four different instantiations of D_1, D_2. For example, assume that we have:

$$\begin{aligned} p(X_1 \mid d_1^0) &= \mathcal{N}\left(0; 0.7^2\right) \\ p(X_1 \mid d_1^1) &= \mathcal{N}\left(1.5; 0.6^2\right) \\ p(X_2 \mid X_1, d_2^0) &= \mathcal{N}\left(-1.5X_1; 0.6^2\right) \\ p(X_2 \mid X_1, d_2^1) &= \mathcal{N}\left(1.5; 0.7^2\right). \end{aligned}$$

The joint marginal distribution $p(X_1, X_2)$ is shown in figure 14.3. Note that the mixture contains two components where X_1 and X_2 are independent; these components correspond to the instantiations where $D_2 = d_2^1$, in which we have $\alpha_{2,1} = 0$. As shown in lemma 7.1, the marginal distribution of a Gaussian is also a Gaussian, and the same applies to a mixture. Hence the marginal distribution $p(X_2)$ is also a mixture of four Gaussians. We can easily extend this argument, showing that $p(X_i)$ is a mixture of 2^i Gaussians. **In general, even representing the correct marginal distribution in a hybrid network can require space that is exponential in the size of network.**

Indeed, this type of example can be used as the basis for proving a result about the hardness of inference in models of this type. Clearly, as CLG networks subsume standard discrete networks, exact inference in such networks is necessarily $\mathcal{NP}$-hard. More surprising, however, is the fact that this task is $\mathcal{NP}$-hard even in very simple network structures such as polytrees. In fact, the problem of computing the probability of a single discrete variable, or even approximating this probability with any absolute error strictly less than $1/2$, is $\mathcal{NP}$-hard.

To define the problem precisely, assume we are working with finite precision continuous variables. We define the following decision problem *CLG-DP*.

Input: A CLG Bayesian network $\mathcal{B}$ over $\Delta \cup \Gamma$, evidence $\boldsymbol{E} = \boldsymbol{e}$, and a discrete variable $A \in \Delta$.

Output: "Yes" if $P_{\mathcal{B}}(A = a^1 \mid \boldsymbol{E} = \boldsymbol{e}) > 0.5$.

Theorem 14.2 *The problem CLG-DP is $\mathcal{NP}$-hard even if $\mathcal{B}$ is a polytree.*

The fact that exact inference in polytree CLGs is $\mathcal{NP}$-hard may not be very surprising by itself. After all, the distribution of a continuous variable in a CLG distribution, even in a simple polytree, can be a mixture of exponentially many Gaussians. Therefore, it might be expected that tasks that require that we reason directly with such a distribution are hard. Somewhat more surprisingly, this phenomenon arises even in networks where the prior distribution of every continuous variable is a mixture of at most two Gaussians.

Theorem 14.3 *The problem CLG-DP is $\mathcal{NP}$-hard even if $\mathcal{B}$ is a polytree where all of the discrete variables are binary-valued, and where every continuous variable has at most one discrete ancestor.*

Intuitively, this proof relies on the use of activated v-structures to introduce, in the posterior, dependencies where a continuous variable can have exponentially many modes.

Overall, these results show that even the easiest approximate inference task — inference over a binary-valued variable that achieves absolute error less than 0.5 — is intractable in CLG networks. This fact implies that one should not expect to find a polynomial-time approximate inference algorithm with a useful error bound without further restrictions on the structure or the parameters of the CLGs.

14.3.2 Factor Operations for Hybrid Gaussian Networks

Despite the discouraging results in the previous section, one can try to produce useful algorithms for hybrid networks in order to construct an approximate inference algorithm that has good performance, at least in practice. We now present the basic factor operations required for message passing or variable elimination in hybrid networks. In subsequent sections, we describe two algorithms that use these operations for inference.

14.3.2.1 Canonical Tables

As we discussed, the key decision in adapting an exact inference algorithm to a class of hybrid networks is the representation of the factors involved in the process. In section 14.2, when doing inference for linear Gaussian networks, we used canonical forms to represent factors. This representation is rich enough to capture both Gaussians and linear Gaussian CPDs, as well as all of the intermediate expressions that arise during the course of inference. In the case of CLG networks, we must contend with discrete variables as well as continuous ones. In particular, a CLG CPD has a linear Gaussian model for each instantiation of the discrete parents.

Extending on the canonical form, we can represent this CPD as a table, with one entry for each instantiation of the discrete variables, each associated with a canonical form over the continuous ones:

Definition 14.2 canonical table

A canonical table ϕ *over* $\boldsymbol{D}, \boldsymbol{X}$ *for* $\boldsymbol{D} \subseteq \Delta$ *and* $\boldsymbol{X} \subseteq \Gamma$ *is a table with an entry for each* $\boldsymbol{d} \in Val(\boldsymbol{D})$*, where each entry contains a canonical form* $\mathcal{C}(\boldsymbol{X}; K_{\boldsymbol{d}}, \boldsymbol{h}_{\boldsymbol{d}}, g_{\boldsymbol{d}})$ *over* $\boldsymbol{X}$*. We use* $\phi(\boldsymbol{d})$ *to denote the canonical form over* $\boldsymbol{X}$ *associated with the instantiation* $\boldsymbol{d}$*.* ■

The canonical table representation subsumes both the canonical form and the table factors used in the context of discrete networks. For the former, $\boldsymbol{D} = \emptyset$, so we have only a single canonical form over $\boldsymbol{X}$. For the latter, $\boldsymbol{X} = \emptyset$, so that the parameters $K_{\boldsymbol{d}}$ and $\boldsymbol{h}_{\boldsymbol{d}}$ are vacuous, and we remain with a canonical form $\exp(g_{\boldsymbol{d}})$ for each entry $\phi(\boldsymbol{d})$. Clearly, a standard table factor $\phi(\boldsymbol{D})$ can be reformulated in this way by simply taking $g_{\boldsymbol{d}} = \ln(\phi(\boldsymbol{d}))$. Therefore, we can represent any of the original CPDs or Gaussian potentials in a CLG network as a canonical table.

Now, consider the operations on factors used by the various exact inference algorithms. Let us first consider the operations of factor product and factor division. As for the table representation of discrete factors, these operations are performed between corresponding table entries, in the usual way. The product or division operations for the individual entries are performed over the associated canonical forms, as specified in equation (14.2) and equation (14.3).

Example 14.3

Assume we have two factors $\phi_1(A, B, X, Y)$ *and* $\phi_2(B, C, Y, Z)$*, which we want to multiply in order to produce* $\tau(A, B, C, X, Y, Z)$*. The resulting factor* τ *has an entry for each instantiation of the discrete variables* A, B, C*. The entry for a particular instantiation* a, b, c *is a canonical form, which is derived as the product of the two canonical forms: the one associated with* a, b *in* ϕ_1 *and the one associated with* b, c *in* ϕ_2*. The product operation for two canonical forms of different scopes is illustrated in example 14.1.* ■

Similarly, reducing a canonical table with evidence is straightforward. Let $\{\boldsymbol{d}, \boldsymbol{x}\}$ be a set of observations (where $\boldsymbol{d}$ is discrete and $\boldsymbol{x}$ is continuous). We instantiate $\boldsymbol{d}$ in a canonical table by

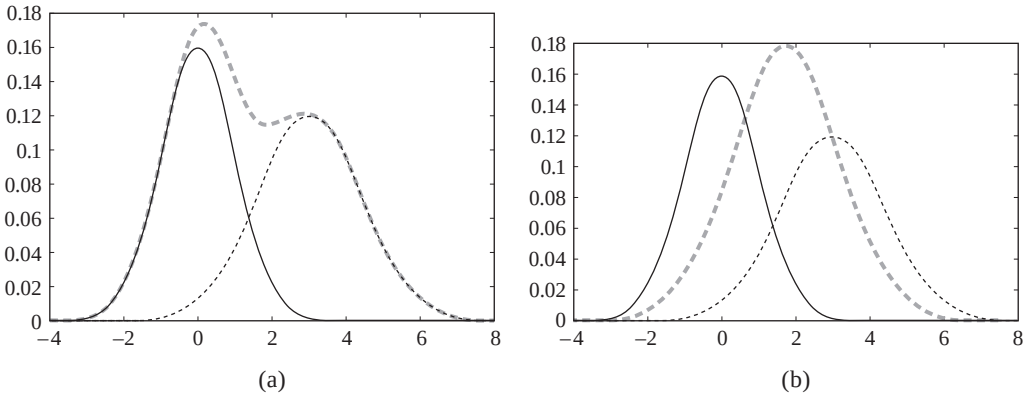


Figure 14.4 Summing and collapsing a Gaussian mixture. (a) Two Gaussian measures and the measure resulting from summing them. (b) The measure resulting from collapsing the same two measures into a single Gaussian.

setting the entries which are not consistent with $\boldsymbol{d}$ to zero. We instantiate $\boldsymbol{x}$ by instantiating every canonical form with this evidence, as in equation (14.6).

Finally, consider the marginalization operation. Here, we have two very different cases: integrating out a continuous variable and summing out a discrete one. The operation of continuous marginalization (integration of of a continuous variable) is straightforward: we simply apply the operation of equation (14.5) to each of the canonical forms in our table. More precisely, assume that our canonical table consists of a set of canonical forms $\mathcal{C}(\boldsymbol{X}, \boldsymbol{Y}; K_{\boldsymbol{d}}, \boldsymbol{h}_{\boldsymbol{d}}, g_{\boldsymbol{d}})$, indexed by $\boldsymbol{d}$. Now, for each $\boldsymbol{d}$ separately, we can integrate out $\boldsymbol{Y}$ in the appropriate canonical form, as in equation (14.5). This results in a new canonical table $\mathcal{C}\left(\boldsymbol{X}; K'_{\boldsymbol{d}}, \boldsymbol{h}'_{\boldsymbol{d}}, g'_{\boldsymbol{d}}\right)$, indexed by $\boldsymbol{d}$. Clearly, this has the desired effect: Before the operation, we have a mixture, where each mixture is a function over a set of variables $\boldsymbol{X}, \boldsymbol{Y}$; after the operation, we have a mixture with the same set of components, but now each component only represents the function over the variables $\boldsymbol{X}$. The only important restriction, as we noted in the derivation of equation (14.5), that each of the matrices $K_{\boldsymbol{d},\boldsymbol{YY}}$ be positive definite, so that the integral is well defined.

14.3.2.2 Weak Marginalization

The task of discrete marginalization, however, is significantly more complex. To understand the difficulty, consider the following example:

Example 14.4

Assume that we have a canonical form $\phi(A, X)$, for a binary-valued variable A and a continuous variable X. Furthermore, assume that the two canonical forms in the table (associated with a^0 and a^1) are both weighted Gaussians:

$$\begin{aligned}\phi(a^0) &= 0.4 \times \mathcal{N}(X \mid 0; 1) \\ \phi(a^1) &= 0.6 \times \mathcal{N}(X \mid 3; 4).\end{aligned}$$

Figure 14.4a shows the two canonical forms, as well as the marginal distribution over X. Clearly, this distribution is not a Gaussian; in fact, it cannot be represented at all as a canonical form. ■

We see that the family of canonical tables is not closed under discrete marginalization: this operation takes a canonical table and produces something that is not representable in this form.

We now have two alternatives. The first is to enrich the family that we use for our representation of factors. Specifically, we would have to use a table where, for each instantiations of the discrete variables, we have a *mixture* of canonical forms. In this case, the discrete marginalization operation is trivial: In example 14.4, we would have a single entry that contains the marginal distribution shown in figure 14.4a. While this family is closed under discrete marginalization, we have only resurrected our original problem: As discrete variables are "eliminated," they simply induce more components in the mixture of canonical forms; the end result of this process is simply the original exponentially large mixture that we were trying to avoid.

The second alternative is to approximate the result of the discrete marginalization operation. In our example, when marginalizing D, we can approximate the resulting mixture of Gaussians by *collapsing* it into a single Gaussian, as shown in figure 14.4b. An appropriate approximation to use in this setting is the *M-projection* operation introduced in definition 8.4. Here, we select the Gaussian distribution $\hat{p}$ that minimizes $\boldsymbol{D}(p\|\hat{p})$. In example 8.15 we provided a precise characterization of this operation:

mixture collapsing

M-projection

Proposition 14.2 *Let p be an arbitrary distribution over $X_1, \ldots, X_k$. Let $\boldsymbol{\mu}$ be the mean vector of p, and Σ be the matrix of covariances in p:*

$$\begin{aligned} \mu_i &= \boldsymbol{E}_p[X_i] \\ \Sigma_{i,j} &= \boldsymbol{Cov}_p[X_i; X_j]. \end{aligned}$$

Then the Gaussian distribution $\hat{p} = \mathcal{N}(\boldsymbol{\mu}; \Sigma)$ is the one that minimizes $\boldsymbol{D}(p\|\hat{p})$ among all Gaussian distributions.

Using this result, we have:

Proposition 14.3 *Let p be the density function of a mixture of k Gaussians $\{\langle w_i, \mathcal{N}(\boldsymbol{\mu}_i; \Sigma_i)\rangle\}_{i=1}^k$ for $\sum_{i=1}^k w_i = 1$. Let $q = \mathcal{N}(\boldsymbol{\mu}; \Sigma)$ be a Gaussian distribution defined as:*

$$\boldsymbol{\mu} = \sum_{i=1}^{k} w_i \boldsymbol{\mu}_i \tag{14.10}$$

$$\Sigma = \sum_{i=1}^{k} w_i \Sigma_i + \sum_{i=1}^{k} w_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T. \tag{14.11}$$

Then q has the same first two moments (means and covariances) as p, and is therefore the Gaussian distribution that minimizes $\boldsymbol{D}(p\|q)$ among all Gaussian distributions.

The proof is left as an exercise (exercise 14.2).

Note that the covariance matrix, as defined by the collapsing operation, has two terms: one term is the weighted average of the covariance matrices of the mixture components; the second corresponds to the distances between the means of the mixture components — the larger these

distances, the larger the "space" between the mixture components, and thus the larger the variances in the new covariance matrix.

Example 14.5

Consider again the discrete marginalization problem in example 14.4. Using proposition 14.3, we have that the mean and variance of the optimal Gaussian approximation to the mixture are:

$$\begin{aligned} \mu &= 0.4 \cdot 0 + 0.6 \cdot 3 = 1.8 \\ \sigma^2 &= (0.4 \cdot 1 + 0.6 \cdot 4) + (0.4 \cdot (1.8 - 0)^2 + 0.6 \cdot (3 - 1.8)^2 = 4.96. \end{aligned}$$

The resulting Gaussian approximation is shown in figure 14.4b. ■

Clearly, when approximating a mixture of Gaussians by one Gaussian, the quality of the approximation depends on how close the mixture density is to a single multivariate Gaussian. When the Gaussians are very different, the approximation can be quite bad.

Using these tools, we can define the discrete marginalization operation.

Definition 14.3

weak marginalization

Assume we have a canonical table defined over $\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}\}$, where $\boldsymbol{A}, \boldsymbol{B} \subseteq \Delta$ and $\boldsymbol{X} \subseteq \Gamma$. Its weak marginal *is a canonical table over $\boldsymbol{A}, \boldsymbol{X}$, defined as follows: For every value $\boldsymbol{a} \in Val(\boldsymbol{A})$, we select the table entries consistent with $\boldsymbol{a}$ and sum them together to obtain a single table entry. The summation operation uses the collapsing operation of proposition 14.3.* ■

One problem with this definition is that the collapsing operation was defined in proposition 14.3 only for a mixture of Gaussians and not for a mixture of general canonical forms. Indeed, the operation of combining canonical forms is well defined if and only if the canonical forms have finite first two moments, which is the case only if they can be represented as Gaussians. This restriction places a constraint on our inference algorithm: we can marginalize a discrete variable only when the associated canonical forms represent Gaussians. We will return to this point.

14.3.3 EP for CLG Networks

The previous section described the basic data structure that we can use to encode factors in a hybrid Gaussian network, and the basic factor operations needed to manipulate them. Most important was the definition of the weak marginalization operation, which approximates a mixture of Gaussians as a single Gaussian, using the concept of M-projection.

Gaussian EP

With our definition of weak marginalization and other operations on canonical tables, we can now define a message passing algorithm based on the framework of the *expectation propagation* described in section 11.4. As a reminder, to perform a message passing step in the EP algorithm, a cluster multiplies all incoming messages, and then performs an approximate marginalization on the resulting product factor. This last step, which can be viewed abstractly as a two-step process — exact marginalization followed by M-projection — is generally performed in a single approximate marginalization step. For example, in section 11.4.2, the approximate marginals were computed by calibrating a cluster graph (or clique tree) and then extracting from it a set of required marginals.

To apply EP in our setting, we need only to define the implementation of the M-projection operation M-Project-Distr, as needed in line 1 of algorithm 11.5. This operation can be performed using the weak marginalization operation described in section 14.3.2.2, as shown in detail in algorithm 14.1. The marginalization step uses two types of operation. The continuous variables

Algorithm 14.1 Expectation propagation message passing for CLG networks

Procedure CLG-M-Project-Distr (
  $\boldsymbol{Z}$, // Scope to remain following projection
  $\vec{\phi}$ // Set of canonical tables
)
1 // Compute overall measure using product of canonical tables
2 $\tilde{\beta} \leftarrow \prod_{\phi \in \vec{\phi}} \phi$
3 // Variables to be preserved
4 $\boldsymbol{A} \leftarrow \boldsymbol{Z} \cap \Delta$
5 $\boldsymbol{X} \leftarrow \boldsymbol{Z} \cap \Gamma$
6 // Variables to be eliminated
7 $\boldsymbol{B} \leftarrow (\mathit{Scope}[\vec{\phi}] - \boldsymbol{A}) \cap \Delta$
8 $\boldsymbol{Y} \leftarrow (\mathit{Scope}[\vec{\phi}] - \boldsymbol{X}) \cap \Gamma$
9 **for** each $\boldsymbol{a}, \boldsymbol{b} \in \mathit{Val}(\boldsymbol{A}, \boldsymbol{B})$
10 $\tau(\boldsymbol{a}, \boldsymbol{b}) \leftarrow \int \beta_i(\boldsymbol{a}, \boldsymbol{b}) d\boldsymbol{Y}$ using equation (14.5)
11 $\tilde{\sigma} \leftarrow \sum_{\boldsymbol{B}} \tau$ using definition 14.3
12 **return** $\tilde{\sigma}$

are eliminated using the marginalization operation of equation (14.5) over each entry in the canonical table separately. The discrete variables are then summed out. For this last step, there are two cases. If the factor τ contains only discrete variables, then we use standard discrete marginalization. If not, then we use the weak marginalization operation of definition 14.3.

In principle, this application of the EP framework is fairly straightforward. There are, however, two important subtleties that arise in this setting.

14.3.3.1 Ordering Constraints

First, as we discussed, for weak marginalization to be well defined, the canonical form being marginalized needs to be a Gaussian distribution, and not merely a canonical form. In some cases, this requirement is satisfied simply because of the form of the potentials in the original network factorization. For example, in the Gaussian case, recall that our message passing process is well defined if our distribution is pairwise normalizable. In the conditional Gaussian case, we can guarantee normalizability if for each cluster over scope $\boldsymbol{X}$, the initial factor in that cluster is a canonical table such that each canonical form entry $\mathcal{C}(\boldsymbol{X}; K_{\boldsymbol{d}}, \boldsymbol{h}_{\boldsymbol{d}}, g_{\boldsymbol{d}})$ is normalizable. Because normalizability is closed under factor product (because the sum of two PSD matrices is also PSD) and (both weak and strong) marginalization, this requirement guarantees us that all factors produced by a sum-product algorithm will be normalizable.

However, this requirement is not always easy to satisfy:

Example 14.6 *Consider a CLG network structured as in figure 14.5a, and the clique tree shown in figure 14.5b. Note that the only clique where each canonical form is normalizable is $\boldsymbol{C}_1 = \{A, X\}$; in $\boldsymbol{C}_2 = \{B, X, Y\}$, the canonical forms in the canonical table are all linear Gaussians whose integral is infinite, and hence cannot be collapsed in the operation of proposition 14.3.*

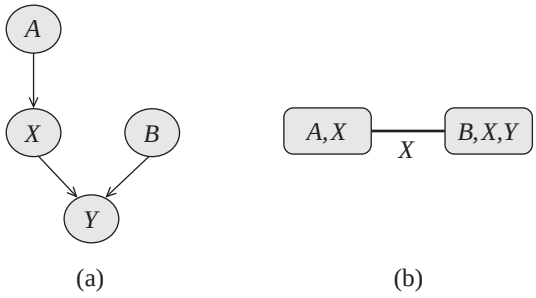


Figure 14.5 Example of unnormalizable potentials in a CLG clique tree. (a) A simple CLG (b) A clique tree for it.

In this network, with an appropriate message passing order, one can guarantee that canonical tables are normalizable at the appropriate time point. In particular, we can first pass a message from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$; this message is a Gaussian obtained by weak marginalization of $p(A, X)$ onto X. The resulting potential at $\boldsymbol{C}_2$ is now a product of a legal Gaussian density over X (derived from the incoming message) multiplied by $P(B)$ and the conditional linear Gaussian $p(Y \mid X, B)$. The resulting distribution is a standard mixture of Gaussians, where each component in the mixture is normalizable. Thus, weak marginalization onto Y can be performed, allowing the message passing process to continue. ■

This example illustrates that we can sometimes find a legal message passing order even in cases where the initial potentials are not normalizable. However, such a message passing order may not always exist.

Example 14.7

Consider the network in figure 14.6a. After moralization, the graph, shown in figure 14.6b, is already triangulated. If we now extract the maximum cliques and build the clique tree, we get the tree shown in figure 14.6c. Unfortunately, at this point, neither of the two leaves in this clique tree can send a message. For example, the clique $\{B, X, Y\}$ contains the CPDs for $P(B)$ and $P(Y \mid B, X)$, but not the CPD for X. Hence, the canonical forms over $\{X, Y\}$ represent linear Gaussian CPDs and not Gaussians. It follows that we cannot marginalize out B, and thus this clique cannot send a message. For similar reasons, the clique $\{A, C, Y, Z\}$ cannot send a message. Note, however, that a different clique tree can admit message passing. In particular, both the trees in (d) and (e) admit message passing using weak marginalization. ■

order-constrained message passing

As we can see, not every cluster graph allows message passing based on weak marginalization to take place. More formally, we say that a variable X_i is *order constrained* at cluster $\boldsymbol{C}_k$ if we require a normalizable probability distribution over X_i at $\boldsymbol{C}_k$ in order to send messages. In a cluster graph for a CLG Bayesian network, if $\boldsymbol{C}_k$ requires weak marginalization, then any continuous variable X_i in $\boldsymbol{C}_k$ is order constrained.

If X_i is order constrained in $\boldsymbol{C}_k$, then we need to ensure that X_i has a well-defined distribution. In order for that to hold, $\boldsymbol{C}_k$ must have obtained a valid probability distribution over X_i's parents, whether from a factor within $\boldsymbol{C}_k$ or from a message sent by another cluster. Now, consider a continuous parent X_j of X_i, and let $\boldsymbol{C}_l$ be the cluster from which $\boldsymbol{C}_k$ obtains the

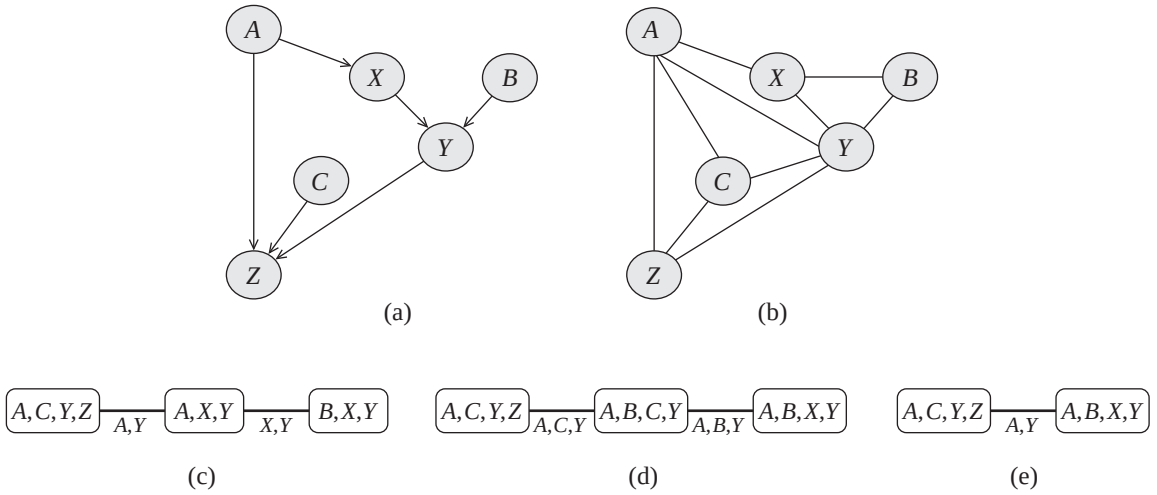


Figure 14.6 A simple CLG and possible clique trees with different correctness properties. (a) A simple CLG. (b) The moralized (and triangulated) graph for the CLG. (c) Clique tree derived the moralized graph in (b). (d) A clique tree with a strong root. (e) A clique tree without a strong root that allows message passing using weak marginalization.

distribution over X_j. (In the first case, we have $k = l$.) In order for X_j to have a well-defined distribution in $\boldsymbol{C}_l$, we must have that X_j is order constrained in $\boldsymbol{C}_l$. This process continues until the roots of the networks are reached.

In the context of Markov networks, the situation is somewhat less complex. The use of a global partition function allows us to specify models where individual factors are all normalizable, ensuring a legal measure. In particular, extending definition 7.3, we can require that all of the entries in all of the canonical tables parameterizing the network be *normalizable*. In this case, the factors in all clusters in the network can be normalized to produce valid distributions, avoiding any constraints on message passing. Of course, the factor normalizability constraint is only a sufficient condition, in that there are models that do not satisfy this constraint and yet allow weak marginalization to take place, if messages are carefully ordered.

Gaussian normalizability

As this discussion shows, **the constraint on normalizability of the Gaussian in the context of weak marginalization can impose significant constraints on the structure of the cluster graph and/or the order of the message passing.**

14.3.3.2 Ill-Defined Messages

belief-update

The second subtlety arises when we apply the *belief-update* message passing algorithm rather than sum product. Recall that the belief update algorithm has important advantages, in that it gradually tunes the approximation to the more relevant regions in the probability space.

Example 14.8

Consider the application of the sum-product belief propagation algorithm to the network of figure 14.5. Assume furthermore that the distribution at $\boldsymbol{C}_1$ is as described in example 14.5, so that

the result of weak marginalization is as shown in figure 14.4b. Now, assume that $p(Y \mid x, b^1) = \mathcal{N}(Y \mid x; 1)$*, and that we observe* $b = b^1$ *and* $Y = -2$*. The evidence* $Y = -2$ *is much more consistent with the left-hand (mean* 0*) component in the mixture, which is the one derived from* $p(X \mid a^0)$*. Given this evidence, the exact posterior over* X *would give much higher probability to the* a^0 *component in the mixture. However, our Gaussian approximation was the M-projection of a mixture where this component had its prior weight of* 0.4*. To obtain a better approximation, we would want to construct a new M-projection in* $\boldsymbol{C}_1$ *that gives more weight to the* $\mathcal{N}(X \mid 0; 1)$ *component in the mixture when collapsing the two Gaussians.* ■

This issue is precisely the motivation that we used in section 11.4.3.2 for the belief update algorithm. **However, the use of division in the Gaussian EP algorithm can create unexpected complications, in that the messages passed can represent unnormalizable densities, with negative variance terms; these can lead, in turn, to unnormalizable densities in the clusters, and thereby to nonsensical results.**

Example 14.9

Consider again the CLG in figure 14.5a, but where we now assume that the CPDs of the discrete variables are uniform and the CPDs of the continuous nodes are defined as follows:

$$\begin{aligned} p(X \mid A) &= \begin{cases} \mathcal{N}(0; 2) & A = a^0 \\ \mathcal{N}(0; 6) & A = a^1 \end{cases} \\ p(Y \mid B, X) &= \begin{cases} \mathcal{N}(X; 0.01) & B = b^0 \\ \mathcal{N}(-X; 0.01) & B = b^1. \end{cases} \end{aligned}$$

Consider the execution of message passing on the clique tree of figure 14.5b, with the evidence $Y = 4$*. To make the presentation easier to follow, we present some of the intermediate results in moments form (means and covariances) rather than canonical form.*

The message passing algorithm first sends a message from the clique $\{A, X\}$ *to the clique* $\{B, X, Y\}$*. Since the cliques share only the variable* X*, we collapse the prior distribution of* X *using proposition 14.3 and get the message* $\delta_{1\to 2} = \mathcal{N}(X \mid 0; 4)$*. This message is stored in the sepset at* $\mu_{1,2}$*. We also multiply the potential of* $\{B, X, Y\}$ *by this message, getting a mixture of two Gaussians with equal weights:*

$$\begin{aligned} \beta_2(b^0) &= \mathcal{N}\left(X, Y \mid \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \begin{bmatrix} 4 & 4 \\ 4 & 4.01 \end{bmatrix}\right) \\ \beta_2(b^1) &= \mathcal{N}\left(X, Y \mid \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \begin{bmatrix} 4 & -4 \\ -4 & 4.01 \end{bmatrix}\right). \end{aligned}$$

After instantiating the evidence $Y = 4$ *we get a new mixture of two Gaussians:*

$$\begin{aligned} \beta_2(b^0) &= \mathcal{N}(X \mid 3.99; 0.00998) \\ \beta_2(b^1) &= \mathcal{N}(X \mid -3.99; 0.00998). \end{aligned}$$

Note that, in the original mixture, Y *has the same marginal — the same* $\mathcal{N}(Y \mid 0; 4.01)$ *— for both* b^0 *and* b^1*. Therefore, the evidence* $Y = 4$ *has the same likelihood in both cases, so that the posterior weights of the two cases are still the same as each other.*

We now need to send a message back to the clique $\{A, X\}$*. To do so, we collapse the two Gaussians, resulting in a message* $\delta_{2\to 1} = \mathcal{N}(X \mid 0; 15.93)$*. Note that, in this example, the evidence causes the variance of* X *to increase.*

To incorporate the message $\delta_{2\to 1}$ into the clique $\{A, X\}$, we divide it by the message $\mu_{1,2}$, and multiply each entry in $\beta_1(A, X)$ by the resulting quotient $\frac{\delta_{2\to 1}}{\mu_{1,2}}$. In particular, for $A = a^1$, we perform the operation:

$$\frac{\mathcal{N}(0;6) \cdot \mathcal{N}(0;15.93)}{\mathcal{N}(0;4)}.$$

This operation can be carried out using the canonical form $\mathcal{C}(K, \boldsymbol{h}, g)$. Consider the operation over the coefficient K, which represents the inverse of the covariance matrix: $K = \Sigma^{-1}$. In our case, the Gaussians are all one-dimensional, so $K = \frac{1}{\sigma^2}$. As in equation (14.2) and equation (14.3), the product and division operation reduces to addition and subtraction of the coefficient K. Thus, the K for the resulting potential is:

$$\frac{1}{6} + \frac{1}{15.93} - \frac{1}{4} = -0.0206 < 0!$$

However, $K < 0$ does not represent a legal Gaussian: it corresponds to $\sigma^2 = \frac{1}{-0.0206}$, which is not a legal variance. ■

In practice, this type of situation does occur, but not often. Therefore, despite this complication, the belief-update variant of the EP algorithm is often used in practice.

14.3.4 An "Exact" CLG Algorithm ⋆

There is one case where we can guarantee that the belief update algorithm is not only well defined but even returns "exact" answers. Of course, truly exact answers are not generally possible in a CLG network. Recall that a CLG distribution is a mixture of possibly exponentially many Gaussian hypotheses. The marginal distribution over a single variable can similarly be an exponentially large mixture (as in figure 14.2). Thus, for a query regarding the marginal distribution of a single continuous variable, even representing the answer might be intractable.

Lauritzen's algorithm

Lauritzen's algorithm provides a compromise between correctness and computational efficiency. It is exact for queries involving discrete variables, and it provides the exact first and second moments — means and (co)variances — for the continuous variables. More precisely, for a query $p(\boldsymbol{D}, \boldsymbol{X} \mid \boldsymbol{e})$ for $\boldsymbol{D} \subseteq \Delta$ and $\boldsymbol{X} \subseteq \Gamma$, Lauritzen's algorithm returns an answer $\hat{p}$ such that $\hat{p}(\boldsymbol{D}) = p(\boldsymbol{D} \mid \boldsymbol{e})$ is correct, and for each $\boldsymbol{d}$, $\hat{p}(\boldsymbol{X} \mid \boldsymbol{d})$ has the correct first and second moments. For many applications, queries of this type are sufficient.

The algorithm is a modification of the clique tree algorithm for discrete networks. It uses precisely the same type of message passing that we described, but over a highly restricted clique tree data structure. As we will show, these restrictions can (and do) cause significant blowup in the size of the clique tree (much larger than the induced width of the network) and hence do not violate the $\mathcal{NP}$-hardness of inference in these graphs. Indeed, these restrictions restrict the algorithm's usefulness to a fairly narrow class of problems, and they render it primarily of theoretical interest.

14.3.4.1 Strongly Rooted Trees

The ordering constraint we employ on the clique tree, as described in section 14.3.3.1, guarantees that we can calibrate the clique tree without requiring weak marginalization on the upward pass.

That is, the clique tree has a particular clique that is a *strong root*; when we pass messages from the leaves of the clique tree toward this root, no weak marginalization is required.

This property has several implications. First, weak marginalization is the only operation that requires that the clique potential be normalizable. In the upward pass (from the leaves toward the root), no such marginalization is required, and so we need not worry about this constraint. Intuitively, in the downward pass, each clique has already obtained all of the necessary information to define a probability distribution over its scope. This distribution allows weak marginalization to be performed in a well-defined way. Second, because weak marginalization is the only operation that involves approximation, this requirement guarantees that the message passing in the upward pass is exact. As we will discuss, this property suffices to guarantee that the weak marginals in the downward pass are all legal. Indeed, as we will show, this property also allows us to guarantee that these marginals all possess the correct first and second moments (means and covariances).

When does our clique tree structure guarantee that no weak marginalization has to take place? Consider two cliques $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, such that $\boldsymbol{C}_2$ is the upward neighbor of $\boldsymbol{C}_1$. There are two cases. Either no marginalization of any discrete variable takes place between $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, or it does. In the first case, only continuous variables are eliminated, so that $\boldsymbol{C}_2 - \boldsymbol{C}_1 \subseteq \Gamma$. In the second case, in order to avoid weak marginalization, we must avoid collapsing any Gaussians. Thus, we must have that the message from $\boldsymbol{C}_2$ to $\boldsymbol{C}_1$ contains no continuous variables, so that $\boldsymbol{C}_1 \cap \boldsymbol{C}_2 \subseteq \Delta$. Overall, we have:

Definition 14.4 strong root

A clique $\boldsymbol{C}_r$ in a clique tree $\mathcal{T}$ is a strong root *if for every clique $\boldsymbol{C}_1$ and its upward neighbor $\boldsymbol{C}_2$, we have that $\boldsymbol{C}_2 - \boldsymbol{C}_1 \subseteq \Gamma$ or that $\boldsymbol{C}_1 \cap \boldsymbol{C}_2 \subseteq \Delta$. A clique tree is called* strongly rooted *if it has a strong root.* ■

Example 14.10

figure 14.6d shows a strongly rooted clique tree for the network of figure 14.6a. Here, the strong root is the clique $\{A, B, C, Y\}$. For both of the two nonroot cliques, we have that no discrete variables are marginalized between them and their upward neighbor $\{A, B, C, Y\}$, so that the first condition of the definition holds. ■

If we now apply the message passing procedure of algorithm 14.1, we note that the weak marginalization can only occur in the downward pass. In the downward pass, $\boldsymbol{C}_i$ has received messages from all of its neighbors, and therefore $\beta_i(\boldsymbol{C}_i)$ represents a probability distribution over $\boldsymbol{C}_i$. Hence, $\tau(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X})$ is a mixture of Gaussians over $\boldsymbol{X}$, so that the weak marginalization operation is well defined.

14.3.4.2 Strong Roots and Correctness

So far, we have shown that the strongly rooted requirement ensures that the operations in the clique tree are well defined, specifically, that no collapsing is performed unless the canonical forms represent Gaussians. However, as we have already shown, this condition is not necessary for the message passing to be well defined; for example, the clique tree of figure 14.6e is not strongly rooted, but allows weak marginalization. However, as we now show, there are other reasons for using strongly rooted trees for inference in hybrid networks. Specifically, the presence of a strong root ensures not only that message passing is well defined, but also that message passing leads to exact results, in the sense that we described.

Theorem 14.4

Let $\mathcal{T}$ be a clique tree and $\boldsymbol{C}_r$ be a strong root in $\mathcal{T}$. After instantiating the evidence and running CTree-BU-Calibrate *using* EP-Message *with* CLG-M-Project-Distr *for message passing, the tree $\mathcal{T}$ is calibrated and every potential contains the correct (weak) marginal. In particular, every clique $\boldsymbol{C}$ contains the correct probability distribution over the discrete variables $\boldsymbol{C} \cap \Delta$ and the correct mean and covariances of the continuous variables $\boldsymbol{C} \cap \Gamma$.*

Proof Consider the steps in the algorithm CTree-BU-Calibrate. The clique initialization step is exact: each CPD is multiplied into a clique that contains all of its variables, and the product operation for canonical tables is exact. Similarly, evidence instantiation is also exact. It remains only to show that the message passing phase is exact.

The upward pass is simple — as we discussed, all of the marginalization operations involved are strong marginalizations, and therefore all the operations are exact. Thus, the upward pass is equivalent to running the variable elimination algorithm for the variables in the strong root, and its correctness follows from the correctness of the variable elimination algorithm. The result of the upward pass is the correct (strong) marginal in the strong root $\boldsymbol{C}_r$.

The downward pass involves weak marginalization, and therefore, it will generally not result in the correct distribution. We wish to show that the resulting distributions in the cliques are the correct *weak* marginals. The proof is by induction on the distance of the clique from the strong root $\boldsymbol{C}_r$. The base case is the root clique itself, where we have already shown that we have exactly the correct marginal. Now, assume now that we have two cliques $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ such that $\boldsymbol{C}_i$ is the upward neighbor of $\boldsymbol{C}_j$. By the inductive hypothesis, after $\boldsymbol{C}_i$ receives the message from its upward neighbor, it has the correct weak marginals. We need to show that, after $\boldsymbol{C}_j$ receives the message from $\boldsymbol{C}_i$, it also has the correct weak marginal.

Let β_j and β'_j denote the potential of $\boldsymbol{C}_j$ before and after $\boldsymbol{C}_i$ sends the downward message to $\boldsymbol{C}_j$. Let $\mu_{i,j}$ denote the sepset message before the downward message, and $\delta_{i \to j}$ denote the actual message sent. Note that $\delta_{i \to j}$ is the (weak) marginal of the clique potential β_i, and is therefore the correct weak marginal, by the inductive hypothesis.

We first show that, after the message is sent, $\boldsymbol{C}_j$ agrees with $\boldsymbol{C}_i$ on the marginal distribution of the sepset $\boldsymbol{S}_{i,j}$.

$$\sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta'_j = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j \cdot \frac{\delta_{i \to j}}{\mu_{i,j}} = \frac{\delta_{i \to j}}{\mu_{i,j}} \cdot \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j = \frac{\delta_{i \to j}}{\mu_{i,j}} \cdot \mu_{i,j} = \delta_{i \to j}, \tag{14.12}$$

where the marginalization $\sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}}$ also denotes integration, when appropriate. This derivation is correct because this marginalization $\sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}}$ is an exact operation: By the strong root property, all marginalizations toward the strong root (that is, from j to i) are strong marginalizations. Thus, the marginal of $\boldsymbol{C}_j$ after the message is the same as the (weak) marginal of $\boldsymbol{C}_i$, and the two cliques are calibrated. Because the (weak) marginal of $\boldsymbol{C}_i$ is correct, so is the marginal of $\boldsymbol{C}_j$. Note that this property does *not* hold in a tree that is not strongly rooted. In general, in such a clique tree, $\mu_{i,j} \neq \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j$.

It remains to show that β'_j is the correct weak marginal for all the variables in $\boldsymbol{C}_j$. As shown in exercise 10.5, the premessage potential β_j already encodes the correct posterior conditional distribution $P(\boldsymbol{C}_j \mid \boldsymbol{S}_{i,j})$. (We use P to denote the posterior, after conditioning on the

evidence.) In other words, letting $\boldsymbol{X} = \boldsymbol{C}_j - \boldsymbol{S}_{i,j}$ and $\boldsymbol{S} = \boldsymbol{S}_{i,j}$, we have that:

$$\frac{\beta_j(\boldsymbol{X}, \boldsymbol{S})}{\beta_j(\boldsymbol{S})} = P(\boldsymbol{X} \mid \boldsymbol{S}).$$

Now, since the last message along the edge $\boldsymbol{C}_i$—$\boldsymbol{C}_j$ was sent from $\boldsymbol{C}_j$ to $\boldsymbol{C}_i$, we have that

$$\frac{\beta_j(\boldsymbol{X}, \boldsymbol{S})}{\beta_j(\boldsymbol{S})} = \frac{\beta_j(\boldsymbol{X}, \boldsymbol{S})}{\mu_{i,j}(\boldsymbol{S})}.$$

We therefore have that the potential of $\boldsymbol{C}_j$ after the message from $\boldsymbol{C}_i$ is:

$$\beta'_j = \frac{\beta_j \delta_{i \to j}}{\mu_{i,j}} = \delta_{i \to j} P(\boldsymbol{X} \mid \boldsymbol{S}).$$

Thus, every entry in β'_j is a Gaussian, which is derived as a product of two terms: one is a Gaussian over $\boldsymbol{S}$ that has the same first and second moments as P, and the second is the correct $P(\boldsymbol{X} \mid \boldsymbol{S})$. It follows easily that the resulting product $\beta_j(\boldsymbol{X}, \boldsymbol{S})$ is a Gaussian that has the same first and second moments as P (see exercise 14.5). This concludes the proof of the result. ■

Note that, if the tree is not strongly rooted, the proof breaks down in two places: the upward pass is not exact, and equation (14.12) does not hold. Both of these arise from the fact that $\sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j$ is the exact marginal, whereas, if the tree is not strongly rooted, $\mu_{i,j}$ is computed (in some cliques) using weak marginalization. Thus, the two are not, in general, equal. As a consequence, although the weak marginal of β_j *is* $\mu_{i,j}$, the second equality fails in the derivation: the weak marginal of a product $\beta_j \cdot \frac{\delta_{i \to j}}{\mu_{i,j}}$ is not generally equal to the product of $\frac{\delta_{i \to j}}{\mu_{i,j}}$ and the weak marginal of β_j. Thus, the strong root property is essential for the strong correctness properties of this algorithm.

14.3.4.3 Strong Triangulation and Complexity

We see that strongly rooted trees are necessary for the correct execution of the clique tree algorithm in CLG networks. Thus, we next consider the task of constructing a strongly rooted tree for a given network. As we discussed, one of our main motivations for the strong root requirement was the ability to perform the upward pass of the clique tree algorithm without weak marginalization. Intuitively, the requirement that discrete variables not be eliminated before their continuous neighbors implies a *constraint on the elimination ordering* within the clique tree. Unfortunately, constrained elimination orderings can give rise to clique trees that are much larger — exponentially larger — than the optimal, unconstrained clique tree for the same network. We can analyze the implications of the strong triangulation constraint in terms of the network structure.

constrained elimination ordering

Definition 14.5

Let $\mathcal{G}$ be a hybrid network. A continuous connected component *in $\mathcal{G}$ is a set of variables $\boldsymbol{X} \subseteq \Gamma$ such that: if $X_1, X_2 \in \boldsymbol{X}$ then there exists a path between X_1 and X_2 in the moralized graph $\mathcal{M}[\mathcal{G}]$ such that all the nodes on the path are in Γ. A continuous connected component is* maximal *if it is not a proper subset of any larger continuous connected component. The* discrete neighbors *of a continuous connected component $\boldsymbol{X}$ are all the discrete variables that are adjacent to some node $X \in \boldsymbol{X}$ in the moralized graph $\mathcal{M}[\mathcal{G}]$.* ■

For example, in the network figure 14.6a, all of the continuous variables $\{X, Y, Z\}$ are in a single continuous connected component, and all the discrete variables are its neighbors.

We can understand the implications of the strong triangulation requirement in terms of continuous connected components:

Theorem 14.5 *Let $\mathcal{G}$ be a hybrid network, and let $\mathcal{T}$ be a strongly rooted clique tree for $\mathcal{G}$. Then for any maximal continuous connected component $\boldsymbol{X}$ in $\mathcal{G}$, with $\boldsymbol{D}$ its discrete neighbors, $\mathcal{T}$ includes a clique that contains (at least) all of the nodes in $\boldsymbol{D}$ and some node $X \in \boldsymbol{X}$.*

The proof is left as an exercise (exercise 14.6).

In the CLG of figure 14.6a, all of the continuous variables are in one connected component, and all of the discrete variables are its neighbors. Thus, a strongly rooted tree must have a clique that contains all of the discrete variables and one of the continuous ones. Indeed, the strongly rooted clique tree of figure 14.6d contains such a clique — the clique $\{A, B, C, Y\}$.

This analysis allows us to examine a CLG network, and immediately conclude lower bounds on the computational complexity of clique tree inference in that network. For example, the polytree CLG in figure 14.2 has a continuous connected component containing all of the continuous variables $\{X_1, \ldots, X_n\}$, which has all of the discrete variables as neighbors. Thus, any strongly rooted clique tree for this network necessarily has an exponentially large clique, which is as we would expect, given that this network is the basis for our $\mathcal{NP}$-hardness theorem.

Because many CLG networks have large continuous connected components that are adjacent to many discrete variables, a strongly rooted clique tree is often far too large to be useful. However, the algorithm presented in this section is of conceptual interest, since it clearly illustrates when the EP message passing can lead to inaccurate or even nonsensical answers.

14.4 Nonlinear Dependencies

In the previous sections, we dealt with a very narrow class of continuous networks: those where all of the CPDs are parameterized as linear Gaussians. Unfortunately, this class of networks is inadequate as a model for many practical applications, even those involving only continuous variables. For example, as we discussed, in modeling the car's position as a function of its previous position and its velocity (as in example 5.20), rather than assume that the variance in the car's position is constant, it might be more reasonable to assume that the variance of the car is larger if the velocity is large. This dependence is nonlinear, and it cannot be accommodated within the framework of linear Gaussians. In this section, we relax the assumption of linearity and present one approach for dealing with continuous networks that include nonlinear dependencies. We note that the techniques in this section can be combined with the ones described in section 14.3 to allow our algorithms to extend to networks that allow discrete variables to depend on continuous parents.

Once again, the standard solution in this setting can be viewed as an instance of the general expectation propagation framework described in section 11.4.4, and used before in the context of the CLG models. Since we cannot tractably represent and manipulate the exact factors in this setting, we need to use an approximation, by which intermediate factors in the computation are approximated using a compact parametric family. Once again, we choose the family of Gaussian

measures as our representation.

At a high level, the algorithm proceeds as follows. As in expectation propagation (EP), each cluster $\boldsymbol{C}_i$ maintains its potentials in a nonexplicit form, as a factor set $\vec{\phi}_i$; some of these factors are from the initial factor set Φ, and others from the incoming messages into $\boldsymbol{C}_i$. Importantly, due to our use of a Gaussian representation for the EP factors, the messages are all Gaussian measures.

linearization

To pass a message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$, $\boldsymbol{C}_i$ approximates the product of the factors in $\vec{\phi}_i$ as a Gaussian distribution, a process called *linearization*, for reasons we will explain. The resulting Gaussian distribution can then be marginalized onto $\boldsymbol{S}_{i,j}$ to produce the approximate cluster marginal $\tilde{\sigma}_{i \to j}$. Essentially, the combination of the linearization step and the marginalization of the resulting Gaussian over the sepset give rise to the weak marginalization operation that we described.

The basic computational step in this algorithm is the linearization operation. We provide several options for performing this operation, and discuss their trade-offs. We then describe in greater detail how this operation can be used within the EP algorithm, and the constraints that it imposes on its detailed implementation.

14.4.1 Linearization

We first consider the basic computational task of approximating a distribution $p(X_1, \ldots, X_d)$ as a Gaussian distribution $\hat{p}(\boldsymbol{X})$. For the purposes of this section, we assume that our distribution is defined in terms of a Gaussian distribution $p_0(Z_1, \ldots, Z_l) = \mathcal{N}(\boldsymbol{Z} \mid \boldsymbol{\mu}; \Sigma)$ and a set of deterministic functions $X_i = f_i(\boldsymbol{Z}_i)$. Intuitively, the auxiliary $\boldsymbol{Z}$ variables encompass all of the stochasticity in the distribution, and the functions f_i serve to convert these auxiliary variables into the variables of interest. For a vector of functions $\vec{f} = (f_1, \ldots, f_d)$ and a Gaussian distribution p_0, we can now define $p(\boldsymbol{X}, \boldsymbol{Z})$ to be the distribution that has $p(\boldsymbol{Z}) = p_0(\boldsymbol{Z})$ and $X_i = f_i(\boldsymbol{Z})$ with probability 1. (Note that this distribution has a point mass at the discrete points where $\boldsymbol{X} = \vec{f}(\boldsymbol{Z})$.) We use $p(\boldsymbol{X}) = p_0(\boldsymbol{Z}) \bigoplus [\boldsymbol{X} = \vec{f}(\boldsymbol{Z})]$ to refer to the marginal of this distribution over $\boldsymbol{X}$.

linearization

Our goal is to compute a Gaussian distribution $\hat{p}(\boldsymbol{X})$ that approximates this marginal distribution $p(\boldsymbol{X})$. We call the procedure of determining $\hat{p}$ from p_0 and $\vec{f}$ a *linearization* of $\vec{f}$. We now describe the two main approaches that have been proposed for the linearization operation.

14.4.1.1 Taylor Series Linearization

As we know, if $p_0(\boldsymbol{Z})$ is a Gaussian distribution and $X = f(\boldsymbol{Z})$ is a linear function, then $p(X) = p(f(\boldsymbol{Z}))$ is also a Gaussian distribution. Thus, one very simple and commonly used approach is to approximate f as a linear function $\hat{f}$, and then define $\hat{p}$ in terms of $\hat{f}$.

Taylor series

The most standard linear approximation for $f(\boldsymbol{Z})$ is the *Taylor series* expansion around the mean of $p_0(\boldsymbol{Z})$:

$$\hat{f}(\boldsymbol{Z}) = f(\boldsymbol{\mu}) + \nabla f|_{\boldsymbol{\mu}} \boldsymbol{Z}. \tag{14.13}$$

Extended Kalman filter

The Taylor series is used as the basis for the famous *extended Kalman filter* (see section 15.4.1.2).

Although the Taylor series expansion provides us with the optimal linear approximation to

f, the Gaussian $\hat{p}(X) = p_0(\boldsymbol{Z}) \bigoplus \hat{f}(\boldsymbol{Z})$ may not be the optimal Gaussian approximation to $p(X) = p_0(\boldsymbol{Z}) \bigoplus f(\boldsymbol{Z})$.

Example 14.11

Consider the function $X = Z^2$, and assume that $p(Z) = \mathcal{N}(Z \mid 0; 1)$. The mean of X is simply $\boldsymbol{E}_p[X] = \boldsymbol{E}_p[Z^2] = 1$. The variance of X is

$$\boldsymbol{Var}_p[X] = \boldsymbol{E}_p[X^2] - \boldsymbol{E}_p[X]^2 = \boldsymbol{E}_p[Z^4] - \boldsymbol{E}_p[Z^2]^2 = 3 - 1^2 = 2.$$

On the other hand, the first-order Taylor series approximation of f at the mean value $Z = 0$ is:

$$\hat{f}(Z) = 0^2 + (2Z)_{U=0} Z \equiv 0.$$

Thus, $\hat{p}(X)$ will simply be a delta function where all the mass is located at $X = 0$, a very poor approximation to p. ■

This example illustrates a limitation of this simple approach. In general, **the quality of the Taylor series approximation depends on how well $\hat{f}$ approximates f in the neighborhood of the mean of $\boldsymbol{Z}$, where the size of the neighborhood is determined by the variance of $p_0(\boldsymbol{Z})$. The approximation is good only if the linear term in the Taylor expansion of f dominates in this neighborhood, and the higher-order terms are small.** In many practical situations, this is not the case, for example, when f changes very rapidly relative to the variance of $p_0(\boldsymbol{Z})$. In this case, using the simple Taylor series approach can lead to a very poor approximation.

14.4.1.2 M-Projection Using Numerical Integration

The Taylor series approach uses what may be considered an indirect approach to approximating p: we first simplify the nonlinear function f and only then compute the resulting distribution.

M-projection

Alternatively, we can directly approximate p using a Gaussian distribution $\hat{p}$, by using the *M-projection* operation introduced in definition 8.4. Here, we select the Gaussian distribution $\hat{p}$ that minimizes $\boldsymbol{D}(p \| \hat{p})$.

In proposition 14.2 we provided a precise characterization of this operation. In particular, we showed that we can obtain the M-projection of p by evaluating the following set of integrals, corresponding to the moments of p:

$$\boldsymbol{E}_p[X_i] = \int_{-\infty}^{\infty} f_i(\boldsymbol{z}) p_0(\boldsymbol{z}) d\boldsymbol{z} \tag{14.14}$$

$$\boldsymbol{E}_p[X_i X_j] = \int_{-\infty}^{\infty} f_i(\boldsymbol{z}) f_j(\boldsymbol{z}) p_0(\boldsymbol{z}) d\boldsymbol{z}. \tag{14.15}$$

From these moments, we can derive the mean and covariance matrix for p, which gives us precisely the M-projection. Thus, the M-projection task reduces to one of computing the expectation of some function f (which may be f_i or a product $f_i f_j$) relative to our distribution p_0. Before we discuss the solution of these integrals, it is important to inject a note of caution. Even if p is a valid density, its moments may be infinite, preventing it from being approximated by a Gaussian.

In some cases, it is possible to solve the integrals in closed form, leading to an efficient and optimal way of computing the best Gaussian approximation. For instance, in the case of

example 14.11, equation (14.14) reduces to computing $\mathbf{E}_p\left[Z^2\right]$, where p is $\mathcal{N}(0;1)$, an integral that can be easily solved in closed form. Unfortunately, for many functions f, these integrals have no closed-form solutions. However, because our goal is simply to estimate these quantities, we can use *numerical integration* methods. There are many such methods, with various trade-offs. In our setting, we can exploit the fact that our task is to integrate the product of a function and a Gaussian. Two methods that are particularly effective in this setting are described in the following subsections.

numerical integration

Gaussian quadrature

Gaussian Quadrature *Gaussian quadrature* is a method that was developed for the case of one-dimensional integrals. It approximates integrals of the form $\int_a^b W(z)f(z)dz$ where $W(z)$ is a known nonnegative function (in our case a Gaussian). Based on the function W, we choose m points $z_1, \ldots, z_m$ and m weights $w_1, \ldots, w_m$ and approximate the integral as:

$$\int_a^b W(z)f(z)dz \approx \sum_{j=1}^{m} w_j f(z_j). \tag{14.16}$$

integration rule

precision

The points and weights are chosen such that the integral is exact if f is a polynomial of degree $2m-1$ or less. Such *rules* are said to have *precision* $2m-1$.

To understand this construction, assume that we have chosen m points $z_1, \ldots, z_m$ and m weights $w_1, \ldots, w_m$ so that equation (14.16) holds with equality for any monomial z^i for $i = 1, \ldots, 2m-1$. Now, consider any polynomial of degree at most $2m-1$: $f(z) = \sum_{i=0}^{2m-1} \alpha_i z^i$. For such an f, we can show (exercise 14.8) that:

$$\int_a^b W(z)f(z)dz = \sum_{j=1}^{m} w_j z_j^i.$$

Thus, if our points are exact for any monomial of degree up to $2m-1$, it is also exact for any polynomial of this degree.

Example 14.12

Consider the case of $m = 2$. In order for the rule to be exact for $f_0, \ldots, f_3$, it must be the case that for $i = 0, \ldots, 3$ we have

$$\int_a^b W(z)f_i(z)dz = w_1 f_i(z_1) + w_2 f_i(z_2).$$

Assuming that $W(z) = \mathcal{N}(0;1)$, $a = -\infty$, and $b = \infty$, we get the following set of four nonlinear equations

$$\begin{aligned}
w_1 + w_2 &= \int_{-\infty}^{\infty} \mathcal{N}(z \mid 0;1)\, dz = 1 \\
w_1 z_1 + w_2 z_2 &= \int_{-\infty}^{\infty} \mathcal{N}(z \mid 0;1)\, z\, dz = 0 \\
w_1 z_1^2 + w_2 z_2^2 &= \int_{-\infty}^{\infty} \mathcal{N}(z \mid 0;1)\, z^2\, dz = 1 \\
w_1 z_1^3 + w_2 z_2^3 &= \int_{-\infty}^{\infty} \mathcal{N}(z \mid 0;1)\, z^3\, dz = 0
\end{aligned}$$

The solution for these equations (up to swapping z_1 and z_2) is $w_1 = w_2 = 0.5$, $z_1 = -1$, $z_2 = 1$.

This solution gives rise to the following approximation, which we apply to any function f:

$$\int_{-\infty}^{\infty} \mathcal{N}(0;1)\, f(z)dz \approx 0.5f(-1) + 0.5f(1).$$

This approximation is exact for any polynomial of degree 3 or less, and approximate for other functions. The error in the approximation depends on the extent to which f can be well approximated by a polynomial of degree 3. ■

This basic idea generalizes to larger values of m.

Now, consider the more complex task of integrating a multidimensional function $f(Z_1, \ldots, Z_d)$. One approach is to use the Gaussian quadrature grid in each dimension, giving rise to a d-dimensional grid with m^d points. We can then evaluate the function at each of the grid points, and combine the evaluations together using the appropriate weights. Viewed slightly differently, this approach computes the d-dimensional integral recursively, computing, for each point in dimension i, the Gaussian-quadrature approximation of the integral up to dimension $i-1$. This integration rule is accurate for any polynomial which is a sum of monomial terms, each of the form $\prod_{i=1}^{d} z_i^{a_i}$, where each $a_i \leq 2m-1$. Unfortunately, this grid grows exponentially with d, which can be prohibitive in certain applications.

unscented transformation

exact monomials

Unscented Transformation An alternative approach, called the *unscented transformation*, is based on the integration method of *exact monomials*. This approach uses grids designed specifically for Gaussians over $\mathbb{R}^d$. Intuitively, it uses the symmetry of the Gaussian around its axes to reduce the density of the required grid.

The simplest instance of the exact monomials framework uses $2d+1$ points, as compared to the 2^d points required for the rule derived from Gaussian quadrature. To apply this transformation, it helps to assume that $p_o(\boldsymbol{Z})$ is a standard Gaussian $p_0(\boldsymbol{Z}) = \mathcal{N}(\boldsymbol{Z} \mid \mathbf{0}; I)$ where I is the identity matrix. In cases where p_0 is not of that form, so that $p_0(\boldsymbol{Z}) = \mathcal{N}(\boldsymbol{Z} \mid \boldsymbol{\mu}; \Sigma)$, we can do the following change of variable transformation: Let A be the *matrix square root* of Σ, that is, A is a $d \times d$ matrix such that $\Sigma = A^T A$. We define $\tilde{p}_0(\boldsymbol{Z}') = \mathcal{N}(\boldsymbol{Z}' \mid \mathbf{0}; I)$. We can now show that

$$p_0(\boldsymbol{Z}) = \tilde{p}_0(\boldsymbol{Z}') \bigoplus [\boldsymbol{Z} = A\boldsymbol{Z}' + \boldsymbol{\mu}]. \tag{14.17}$$

We can now perform a change of variables for each of our functions, defining $\tilde{f}_i(\boldsymbol{Z}) = f_i(A\boldsymbol{Z} + \boldsymbol{\mu})$, and perform our moment computation relative to the functions $\tilde{f}_i$ rather than f_i.

Now, for $i = 1, \ldots, d$, let $\boldsymbol{z}_i^+$ be the point in $\mathbb{R}^d$ which has $z_i = +1$ and $z_j = 0$ for all $j \neq i$. Similarly, let $\boldsymbol{z}_i^- = -\boldsymbol{z}_i^+$. Let $\lambda \neq 0$ be any number. We then use the following integration rule:

$$\int_{-\infty}^{\infty} W(\boldsymbol{z}) f(\boldsymbol{z}) d\boldsymbol{z} \approx \left(1 - \frac{d}{\lambda^2}\right) f(\mathbf{0}) + \sum_{i=1}^{d} \frac{1}{2\lambda^2} f(\lambda \boldsymbol{z}_i^+) + \sum_{i=1}^{d} \frac{1}{2\lambda^2} f(\lambda \boldsymbol{z}_i^-). \tag{14.18}$$

In other words, we evaluate f at the mean of the Gaussian, $\mathbf{0}$, and then at every point which is $\pm\lambda$ away from the mean for one of the variables Z_i. We then take a weighted average of these

points, for appropriately chosen weights. Thus, this rule, like Gaussian quadrature, is defined in terms of a set of points $\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{2d}$ and weights $w_0, \ldots, w_{2d}$, so that

$$\int_{-\infty}^{\infty} W(\boldsymbol{z}) f(\boldsymbol{z}) d\boldsymbol{z} = \sum_{i=0}^{2d} w_i f(\boldsymbol{z}_i).$$

unscented Kalman filter

This integration rule is used as the basis for the *unscented Kalman filter* (see section 15.4.1.2).

The method of exact monomials can be used to provide exact integration for all polynomials of degree p or less, that is, to all polynomials where each monomial term $\prod_{i=1}^{d} z_i^{a_i}$ has $\sum_{i=1}^{d} a_i \leq p$. Therefore, the method of exact monomials has precision p. In particular, equation (14.18) provides us with a rule of precision 3. Similar rules exist that achieve higher precision. For example, we can obtain a method of precision 5 by evaluating f at $\mathbf{0}$, at the $2d$ points that are $\pm\lambda$ away from the mean along one dimension, and at the $2d(d-1)$ points that are $\pm\lambda$ away from the mean along two dimensions. The total number of points is therefore $2d^2 + 1$.

Note that the precision-3 rule is less precise than the one obtained by using Gaussian quadrature separately for each dimension: For example, if we combine one-dimensional Gaussian quadrature rules of precision 2, we will get a rule that is also exact for monomials such as $z_1^2 z_2^2$ (but not for the degree 3 monomial z_1^3). However, the number of grid points used in this method is exponentially lower.

The parameter λ is a free parameter. Every choice of $\lambda \neq 0$ results in a rule of precision 3, but different choices lead to different approximations. Small values of λ lead to more local approximations, which are based on the behavior of f near the mean of the Gaussian and are less affected by the higher order terms of f.

14.4.1.3 Discussion

We have suggested several different methods for approximating q as a Gaussian distribution. What are the trade-offs between them? We begin with two examples.

Example 14.13

Figure 14.7(top) illustrates the two different approximations in comparison to the optimal approximation (the correct mean and covariance) obtained by sampling. We can see that the unscented transformation is almost exact, whereas the linearization method makes significant errors in both mean and covariance.

The bottom row provides a more quantitative analysis for the simple nonlinear function $Y = \sqrt{(\sigma Z_1)^2 + (\sigma Z_2)^2}$. *The left panel presents results for* $\sigma = 2$, *showing the optimal Gaussian M-projection and the approximations using three methods: Taylor series, exact monomials with precision 3, and exact monomials with precision 5. The "optimal" approximation is estimated using a very accurate Gaussian quadrature rule with a grid of* 100×100 *integration points. We can see that the precision-5 rule is very accurate, but even the precision-3 rule is significantly more accurate than the Taylor series. The right panel shows the KL-divergence between the different approximations and the optimal approximation. We see that the quality of approximation of every method degrades as* σ *increases. This behavior is to be expected, since all of the methods are accurate for low-order polynomials, and the larger the* σ, *the larger the contribution of the higher-order terms. For small and medium variances, the Taylor series is the least exact of the three methods. For large variances, the precision 3 rule becomes significantly less accurate. The reason is that for* $\sigma > 0.23$, *the covariance matrices returned by the numerical integration procedure*

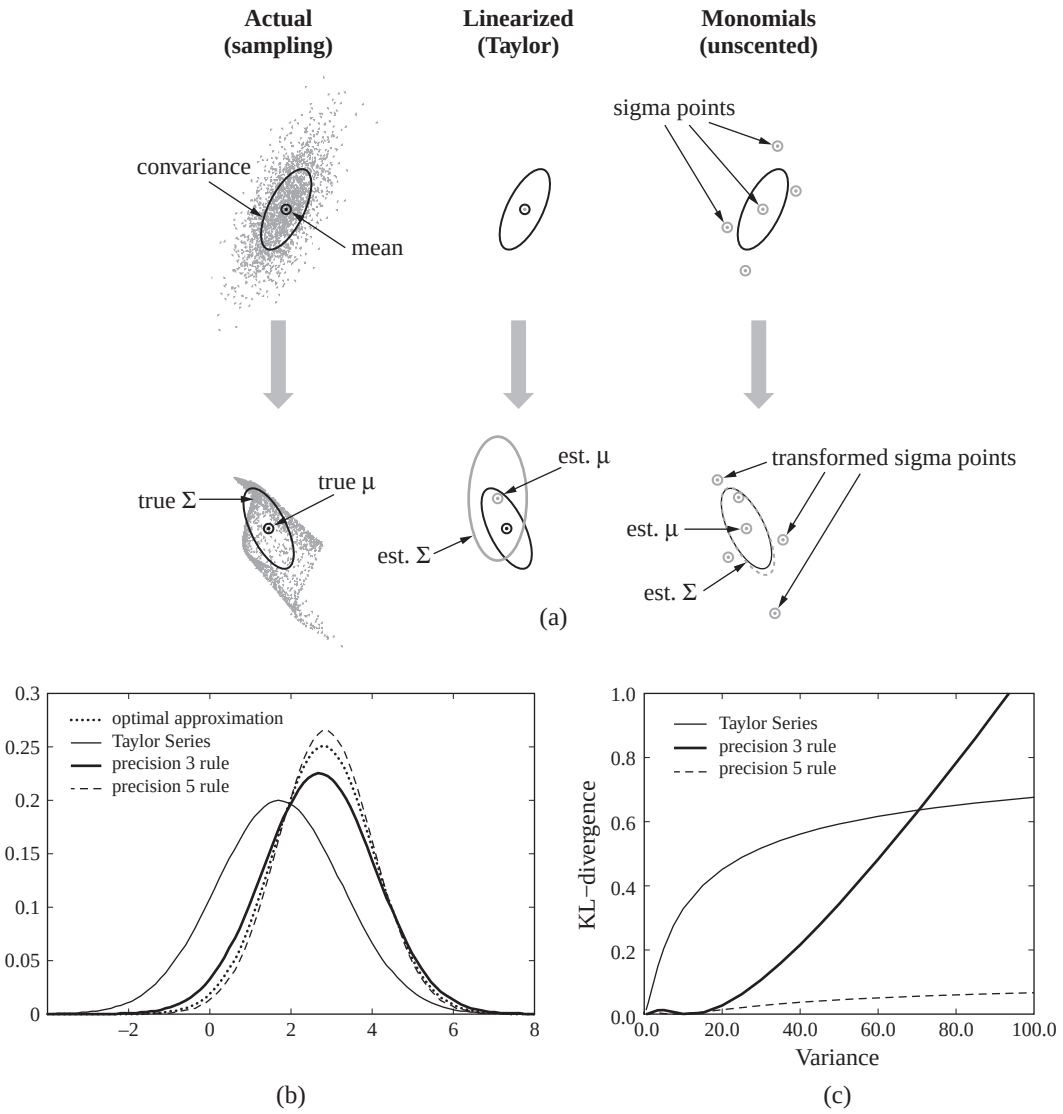


Figure 14.7 Comparisons of different Gaussian approximations for a nonlinear dependency. The top row (adapted with permission from van der Merwe et al. (2000a)) illustrates the process of different approximation methods and the results they obtain; the function being linearized is a feed-forward neural network with random weights. The bottom row shows a more quantitative analysis for the function $f(Z_1, Z_2) = \sqrt{(\sigma Z_1)^2 + (\sigma Z_2)^2}$. The left panel shows the different approximations when $\sigma^2 = 4$, and the right panel the KL-divergence from optimal approximation as a function of σ^2.

are illegal, and must be corrected. The correction produces reasonable answers for values of σ up to $\sigma = 4$, and then degrades. However, it is important to note that, for high variances, the Gaussian approximation is a poor approximation to p in any case, so that the whole approach of using a Gaussian approximation in inference breaks down. For low variances, where the Gaussian approximation is reasonable, even the corrected precision-3 rule significantly dominates the Taylor series approach. ■

From a computational perspective, the Gaussian quadrature method is the most precise, but also the most expensive. In practice, one would only apply it in cases where precision was of critical important and the dimension was very low. The cost of the other two methods depends on the function f that we are trying to linearize. The linearization method (equation (14.13)) requires that we evaluate f and each of f's d partial derivatives at the point $\mathbf{0}$. In cases where the partial derivative functions can be written in closed form, this process requires only $d + 1$ function evaluations. By contrast, the precision 3 method requires $2d + 1$ evaluations of the function f, and the precision 5 method requires $2d^2 + 1$ function evaluations of f. In addition, to use the numerical integration methods we need to convert our distribution to the form of equation (14.17), which is not always required for the Taylor series linearization. Finally, one subtle problem can arise when using numerical integration to perform the M-projection operation: Here, the quantities of equation (14.14)–(14.15) are computed using an approximate procedure. Thus, although for exact integration, the covariance matrix defined by these equations is guaranteed to be positive definite; this is not the case for the approximate quantities, where the approximation may give rise to a matrix that is not positive definite. See section 14.7 for references to a modified approach that avoids this problem.

Putting computational costs aside, **the requirement in the linearization approach of computing the gradient may be a significant issue in some settings. Some functions may not be differentiable (for example, the $\max$ function), preventing the use of the Taylor series expansion. Furthermore, even if f is differentiable, computing its gradient may still be difficult. In some applications, f might not even be given in a parametric closed form; rather, it might be implemented as a lookup table, or as a function in some programming language. In such cases, there is no simple way to compute the derivatives of the Taylor series expansion, but it is easy to evaluate f on a given point, as required for the numerical integration approach.**

14.4.2 Expectation Propagation with Gaussian Approximation

The preceding section developed the basic tool of approximating a distribution as a Gaussian. We now show how these methods can be used to perform expectation propagation message passing inference in nonlinear graphical models. Roughly speaking, at each step, we take all of the factors that need to be multiplied, and we approximate as a Gaussian the measure derived by multiplying all of these factors. We can then use this Gaussian distribution to form the desired message.

For the application of this method, we assume that each of the factors in our original distribution Φ defines a conditional distribution over a single random variable in terms of others. This assumption certainly holds in the case of standard Bayesian networks, where all CPDs are (by definition) of this form. It is also the case for many Markov networks, especially in the continuous case. We further assume that each of the factors in Φ can be written as a

deterministic function:

$$X_i = f_i(\boldsymbol{Y}_i, \boldsymbol{W}_i), \tag{14.19}$$

where $\boldsymbol{Y}_i \subseteq \mathcal{X}$ are the model variables on which X_i's CPD depends, and $\boldsymbol{W}_i$ are "new" standard Gaussian random variables that capture all of the stochasticity in the CPD of X_i. We call these W's *exogenous* variables, since they capture stochasticity that is outside the model. (See also section 21.4.)

Although there are certainly factors that do not satisfy these assumptions, this representational class is quite general, and encompasses many of the factors used in practical applications. Most obviously, using the transformation of equation (14.17), this representation encompasses any Gaussian distribution. However, it also allows us to represent many nonlinear dependencies:

Example 14.14 *Consider the nonlinear CPD $X \sim \mathcal{N}\left(\sqrt{Y_1^2 + Y_2^2}; \sigma^2\right)$. We can reformulate this CPD in terms of a deterministic, nonlinear function, as follows: We introduce a new exogenous variable W that captures the stochasticity in the CPD. We then define $X = f(Y_1, Y_2, W)$ where $f(Y_1, Y_2, W) = \sqrt{Y_1^2 + Y_2^2} + \sigma W$.* ■

In this example, as in many real-world CPDs, the dependence of X on the stochastic variable W is linear. However, the same idea also applies in cases where the variance is a more complex function:

Example 14.15 *Returning again to example 5.20, assume we want the variance of the vehicle's position at time $t+1$ to depend on its time t velocity, so that we have more uncertainty about the vehicle's next position if it is moving faster. Thus, for example, we might want to encode a distribution $\mathcal{N}\left(X' \mid X + V; \rho V^2\right)$. We can do so by introducing an exogenous standard Gaussian variable Z and defining*

$$X' = X + V + \sqrt{\rho} V Z.$$

It is not difficult to verify that X' has the appropriate Gaussian distribution. ■

We now apply the EP framework of section 11.4.3.2. As there, we maintain the potential at each cluster $\boldsymbol{C}_i$ as a factor set $\vec{\phi}_i$; some of those factors are initial factors, whereas others are messages sent to cluster $\boldsymbol{C}_i$. Our initial potentials are all of the form equation (14.19), and since we project all messages into the Gaussian parametric family, all of the incoming messages are Gaussians, which we can reformulate as a standard Gaussian and a set of deterministic functions.

In principle, it now appears that we can take all of the incoming messages, along with all of the exogenous Gaussian variables $\boldsymbol{W}_i$, to produce a single Gaussian distribution p_0. We can then apply the linearization procedures of section 14.4.1 to obtain a Gaussian approximation. However, a closer examination reveals several important subtleties that must be addressed.

14.4.2.1 Dealing with Evidence

Our discussion so far has sidestepped the issue of evidence: nowhere did we discuss the place at which evidence is instantiated into the factors. In the context of discrete variables, this issue

was resolved in a straightforward way by restricting the factors (as we discussed in section 9.3.2). This restriction could be done at any time during the course of the message passing algorithm (as long as the observed variable is never eliminated).

In the context of continuous variables, the situation is more complex, since any assignment to a variable induces a density over a subspace of measure 0. Thus, when we observe $X = x$, we must a priori restrict all factors to the space where $X = x$. This operation is straightforward in the case of canonical forms, but it is somewhat subtler for nonlinear functions. For example, consider the nonlinear dependence of example 14.14. If we have evidence $Y_1 = y_1$, we can easily redefine our model as $X = g(Y_2, W)$ where $g(Y_2, W) = \sqrt{y_1^2 + Y_2^2} + \sigma W$. However, it is not so clear what to do with evidence on the dependent variable X.

The simplest solution to this problem, and one which is often used in practice, is to instantiate "downstream" the evidence in a cluster after the cluster is linearized. That is, in the preceding example, we would first linearize the function f in its cluster, resulting in a Gaussian distribution $p(X, Y_1, Y_2)$; we would then instantiate the evidence $X = x$ in the canonical form associated with this distribution, to obtain a new canonical form that is proportional to $p(Y_1, Y_2 \mid x)$.

This approach is simple, but it can be very inaccurate. In particular, the linearization operation (no matter how it is executed) depends on the distribution p_0 relative to which the linearization is performed. Our posterior distribution, given the evidence, can be very different from the prior distribution p_0, leading to a very different linearization of f. Better methods for taking the evidence into account during the linearization operation exist, but they are outside the scope of this book.

14.4.2.2 Valid Linearization

A second key issue is that all of the linearization procedures described earlier require that p_o be a Gaussian distribution, and not a general canonical form. This requirement is a significant one, and imposes constraints on one or more of: the structure of the cluster graph, the order in which messages are passed, and even the probabilistic model itself.

Example 14.16 *Most simply, consider a chain-structured Bayesian network $X_1 \rightarrow X_2 \rightarrow X_3$, where all variables have nonlinear CPDs. We thus have a function $X_1 = f_1(W_1)$ and functions $X_2 = f_2(X_1, W_2)$ and $X_3 = f_3(X_2, W_3)$. In the obvious clique tree, we would have a clique $\boldsymbol{C}_1$ with scope X_1, X_2, containing f_1 and f_2, and a clique $\boldsymbol{C}_2$ with scope X_2, X_3, containing f_3. Further, assume that we have evidence $X_3 = x_3$.*

In the discrete setting, we can simply restrict the factor in $\boldsymbol{C}_2$ with the observation over X_3, producing a factor over X_2. This factor can then be passed as a message to $\boldsymbol{C}_1$. In the nonlinear setting, however, we must first linearize f_3, and for that, we must have a distribution over X_2. But we can only obtain this distribution by multiplying into the factor $p(X_1)$, $p(X_2 \mid X_1)$, and $p(X_3 \mid X_2)$. In other words, in order to deal with $\boldsymbol{C}_2$, we must first pass a message from $\boldsymbol{C}_1$. ■

order-constrained message passing

In general, this requirement on *order constrained* message passing is precisely the same one that we faced for CLG distributions in section 14.3.3.1, with the same consequences. In a Bayesian network, this requirement constrains us to pass messages in a topological order. In other words, before we linearize a function $X_i = f_i(\mathrm{Pa}_{X_i}, \boldsymbol{W}_i)$, we must first linearize and obtain a Gaussian for every $Y_j \in \mathrm{Pa}_{X_i}$.

Example 14.17 *Consider the network of figure 14.6, but where we now assume that all variables (including A, B, C) are continuous and utilize nonlinear CPDs. As in example 14.7, the clique tree of figure 14.6c does not allow messages to be passed at all, since none of the cliques respect the ordering constraint. The clique tree of (e) does allow messages to be passed, but only if the $\{A, B, X, Y\}$ clique is the first to act, passing a message over A, Y to the clique $\{A, C, Y, Z\}$; this message defines a distribution over the parents of C and Z, allowing them to be linearized.* ■

In the context of Markov networks, we again can partially circumvent the problem if we assume that the factors in the network are all factor normalizable. However, this requirement may not always hold in practice.

14.4.2.3 Linearization of Multiple Functions

A final issue, related to the previous one, arises from our assumption in section 14.4.1 that all of the functions in a cluster depend only on a set of standard Gaussian random variables $\boldsymbol{Z}$. This assumption is overly restrictive in almost all cluster graphs.

Example 14.18 *Consider again our chain-structured Bayesian network $X_1 \rightarrow X_2 \rightarrow X_3$ of example 14.16. Here, $\boldsymbol{C}_1$ has scope X_1, X_2, and contains both f_1 and f_2. This structure does not satisfy the preceding requirements, since X_2 relies also on X_1.*

incremental linearization

There are two ways to address the issue in this case. The first, which we call incremental linearization *first linearizes f_1, and subsequently linearize f_2. This approach can be implemented by making a separate clique containing just X_1. In this clique, we have a Gaussian distribution $p_1(Z_1)$, and so we can compute a Gaussian approximation to $f_1(Z_1)$, producing a Gaussian message $\tilde{\delta}_{1\rightarrow 2}(X_1)$. We can then pass this message to a clique containing X_1, X_2, but only the $f_2(X_2, Z_2)$ function; we now have a Gaussian distribution $p_2(X_2, Z_2)$, and can use the techniques of section 14.4.1 to linearize $f_2(X_2, Z_2)$ into this Gaussian, to produce a Gaussian distribution over X_1, X_2. (Note that $p_2(X_2, Z_2)$ is not a standard Gaussian, but we can use the trick of equation (14.17) to change the coordinate system; see exercise 14.7.) We can then marginalize the resulting Gaussian distribution onto X_2, and send a message $\tilde{\delta}_{2\rightarrow 3}(X_2)$ to $\boldsymbol{C}_3$, continuing this process.*

simultaneous linearization

As a second alternative, called simultaneous linearization*, we can linearize multiple nonlinear functions into the same cluster together. We can implement this solution by substituting the variable X_1 with its functional definition; that is, we can define $X_2 = g_2(Z_1, Z_2) = f_2(f_1(Z_1), Z_2)$, and use g_2 rather than f_2 in the analysis of section 14.4.1.* ■

Both of these solutions generalize beyond this simple example. In the incremental approach, we simply define smaller clusters, each of which contains at most one nonlinear function. We then linearize one such function at a time, producing each time a Gaussian distribution. This Gaussian is passed on to another cluster, and used as the basis for linearizing another nonlinear function. The simultaneous approach linearizes several functions at once, substituting each variable with the function that defines it, so as to make all the functions depend only on Gaussian variables.

These two approaches make different approximations: Going back to example 14.16, in the incremental approach, the approximation of f_2 uses a Gaussian approximation to the distribution

over X_1. If this approximation is poor, it may lead to a poor approximation for the distribution over X_2. Conversely, in the simultaneous approach, we are performing an integral for the function g_2, which may be more complicated than f_2, possibly leading to a poor approximation of its moments. In general, the errors made by each method are going to depend on the specific case at hand, and neither necessarily dominates the other.

14.4.2.4 Iterated Message Passing

The general algorithm we described can be applied within the context of different message passing schemes. Most simply, we define a clique tree, and we then do a standard upward and downward pass. However, as we discussed in the context of the general expectation propagation algorithm, our approximation within a cluster depends on the contents of that factor. In our setting, the approximation of $p \bigoplus f$ as a Gaussian depends on the distribution p. Depending on the order in which CPDs are linearized and messages are passed, the resulting distribution might be very different.

Example 14.19 *Consider a network consisting of a variable X and its two children Y and Z, so that our two cliques are $\boldsymbol{C}_1 = \{X, Y\}$ and $\boldsymbol{C}_2 = \{X, Z\}$. Assume that we observe $Y = y$. If we include the prior $p(X)$ within $\boldsymbol{C}_1$, then we first compute $\boldsymbol{C}_1$'s message, producing a Gaussian approximation to the joint posterior $\hat{p}(X, Y)$. This posterior is marginalized to produce $\hat{p}(Y)$, which is then used as the basis for linearizing Z. Conversely, if we include $p(X)$ within $\boldsymbol{C}_2$, then the linearization of Z is done based on an approximation to X's prior distribution, generally leading to a very different linearization for $p(X, Z)$.* ■

In general, the closer the distribution used for the approximation is to the true posterior, the higher the quality of the approximation. Thus, in this example, we would prefer to first linearize Y and condition on its value, and only then to linearize Z. However, we cannot usually linearize all CPDs using the posterior. For example, assume that Z has another child W whose value is also observed; the constraint of using a topological ordering prevents us from linearizing W before linearizing Z, so we are forced to use a distribution over X that does not take into account the evidence on W.

The iterative EP algorithm helps address this limitation. In particular, once we linearize all of the initial clusters (subject to the constraints we mentioned), we can now continue to pass messages between the clusters using the standard belief update algorithm. At any point in the algorithm where cluster $\boldsymbol{C}_i$ must send a message (whether at every message or on a more intermittent basis), it can use the Gaussian defined by the incoming messages and the exogenous Gaussian variables to relinearize the functions assigned to it. The revised messages arising from this new linearization are then sent to adjacent clusters, using the standard EP update rule of equation (11.41) (that is, by subtracting the sufficient statistics of the previously sent message). This generally has the effect of linearizing using a distribution that is closer to the posterior, and hence leading to improved accuracy. However, it is important to remember that, as in section 14.3.3.2, the messages that arise in belief-update message passing may not be positive definite, and hence can give rise to canonical forms that are not legal Gaussians, and for which integration is impossible. To avoid catastrophic failures in the algorithm, it is important to check that any canonical form used for integration is, indeed, a normalizable Gaussian.

14.4.2.5 Augmented CLG Networks

One very important consequence of this algorithm is its ability to address one of the main limitations of CLG networks — namely, that discrete variables cannot have continuous parents. This limitation is a significant one. For example, it prevents us from modeling simple objects such as a thermostat, whose discrete on/off state depends on a continuous temperature variable. As we showed, we can easily model such dependencies, using, for example, a linear sigmoid model or its multinomial extension. The algorithm described earlier, which accommodates nonlinear CPDs, easily deals with this case.

Example 14.20

Consider the network $X \rightarrow A$, where X has a Gaussian distribution given by $p(X) = \mathcal{N}(X \mid \mu; \sigma)$ and the CPD of A is a softmax given by $P(A = a^1 \mid X = x) = 1/(1 + e^{c_1 x + c_0})$. The clique tree has a single clique (X, A), whose potential should contain the product of these two CPDs. Thus, it should contain two entries, one for a^1 and one for a^0, each of which is a continuous function: $p(x)P(a^1 \mid x)$ and $p(x)P(a^0 \mid x)$. As before, we approximate this potential using a canonical table, with an entry for every assignment to A, and where each entry is a Gaussian distribution. Thus, we would approximate each entry $p(x)P(a \mid x)$ as a Gaussian. Once again, we have a product of a function — $P(a \mid x)$ — with a Gaussian — $p(x)$. We can therefore use any of the methods in section 14.4.1 to approximate the result as a single Gaussian. ■

This simple extension forms the basis for a general algorithm that deals with hybrid networks involving both continuous and discrete variables; see exercise 14.11.

14.5 Particle-Based Approximation Methods

All of the message passing schemes described before now utilize the Gaussian distribution as a parametric representation for the messages and (in some cases) even the clique potentials. This representation allows for truly exact inference only in the case of pure Gaussian networks, and it is exact in a weak sense for a very restricted class of CLG networks. If our original factors are far from being Gaussian, and, most particularly if they are multimodal, the Gaussian approximation used in these algorithms can be a very poor one. Unfortunately, there is no general-purpose parametric class that allows us to encode arbitrarily continuous densities.

An alternative approach is to use a semiparametric or nonparametric method, which allows us to avoid parametric assumptions that may not be appropriate in our setting. Such approaches are often applied in continuous settings, since there are many settings where the Gaussian approximation is clearly inappropriate. In this section, we discuss how such methods can be used in the context of inference in graphical models.

14.5.1 Sampling in Continuous Spaces

We begin by discussing the basic task of sampling from a continuous univariate measure. As we discussed in box 12.A, sampling from a discrete distribution can be done using straightforward methods. Efficient solutions, albeit somewhat more complex, are also available for other parametric forms, including Gaussians, Gamma distributions, and others; see section 14.7. Unfortunately, such solutions do not exist for all continuous densities. In fact, even distributions that can be characterized analytically may not have established sampling procedures.

rejection sampling

A number of general-purpose approaches have been designed for sampling from arbitrary continuous densities. Perhaps the simplest is *rejection sampling* (or sometimes acceptance-rejection sampling). The basic idea is to sample a value from a proxy distribution and then "accept" it with probability that is proportional to the skew introduced by the proxy distribution. Suppose that we want to sample a variable X with density $p(x)$. Now suppose that we have a function $q(x)$ such that $q(x) > 0$ whenever $p(x) > 0$ and $q(x) \geq p(x)$. Moreover, suppose that we can sample from the distribution

$$Q(X) = \frac{1}{Z} q(x),$$

where Z is a normalizing constant. (Note that $Z > 1$, as $q(x)$ is an upper bound on a density function whose integral is 1.) In this case, we repeatedly perform the following steps, until acceptance:

1. Sample $x \sim Q(x)$, $u \sim \text{Unif}([0, 1])$.
2. If $u < \frac{p(x)}{q(x)}$ return x.

The probability that this procedure returns the value x is exactly $p(x)$. The proof is analogous to the one we saw for importance sampling (see proposition 12.1). (The difference here is that we need to return a single sample rather than weight multiple samples.) In practice, we can speed up the procedure if we have also a lower bound $b(x)$ for $p(x)$ (that is $b(x) \leq p(x)$). In that case, if $u < \frac{b(x)}{q(x)}$ than we accept the sample x without evaluating $p(x)$. By finding easy to evaluate functions $q(x)$ and $b(x)$ that allow sampling from Q, and provide relatively tight bounds on the density $p(x)$, this method can accept samples within few iterations.

This straightforward approach, however, can be very slow if $q(x)$ is not a good approximation to $p(x)$. Fortunately, more sophisticated methods have also been designed. One example is described in exercise 12.22. See section 14.7 for other references.

14.5.2 Forward Sampling in Bayesian Networks

We now consider the application of forward sampling in a Bayesian network (algorithm 12.1). Assuming that we have a method for generating samples from the continuous density of each variable X_i given an assignment to its parents, we can use this method to generate samples from a broad range of continuous and hybrid Bayesian networks — far greater than the set of networks for which message passing algorithms are applicable. This approach generates random samples from the prior distribution $p(\mathcal{X})$. As we discussed, these samples can be used to estimate the expectation of any function $f(\mathcal{X})$, as in equation (12.1). For example, we can compute the mean $\boldsymbol{E}[X]$ of any variable X by taking $f(\mathcal{X}) = X$, or its variance by taking $f(\mathcal{X}) = X^2$ and subtracting $\boldsymbol{E}[X]^2$.

However, plain forward sampling is generally inapplicable to hybrid networks as soon as we have evidence over some of the network variables. Consider a simple network $X \rightarrow Y$, where we have evidence $Y = y$. Forward sampling generates $(x[m], y[m])$ samples, and rejects those in which $y[m] \neq y$. Because the domain of Y is a continuum, the probability that our sampling process will generate any particular value y is zero. Thus, none of our samples will match the observations, and all will be rejected.

A more relevant approach in the case of hybrid networks is a form of importance sampling, such as the likelihood weighting algorithm of algorithm 12.2. The likelihood weighting algorithm generalizes to the hybrid case in the most obvious way: Unobserved variables are sampled, based on the assignment to their parents, as described earlier. Observed variables are instantiated to their observed values, and the sample weight is adjusted. We can verify, using the same analysis as in section 12.2, that the appropriate adjustment to the importance weight for a continuous variable X_i observed to be x_i is the density $p(x_i \mid \boldsymbol{u}_i)$, where $\boldsymbol{u}_i$ is the assignment, in the sample, to Pa_{X_i}. The only implication of this change is that, unlike in the discrete case, this adjustment may be greater than 1 (because a density function is not bounded in the range $[0, 1]$). Although this difference does not influence the algorithm, it can increase the variance of the sample weights, potentially decreasing the overall quality of our estimator.

14.5.3 MCMC Methods

MCMC

The second type of sampling algorithms is based on *Markov chain Monte Carlo* (MCMC) methods. In general, the theory of Markov chains for continuous spaces is quite complicated, and many of the basic theorems that hold for discrete spaces do not hold, or hold only under certain assumptions. A discussion of these issues is outside the scope of this book. However, simple variants of the MCMC algorithms for graphical models continue to apply in this setting.

Gibbs sampling

The *Gibbs sampling* algorithm (algorithm 12.4) can also be applied to the case of hybrid networks. As in the discrete case, we maintain a sample, which is a complete assignment to the network variables. We then select variables X_i one at a time, and sample a new value for X_i from $p(X_i \mid \boldsymbol{u}_i)$, where $\boldsymbol{u}_i$ is the current assignment to X_i's Markov blanket. The only issue that might arise relates to this sampling step. In equation (12.23), we showed a particularly simple form for the probability of $X_i = x_i'$ given the assignment $\boldsymbol{u}_i$ to X_i's. The same derivation applies to the continuous case, simply replacing the summation with an integral:

$$p(x_i' \mid \boldsymbol{u}_i) = \frac{\prod_{\boldsymbol{C}_j \ni X_i} \beta_j(x_i', \boldsymbol{u}_i)}{\int_{x_i''} \prod_{\boldsymbol{C}_j \ni X_i} \beta_j((x_i'', \boldsymbol{u}_i))},$$

where $\boldsymbol{C}_j$ are the different potentials in the network.

In the discrete case, we could simply multiply the relevant potentials as tables, renormalize the resulting factor, and use it as a sampling distribution. In the continuous case, matters may not be so simple: the product of factors might not have a closed form that allows sampling.

Metropolis-Hastings

In this case, we generally resort to the use of a *Metropolis-Hastings* algorithm. Here, as in the discrete case, changes to a variable X_i are proposed by sampling from a local proposal distribution $\mathcal{T}_i^Q$. The proposed local change is either accepted or rejected using the appropriate acceptance probability, as in equation (12.26):

$$\mathcal{A}(\boldsymbol{u}_i, x_i \to \boldsymbol{u}_i, x_i') \quad = \quad \min\left[1, \frac{p(x_i' \mid \boldsymbol{u}_i)}{p(x_i \mid \boldsymbol{u}_i)} \frac{\mathcal{T}^Q(\boldsymbol{u}_i, x_i' \to \boldsymbol{u}_i, x_i)}{\mathcal{T}^Q(\boldsymbol{u}_i, x_i \to \boldsymbol{u}_i, x_i')}\right],$$

where (as usual) probabilities are replaced with densities.

The only question that needs to be addressed is the choice of the proposal distribution. One common choice is a Gaussian or student-t distribution centered on the current value x_i. Another is a uniform distribution over a finite interval also centered on x_i. These proposal

random-walk chain

distributions define a random walk over the space, and so a Markov chain that uses such a proposal distribution is often called a *random-walk chain*. These choices all guarantee that the Markov chain converges to the correct posterior, as long as the posterior is positive: for all $\boldsymbol{z} \in Val(\mathcal{X} - \boldsymbol{E})$, we have that $p(\boldsymbol{z} \mid \boldsymbol{e}) > 0$. However, the rate of convergence can depend heavily on the window size: the variance of the proposal distribution. Different distributions, and even different regions within the same distribution, can require radically different window sizes. Picking an appropriate window size and adjusting it dynamically are important questions that can greatly impact performance.

14.5.4 Collapsed Particles

As we discussed in section 12.4, it is difficult to cover a large state space using full instantiations to the network variables. Much better estimates — often with much lower variance — can be obtained if we use collapsed particles. Recall that, when using collapsed particles, the variables $\mathcal{X}$ are partitioned into two subsets $\mathcal{X} = \boldsymbol{X}_p \cup \boldsymbol{X}_d$. A collapsed particle then consists of an instantiation $\boldsymbol{x}_p \in Val(\boldsymbol{X}_p)$, coupled with some representation of the distribution $P(\boldsymbol{X}_d \mid \boldsymbol{x}_p, \boldsymbol{e})$. The use of such particles relies on our ability to do two things: to generate samples from $\boldsymbol{X}_p$ effectively, and to represent compactly and reason with the distribution $P(\boldsymbol{X}_d \mid \boldsymbol{x}_p, \boldsymbol{e})$.

The notion of collapsed particles carries over unchanged to the hybrid case, and virtually every algorithm that applied in the discrete case also applies here. Indeed, collapsed particles are often particularly suitable in the setting of continuous or hybrid networks: In many such networks, if we select an assignment to some of the variables, the conditional distribution over the remaining variables can be represented (or well approximated) as a Gaussian. Since we can efficiently manipulate Gaussian distributions, it is generally much better, in terms of our time/accuracy trade-off, to try and maintain a closed-form Gaussian representation for the parts of the distribution for which such an approximation is appropriate.

Although this property can be usefully exploited in a variety of networks, one particularly useful application of collapsed particles is motivated by the observation that inference in a purely continuous network is fairly tractable, whereas inference in the simplest hybrid networks — polytree CLGs — can be very expensive. Thus, if we can "erase" the discrete variables from the network, the result is a much simpler, purely continuous network, which can be manipulated using the methods of section 14.2 in the case of linear Gaussians, and the methods of section 14.4 in the more general case.

Thus, CLG networks are often effectively tackled using a collapsed particles approach, where the instantiated variables in each particular are the discrete variables, $\boldsymbol{X}_p = \Delta$, and the variables maintained in a closed-form distribution are the continuous variables, $\boldsymbol{X}_d = \Gamma$. We can now apply any of the methods described in section 12.4, often with some useful shortcuts. As one example, likelihood weighting can be easily applied, since discrete variables cannot have continuous parents, so that the set $\boldsymbol{X}_p$ is upwardly closed, allowing for easy sampling (see exercise 14.12). The application of MCMC methods is also compelling in this case, and it can be made more efficient using incremental update methods such as those of exercise 12.27.

14.5.5 Nonparametric Message Passing

Yet another alternative is to use a hybrid approach that combines elements from both particle-based and message passing algorithms. Here, the overall algorithm uses the structure of a message passing (clique tree or cluster graph) approach. However, we use the ideas of particle-based inference to address the limitations of using a parametric representation for the intermediate factors in the computation. Specifically, rather than representing these factors using a single parametric model, we encode them using a nonparametric representation that allows greater flexibility to capture the properties of the distribution. Thus, we are essentially still reparameterizing the distribution in terms of a product of cluster potentials (divided by messages), but each cluster potential is now encoded using a nonparametric representation.

The advantage of this approach over the pure particle-based methods is that the samples are generated in a much lower-dimensional space: single cluster, rather than the entire joint distribution. This can alleviate many of the issues associated with sampling in a high-dimensional space. On the other side, we might introduce additional sources of error. In particular, if we are using a loopy cluster graph rather than a clique tree as the basis for this algorithm, we have all of the same errors that arise from the representation of a distribution as a set of pseudo-marginals.

One can construct different instantiations of this general approach, which can vary along several axes. The first is the message passing algorithm used: clique tree versus cluster graph, sum-product versus belief update. The second is the form of the representation used for factors and messages: plain particles; a nonparametric density representation; or a semiparametric representation such as a histogram. Finally, we have the approach used to approximate the factor in the chosen representation: importance sampling, MCMC, or a deterministic approximation.

14.6 Summary and Discussion

In this chapter, we have discussed some of the issues arising in applying inference to networks involving continuous variables. While the semantics of such networks is easy to define, they raise considerable challenges for the inference task.

The heart of the problem lies in the fact that we do not have a universal representation for factors involving continuous variables — one that is closed under basic factor operations such as multiplication, marginalization, and restriction with evidence. This limitation makes it very difficult to design algorithms based on variable elimination or message passing. Moreover, the difficulty is not simply a matter of our inability to find good algorithms. Theoretical analysis shows that classes of network structures that are tractable in the discrete case (such as polytrees) give rise to $\mathcal{NP}$-hard inference problems in the hybrid case.

Despite the difficulties, continuous variables are ubiquitous in practice, and so significant work has been done on the inference problem for such models.

Most message passing algorithms developed in the continuous case use Gaussians as a lingua franca for factors. This representation allows for exact inference only in a very limited class of models: clique trees for linear Gaussian networks. However, the exact same factor operations provide a basis for a belief propagation algorithm for Gaussian networks. This algorithm is easy to implement and has several satisfying guarantees, such as guaranteed convergence under fairly weak conditions, producing the exact mean if convergence is achieved. This technique allows us to perform inference in Gaussians where manipulating the full covariance matrix is intractable.

In cases where not all the potentials in the network are Gaussians, the Gaussian representation is generally used as an approximation. In particular, a standard approach uses an instantiation of the expectation propagation algorithm, using M-projection to approximate each non-Gaussian distribution as a Gaussian during message passing. In CLG networks, where factors represent a mixture of (possibly exponentially many) Gaussians derived from different instantiations of discrete variables, the M-projection is used to collapse the mixture components into a single Gaussian. In networks involving nonlinear dependencies between continuous variables, or between continuous and discrete variables, the M-projection involves linearization of the nonlinear dependencies, using either a Taylor expansion or numerical integration. While simple in principle, this application of EP raises several important subtleties. In particular, the M-projection steps can only be done over intermediate factors that are legal distributions. This restriction imposes significant constraints on the structure of the cluster graph and on the order in which messages can be passed. **Finally, we note that the Gaussian approximation is good in some cases, but can be very poor in others. For example, when the distribution is multimodal, the Gaussian M-projection can be a very broad (perhaps even useless) agglomeration of the different peaks.**

This observation often leads to the use of approaches that use a nonparametric approximation. Commonly used approaches include standard sampling methods, such as importance sampling or MCMC. Even more useful in some cases is the use of collapsed particles, which avoid sampling a high-dimensional space in cases where parts of the distribution can be well approximated as a Gaussian. Finally, there are also useful methods that integrate message passing with nonparametric approximations for the messages, allowing us to combine some of the advantages (and some of the disadvantages) of both types of approaches.

Our presentation in this chapter has only briefly surveyed a few of the key ideas related to inference in continuous and hybrid models, focusing mostly on the techniques that are specifically designed for graphical models. Manipulation of continuous densities is a staple of statistical inference, and many of the techniques developed there could be applied in this setting as well. For example, one could easily imagine message passing techniques that use other representations of continuous densities, or the use of other numerical integration techniques. Moreover, the use of different parametric forms in hybrid networks tends to give rise to a host of "special-case" models where specialized techniques can be usefully applied. In particular, even more so than for discrete models, it is likely that a good solution for a given hybrid model will require a combination of different techniques.

14.7 Relevant Literature

Perhaps the earliest variant of inference in Gaussian networks is presented by Thiele (1880), who defined what is the simplest special case of what is now known as the Kalman filtering algorithm (Kalman 1960; Kalman and Bucy 1961). Shachter and Kenley (1989) proposed the general idea of network-based probabilistic models, in the context of Gaussian influence diagrams. The first presentation of the general elimination algorithm for Gaussian networks is due to Normand and Tritchler (1992). However, the task of inference in pure Gaussian networks is highly related to the basic mathematical problem of solving a system of linear equations, and the elimination-based inference algorithms very similar to Gaussian elimination for solving such systems. Indeed, some

of the early incarnations of these algorithms were viewed from that perspective; see Parter (1961) and Rose (1970) for some early work along those lines.

Iterative methods from linear algebra (Varga 2000) can also be used for solving systems of linear equations; in effect, these methods employ a form of local message passing to compute the marginal means of a Gaussian distribution. Loopy belief propagation methods were first proposed as a way of also estimating the marginal variances. Over the years, multiple authors (Rusmevichientong and Van Roy 2001; Weiss and Freeman 2001a; Wainwright et al. 2003a; Malioutov et al. 2006) have analyzed the convergence and correctness properties of Gaussian belief propagation, for larger and larger classes of models. All of these papers provide conditions that ensure convergence for the algorithm, and demonstrate that if the algorithm converges, the means are guaranteed to be correct. The recent analysis of Malioutov et al. (2006) is the most comprehensive; they show that their sufficient condition, called *walk-summability*, is equivalent to pairwise normalizability, and encompasses all of the classes of Gaussian models that were previously shown to be solvable via LBP (including attractive, nonfrustrated, and diagonally dominant models). They also show that the variances at convergence are an underestimate of the true variances, so that the LBP results are overconfident; their results point the way to partially correcting these inaccuracies. The results also analyze LBP for non-walksummable models, relating convergence of the variance to validity of the LBP computation tree.

walk-summability

The properties of conditional Gaussian distributions were studied by Lauritzen and Wermuth (1989). Lauritzen (1992) extended the clique tree algorithm to the task of inference in these models, and showed, for strongly rooted clique trees, the correctness of the discrete marginals and of the continuous means and variances. Lauritzen and Jensen (2001) and Cowell (2005) provided alternative variants of this algorithm, somewhat different representations, which are numerically more stable and better able to handle deterministic linear relationships, where the associated covariance matrix is not invertible. Lerner et al. (2001) extend Lauritzen's algorithm to CLG networks where continuous variables can have discrete children, and provide conditions under which this algorithm also has the same correctness guarantees on the discrete marginals and the moments of the continuous variables. The $\mathcal{NP}$-hardness of inference in CLG networks of simple structures (such as polytrees) was shown by Lerner and Parr (2001). Collapsed particles have been proposed by several researchers as a successful alternative to full collapsing of the potentials into a single Gaussian; methods include random sampling over particles (Doucet et al. 2000; Paskin 2003a), and deterministic search over the particle assignment (Lerner et al. 2000; Lerner 2002), a method particularly suitable in applications such as fault diagnosis, when the evidence is likely to be of low probability.

The idea of adaptively modifying the approximation of a continuous cluster potential during the course of message passing was first proposed by Kozlov and Koller (1997), who used a variable-resolution discretization approach (a semiparametric approximation). Koller et al. (1999) generalized this approach to other forms of approximate potentials. The expectation propagation algorithm, which uses a parametric approximation, was first proposed by Minka (2001b), who also made the connection to optimizing the energy functional under expectation matching constraints. Opper and Winther (2005) present an alternative algorithm based on a similar idea. Heskes et al. (2005) provide a unifying view of these two works. Heskes and Zoeter (2003) discuss the use of weak marginalization within the generalized belief propagation in a network involving both discrete and continuous variables.

The use of the Taylor series expansion to deal with nonlinearities in probabilistic models is

a key component of the extended Kalman filter, which extends the Kalman filtering method to nonlinear systems; see Bar-Shalom, Li, and Kirubarajan (2001) for a more in-depth presentation of these methods. The method of exact monomials, under the name unscented filter, was first proposed by Julier and Uhlmann (1997). Julier (2002) shows how this approach can be modified to address the problem of producing approximations that are not positive definite.

Sampling from continuous distributions is a core problem in statistics, on which extensive work has been done. Fishman (1996) provides a good overview of methods for various parametric families. Methods for sampling from other distributions include adaptive rejection sampling (Gilks and Wild 1992; Gilks 1992), adaptive rejection metropolis sampling (Gilks et al. 1995) and slice sampling (Neal 2003).

The BUGS system supports sampling for many continuous families, within their general MCMC framework. Several approaches combine sampling with message passing. Dawid et al. (1995); Hernández and Moral (1997); Kjærulff (1995b) propose methods that sample a continuous factor to turn it into a discrete factor, on which standard message passing can be applied. These methods vary on how the samples are generated, but the sampling is performed only once, in the initial message passing step, so that no adaptation to subsequent information is possible.

nonparametric belief propagation

Sudderth et al. (2003) propose *nonparametric belief propagation*, which uses a nonparametric approximation of the potentials and messages, as a mixture of Gaussians — a set of particles each with a small Gaussian kernel. They use MCMC methods to regenerate the samples multiple times during the course of message passing.

Many of the ideas and techniques involving inference in hybrid systems were first developed in a temporal setting; we therefore also refer the reader to the relevant references in section 15.6.

14.8 Exercises

Exercise 14.1

Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two sets of continuous variables, with $|\boldsymbol{X}| = n$ and $|\boldsymbol{Y}| = m$. Let

$$p(\boldsymbol{Y} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{Y} \mid \boldsymbol{a} + B\boldsymbol{X}; C)$$

where $\boldsymbol{a}$ is a vector of dimension m, B is an $m \times n$ matrix, and C is an $m \times m$ matrix. This dependence is a multidimensional generalization of a linear Gaussian CPD. Show how $p(\boldsymbol{Y} \mid \boldsymbol{X})$ can be represented as a canonical form.

Exercise 14.2

Prove proposition 14.3.

Exercise 14.3

Prove that setting evidence in a canonical form can be done as shown in equation (14.6).

Exercise 14.4⋆

Describe a method that efficiently computes the covariance of any pair of variables X, Y in a calibrated Gaussian clique tree.

Exercise 14.5⋆

Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two sets of continuous variables, with $|\boldsymbol{X}| = n$ and $|\boldsymbol{Y}| = m$. Let $p(\boldsymbol{X})$ be an arbitrary density, and let

$$p(\boldsymbol{Y} \mid \boldsymbol{X}) = \mathcal{N}(\boldsymbol{Y} \mid \boldsymbol{a} + B\boldsymbol{X}; C)$$

where $\boldsymbol{a}$ is a vector of dimension m, B is an $m \times n$ matrix, and C is an $m \times m$ matrix. Show that the first two moments of $p(\boldsymbol{X}, \boldsymbol{Y}) = p(\boldsymbol{X})p(\boldsymbol{Y} \mid \boldsymbol{X})$ depend only on the first two moments of $p(\boldsymbol{X})$ and not on the distribution $p(\boldsymbol{X})$ itself.

Exercise 14.6⋆

Prove theorem 14.5.

Exercise 14.7

Prove the equality in equation (14.17).

Exercise 14.8

Prove equation (14.17).

Exercise 14.9⋆

Our derivation in section 14.4.1 assumes that p and $Y = f(\boldsymbol{U})$ have the same scope $\boldsymbol{U}$. Now, assume that $\mathit{Scope}[f] = \boldsymbol{U}$, whereas our distribution p has scope $\boldsymbol{U}, \boldsymbol{Z}$. We can still use the same method if we define $g(\boldsymbol{u}, \boldsymbol{u}') = f(\boldsymbol{u})$, and integrate g. This solution, however, requires that we perform integration in dimension $|\boldsymbol{U} \cup \boldsymbol{Z}|$, which is often much higher than $|\boldsymbol{U}|$. Since the cost of numerical integration grows with the dimension of the integrals, we can gain considerable savings by using only $\boldsymbol{U}$ to compute our approximation.

In this exercise, you will use the interchangeability of the Gaussian and linear Gaussian representations to perform integration in higher dimension.

a. For $Z \in \boldsymbol{Z}$, show how we can write Z as a linear combination of variables in $\boldsymbol{X}$, with Gaussian noise.
b. Use this expression to write $\mathit{Cov}[Z; Y]$ as a function of the covariances $\mathit{Cov}[X_i; Y]$.
c. Put these results together in order to show how we can obtain a Gaussian approximation to $p(Y, \boldsymbol{Z})$.

Exercise 14.10

In some cases, it is possible to decompose a nonlinear dependency $Y = f(\boldsymbol{X})$ into finer-grained dependencies. For example, we may be able to decompose the nonlinear function f as $f(\boldsymbol{X}) = g(g_1(\boldsymbol{X}_1), g_2(\boldsymbol{X}_2))$, where $\boldsymbol{X}_1, \boldsymbol{X}_2 \subset \boldsymbol{X}$ are smaller subsets of variables.

Show how this decomposition can be used in the context of linearizing the function f in several steps rather than in a single step. What are the trade-offs for this approach versus linearizing f directly?

Exercise 14.11⋆⋆

Show how to combine the EP-based algorithms for CLGs and for nonlinear CPDs to address CLGs where discrete variables can have continuous parents. Your algorithm should specify any constraints on the message passing derived from the need to allow for valid M-projection.

Exercise 14.12⋆

Assume we have a CLG network with discrete variables Δ and continuous variables Γ. In this exercise, we consider collapsed methods that sample the discrete variables and perform exact inference over the continuous variables. Let $\boldsymbol{e}_d$ denote the discrete evidence and $\boldsymbol{e}_p$ the continuous evidence.

a. Given a set of weighted particles such as those described earlier, show how we can estimate the expectation of a function $f(X_i)$ for some $X_i \in \Gamma$. For what functions f do you expect this analysis to give you drastically different answers from the "exact" CLG algorithm of section 14.3.4. (Ignore issues of inaccuracies arising from sampling noise or insufficient number of samples.)
b. Show how we can efficiently apply collapsed likelihood weighting, and show precisely how the importance weights are computed.
c. Now, consider a combined algorithm that generates a clique tree over Δ to generate particles $\boldsymbol{x}_p[1]$, . . ., $\boldsymbol{x}_p[M]$ sampled exactly from $P(\Delta \mid \boldsymbol{e}_d)$. Show the computation of importance weights in this case. Explain the computational benefit of this approach over doing clique tree inference over the entire network.

15 *Inference in Temporal Models*

In chapter 6, we presented several frameworks that provide a higher-level representation language. We now consider the issue of performing probabilistic inference relative to these representations. The obvious approach is based on the observation that a template-based model can be viewed as a generator of ground graphical models: Given a skeleton, the template-based model defines a distribution over a ground set of random variables induced by the skeleton. We can then use any of our favorite inference algorithms to answer queries over this ground network. This process is called *knowledge-based model construction*, often abbreviated as *KBMC*. However, applying this simple idea is far from straightforward.

knowledge-based model construction

First, these models can easily produce models that are very large, or even infinite. Several approaches can be used to reduce the size of the network produced by KBMC; most obviously, given a set of ground query variables $\boldsymbol{Y}$ and evidence $\boldsymbol{E} = \boldsymbol{e}$, we can produce only the part of the network that is needed for answering the query $P(\boldsymbol{Y} \mid \boldsymbol{e})$. In other words, our ground network is a dynamically generated object, and we can generate only the parts that we need for our current query. While this approach can certainly give rise to considerable savings in certain cases, in many applications the network generated is still very large. Thus, we have to consider whether our various inference algorithms scale to this setting, and what additional approximations we must introduce in order to achieve reasonable performance.

Second, the ground networks induced by a template-based model can often be densely connected; most obviously, both aggregate dependencies on properties of multiple objects and relational uncertainty can give rise to dense connectivity. Again, dense connectivity causes difficulties for all exact and most approximate inference algorithms, requiring algorithmic treatment.

Finally, these models give rise to new types of queries that are not easily expressible as standard probabilistic queries. For example, we may want to determine the probability that every person in our family tree has at least one close relative (for some appropriate definition of "close") with a particular disease. This query involves both universal and existential quantifiers; while it can be translated into a ground-level conjunction (over people in the family tree) of disjunctions (over their close relatives), this translation is awkward and gives rise to a query over a very large number of variables.

The development of methods that address these issues is very much an open area of research. In the context of general template-based models, the great variability of the models expressed in these languages limits our ability to provide general-purpose solutions; the existing approaches offer only partial solutions whose applicability at the moment is somewhat limited. We therefore do not review these methods in this book; see section 15.6 for some references. In the more

restricted context of temporal models, the networks have a uniform structure, and the set of relevant queries is better established. Thus, more work has been done on this setting. In the remainder of this chapter, we describe some of the exact and approximate methods that have been developed for inference in temporal models.

15.1 Inference Tasks

We now move to the question of inference in a dynamic Bayesian network (DBN). As we discussed, we can view a DBN as a "generator" for Bayesian networks for different time intervals. Thus, one might think that the inference task is solved. Once we generate a specific Bayesian network, we can simply run the inference algorithm of our choice to answer any queries. However, this view is overly simplistic in two different respects.

First, the Bayesian networks generated from a DBN can be arbitrarily large. Second, the type of reasoning we want to perform in a temporal setting is often different from the reasoning applicable in static settings. In particular, many of the reasoning tasks in a temporal domain are executed *online* as the system evolves.

filtering

For example, a common task in temporal settings is *filtering* (also called *tracking*): at any time point t, we compute our most informed beliefs about the current system state, given all of the evidence obtained so far. Formally, let $\boldsymbol{o}^{(t)}$ denote the observation at time t; we want to keep track of $P(\mathcal{X}^{(t)} \mid \boldsymbol{o}^{(1:t)})$ (or of some marginal of this distribution over some subset of variables).

belief state

As a probabilistic query, we can define the *belief state* at time t to be:

$$\sigma^{(t)}(\mathcal{X}^{(t)}) = P(\mathcal{X}^{(t)} \mid \boldsymbol{o}^{(1:t)}).$$

Note that the belief state is exponentially large in the number of unobserved variables in $\mathcal{X}$. We therefore will not, in general, be interested in the belief state in its entirety. Rather, we must find an effective way of encoding and maintaining the belief state, allowing us to query the current probability of various events of interest (for example, marginal distributions over smaller subsets of variables).

The tracking task is the task of maintaining the belief state over time. A related task is the

prediction

prediction task: at time t, given the observations $\boldsymbol{o}^{(1:t)}$, predict the distribution over (some subset of) the variables at time $t' > t$.

smoothing

A third task, often called *smoothing*, involves computing the posterior probability of $\mathcal{X}^{(t)}$ given all of the evidence $\boldsymbol{o}^{(1:T)}$ in some longer trajectory. The term "smoothing" refers to the fact that, in tracking, the evidence accumulates gradually. In cases where new evidence can have significant impact, the belief state can change drastically from one time slice to the next. By incorporating some future evidence, we reduce these temporary fluctuations. This process is particularly important when the lack of the relevant evidence can lead to temporary "misconceptions" in our belief state.

Example 15.1 *In cases of a sensor failure (such as example 6.5), a single anomalous observation may not be enough to cause the system to realize that a failure has occurred. Thus, the first anomalous sensor reading, at time t_1, may cause the system to conclude that the car did, in fact, move in an unexpected direction. It may take several anomalous observations to reach the conclusion that the sensor has failed. By passing these messages backward, we can conclude that the sensor was already broken*

at t_1, allowing us to discount its observation and avoid reaching the incorrect conclusion about the vehicle location. ■

We note that smoothing can be executed with different time horizons of evidence going forward. That is, we may want to use all of the available evidence, or perhaps just the evidence from some window of a few time slices ahead.

A final task is that of finding the most likely trajectory of the system, given the evidence — $\arg\max_{\xi^{(0:T)}} P(\xi^{(0:T)} \mid \boldsymbol{o}^{(1:T)})$. This task is an instance of the MAP problem.

In all of these tasks, we are trying to compute answers to standard probabilistic queries. Thus, we can simply use one of the standard inference algorithms that we described earlier in this book. However, this type of approach, applied naively, would require us to run inference on larger and larger networks over time and to maintain our entire history of observations indefinitely. Both of these requirements can be prohibitive in practice. Thus, alternative solutions are necessary to avoid this potentially unbounded blowup in the network size.

In the remainder of our discussion of inference, we focus mainly on the tracking task, which presents us with many of the challenges that arise in other tasks. The solutions that we present for tracking can generally be extended in a fairly straightforward way to other inference tasks.

15.2 Exact Inference

We now consider the problem of exact inference in DBNs. We begin by focusing on the filtering problem, showing how the Markovian independence assumptions underlying our representation provide a simple recursive rule that does not require maintaining an unboundedly large representation. We then show how this recursive rule corresponds directly to the upward pass of inference in the unrolled network.

15.2.1 Filtering in State-Observation Models

We begin by considering the filtering task for state-observation models. Our goal here is to maintain the belief state $\sigma^{(t)}(\boldsymbol{X}^{(t)}) = P(\boldsymbol{X}^{(t)} \mid \boldsymbol{o}^{(1:t)})$. As we now show, we can provide a simple recursive algorithm for propagating these belief states, computing $\sigma^{(t+1)}$ from $\sigma^{(t)}$.

Initially, $P(\boldsymbol{X}^{(0)})$ is precisely $\sigma^{(0)}$. Now, assume that we have already computed $\sigma^{(t)}(\boldsymbol{X}^{(t)})$. To compute $\sigma^{(t+1)}$ based on $\sigma^{(t)}$ and the evidence $o^{(t+1)}$, we first propagate the state forward:

$$\begin{aligned}
\sigma^{(\cdot t+1)}(\boldsymbol{X}^{(t+1)}) &= P(\boldsymbol{X}^{(t+1)} \mid o^{(1:t)}) \\
&= \sum_{\boldsymbol{X}^{(t)}} P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)}, o^{(1:t)}) P(\boldsymbol{X}^{(t)} \mid o^{(1:t)}) \\
&= \sum_{\boldsymbol{X}^{(t)}} P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)}) \sigma^{(t)}(\boldsymbol{X}^{(t)}). \qquad (15.1)
\end{aligned}$$

prior belief state

In words, this expression is the beliefs over the state variables at time $t+1$, given the observations only up to time t (as indicated by the $\cdot$ in the superscript). We can call this expression the *prior belief state* at time $t+1$. In the next step, we condition this prior belief state to account for the

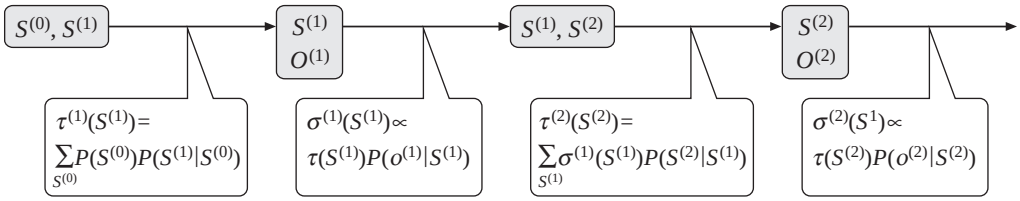


Figure 15.1 Clique tree for HMM

most recent observation $o^{(t+1)}$:

$$\begin{aligned}\sigma^{(t+1)}(\boldsymbol{X}^{(t+1)}) &= P(\boldsymbol{X}^{(t+1)} \mid o^{(1:t)}, o^{(t+1)}) \\ &= \frac{P(o^{(t+1)} \mid \boldsymbol{X}^{(t+1)}, o^{(1:t)})P(\boldsymbol{X}^{(t+1)} \mid o^{(1:t)})}{P(o^{(t+1)} \mid o^{(1:t)})} \\ &= \frac{P(o^{(t+1)} \mid \boldsymbol{X}^{(t+1)})\sigma^{(\cdot t+1)}(\boldsymbol{X}^{(t+1)})}{P(o^{(t+1)} \mid o^{(1:t)})}. \end{aligned} \tag{15.2}$$

recursive filter

This simple *recursive filtering* procedure maintains the belief state over time, without keeping track of a network or a sequence of observations of growing length. To analyze the cost of this operation, let N be the number of states at each time point and T the total number of time slices. The belief-state forward-propagation step considers every pair of states s, s', and therefore it has a cost of $O(N^2)$. The conditioning step considers every state s' (multiplying it by the evidence likelihood and then renormalizing), and therefore it takes $O(N)$ time. Thus, the overall time cost of the message-passing algorithm is $O(N^2T)$. The space cost of the algorithm is $O(N^2)$, which is needed to maintain the state transition model.

15.2.2 Filtering as Clique Tree Propagation

The simple recursive filtering process is closely related to message passing in a clique tree. To understand this relationship, we focus on the simplest state-observation model — the HMM. Consider the process of exact clique tree inference, applied to the DBN for an HMM (as shown in figure 6.2). One possible clique tree for this network is shown in figure 15.1. Let us examine the messages passed in this tree in a sum-product clique tree algorithm, taking the last clique in the chain to be the root. In this context, the upward pass is also called the *forward pass*.

forward pass

The first message has the scope $S^{(1)}$ and represents $\sum_{S^{(0)}} P(S^{(0)})P(S^{(1)} \mid S^{(0)}) = P(S^{(1)})$. The next message, sent from the $S^{(1)}, O^{(1)}$ clique, also has the scope $S^{(1)}$, and represents $P(S^{(1)})P(o^{(1)} \mid S^{(1)}) = P(S^{(1)}, o^{(1)})$. Note that, if we renormalize this message to sum to 1, we obtain $P(S^{(1)} \mid o^{(1)})$, which is precisely $\sigma^{(1)}(S^{(1)})$. Continuing, one can verify that the message from the $S^{(1)}, S^{(2)}$ clique is $P(S^{(2)}, o^{(1)})$, and the one from the $S^{(2)}, O^{(2)}$ clique is $P(S^{(2)}, o^{(1)}, o^{(2)})$. Once again, if we renormalize this last message to sum to 1, we obtain exactly $P(S^{(2)} \mid o^{(1)}, o^{(2)}) = \sigma^{(2)}(S^{(2)})$.

We therefore see that the forward pass of the standard clique-tree message passing algorithm provides us with a solution to the filtering problem. A slight variant of the algorithm gives us

precisely the recursive update equations of equation (15.1) and (15.2). Specifically, assume that we normalize the messages from the $S^{(1)}, O^{(1)}$ clique as they are sent, resulting in a probability distribution. (As we saw in exercise 9.3, such a normalization step has no effect on the belief state computed in later stages, and it is beneficial in reducing underflow.)[1]

It is not hard to see that, with this slight modification, the sum-product message passing algorithm results in precisely the recursive update equations shown here: The message passing step executed by the $S^{(t)}, S^{(t+1)}$ clique is precisely equation (15.1), whereas the message passing step executed by the $S^{(t+1)}, O^{(t+1)}$ clique is precisely equation (15.2), with the division corresponding to the renormalization step.

Thus, the upward (forward) pass of the clique tree algorithm provides a solution to the filtering task, with no need for a downward pass. Similarly, for the prediction task, we can use essentially the same message-passing algorithm, but without conditioning on the (unavailable future) evidence. In terms of the clique tree formulation, the unobserved evidence nodes are barren nodes, and they can thus be dropped from the network; thus, the $S^{(t)}, O^{(t)}$ cliques would simply disappear. When viewed in terms of the iterative algorithm, the operation of equation (15.2) would be eliminated.

For the smoothing task, however, we also need to propagate messages backward in time. Once again, this task is clearly an inference task in the unrolled DBN, which can be accomplished using a clique tree algorithm. In this case, messages are passed in both directions in the clique tree. The resulting algorithm is known as the *forward-backward algorithm*. In this algorithm, the backward messages also have semantics. Assume that our clique tree is over the time slices $0, \ldots, T$. If we use the variable-elimination message passing scheme (without renormalization), the backward message sent to the clique $S^{(t)}, S^{(t+1)}$ represents $P(o^{((t+1):T)} \mid S^{(t+1)})$. If we use the belief propagation scheme, the backward message sent to this clique represents the fully informed (smoothed) distribution $P(S^{(t+1)} \mid o^{(1:T)})$. (See exercise 15.1.)

forward-backward algorithm

For the smoothing task, we need to keep enough information to reconstruct a full belief state at each time t. Naively, we might maintain the entire clique tree at space cost $O(N^2T)$. However, by more carefully analyzing the role of cliques and messages, we can reduce this cost considerably. Consider variable-elimination message passing; in this case, the cliques contain only the initial clique potentials, all of which can be read from the 2-TBN template. Thus, we can cache only the messages and the evidence, at a total space cost of $O(NT)$. Unfortunately, space requirements that grow linearly in the length of the sequence can be computationally prohibitive when we are tracking the system for extended periods. (Of course, some linear growth is unavoidable if we want to remember the observation sequence; however, the size of the state space N is usually much larger than the space required to store the observations.) Exercise 15.2 discusses one approach to reducing this computational burden using a time-space trade-off.

15.2.3 Clique Tree Inference in DBNs

The clique tree perspective provides us with a general algorithm for tracking in DBNs. To derive the algorithm, let us consider the clique tree algorithm for HMMs more closely.

1. Indeed, in this case, the probability $P(S^{(t)}, o^{(1:t)})$ generally decays exponentially at t grows. Thus, without a renormalization step, numerical underflow would be inevitable.

Although we can view the filtering algorithm as performing inference on an unboundedly large clique tree, we never need to maintain more than a clique tree over two consecutive time slices. Specifically, we can create a (tiny) clique tree over the variables $S^{(t)}, S^{(t+1)}, O^{(t+1)}$; we then pass in the message $\sigma^{(t)}$ to the $S^{(t)}, S^{(t+1)}$ clique, pass the message to the $S^{(t+1)}, O^{(t+1)}$ clique, and extract the outgoing message $\sigma^{(t+1)}$. We can now forget this time slice's clique tree and move on to the next time slice's.

template clique tree

It is now apparent that the clique trees for all of the time slices are identical — only the messages passed into them differ. Thus, we can perform this propagation using a *template clique tree* Υ over the variables in the 2-TBN. In this setting, Υ would contain the two cliques $\{S, S'\}$ and $\{S', O'\}$, initialized with the potentials $P(S' \mid S)$ and $P(O' \mid S')$ respectively. To propagate the belief state from time t to time $t+1$, we pass the time t belief state into Υ, by multiplying it into the clique $P(S' \mid S)$, taking S to represent $S^{(t)}$ and S' to represent $S^{(t+1)}$. We then run inference over this clique tree, including conditioning on $O' = o^{(t+1)}$. We can now extract the posterior distribution over S', which is precisely the required belief state $P(S^{(t+1)} \mid o^{(1:(t+1))})$. This belief state can be used as the input message for the next step of propagation.

The generalization to arbitrary DBNs is now fairly straightforward. We maintain a belief state $\sigma^{(t)}(\mathcal{X}^{(t)})$ and propagate it forward from time t to time $t+1$. We perform this propagation step using clique tree inference as follows: We construct a template clique tree Υ, defined over the variables of the 2-TBN. We then pass a time t message into Υ, and we obtain as the result of clique tree inference in Υ a time $t+1$ message that can be used as the input message for the next step. Most naively, the messages are full belief states, specifying the distribution over all of the unobserved variables.

In general, however, we can often reduce the scope of the message passed: As can be seen from equation (6.2), only the time t interface variables are relevant to the time $t+1$ distribution. For example, consider the two generalized HMM structures of figure 6.3. In the factorial HMM structure, all variables but the single observation variable are in the interface, providing little savings. However, in the coupled HMM, all of the private observation variables are not in the interface, leaving a much smaller belief state whose scope is X_1, X_2, X_3. Observation variables are not the only ones that can be omitted from the interface; for example, in the network of figure 15.3a, the nonpersistent variable B is also not in the interface.

reduced belief state

The algorithm is shown in algorithm 15.1. It passes messages corresponding to *reduced belief states* $\sigma^{(t)}(\mathcal{X}_I^{(t)})$. At phase t, it passes a time $t-1$ reduced belief state into the template clique tree, calibrates it, and extracts the time t reduced belief state to be used as input for the next step. Note that the clique-tree calibration step is useful not only for the propagation step. It also provides us with other useful conclusions, such as the marginal beliefs over all individual variables $X^{(t)}$ (and some subsets of variables) given the observations $\boldsymbol{o}^{(1:t)}$.

15.2.4 Entanglement

The use of a clique tree immediately suggests that we are exploiting structure in the algorithm, and so can expect the inference process to be tractable, at least in a wide range of situations. Unfortunately, that does not turn out to be the case. The problem arises from the need to represent and manipulate the (reduced) belief state $\sigma^{(t)}$ (a process not specified in the algorithm). Semantically, this belief state is a joint distribution over $\mathcal{X}_I^{(t)}$; if represented naively,

Algorithm 15.1 Filtering in a DBN using a template clique tree

Procedure CTree-Filter-DBN (
  $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$, // DBN
  $\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)}, \ldots$ // Observation sequence
)
1 Construct template clique tree Υ over $\mathcal{X}_I \cup \mathcal{X}'$
2 $\sigma^{(0)} \leftarrow P_{\mathcal{B}_0}(\mathcal{X}_I^{(0)})$
3 **for** $t = 1, 2, \ldots$
4 $\mathcal{T}^{(t)} \leftarrow \Upsilon$
5 Multiply $\sigma^{(t-1)}(\mathcal{X}_I^{(t-1)})$ into $\mathcal{T}^{(t)}$
6 Instantiate $\mathcal{T}^{(t)}$ with $\boldsymbol{o}^{(t)}$
7 Calibrate $\mathcal{T}^{(t)}$ using clique tree inference
8 Extract $\sigma^{(t)}(\mathcal{X}_I^{(t)})$ by marginalization

it would require an exponential number of entries in the joint. At first glance, this argument appears specious. After all, one of the key benefits of graphical models is that high-dimensional distributions can be represented compactly by using factorization. It certainly appears plausible that we should be able to find a compact representation for our belief state and use our structured inference algorithms to manipulate it efficiently. As we now show, this very plausible impression turns out to be false.

Example 15.2 *Consider our car network of figure 6.1, and consider our belief state at some time t. Intuitively, it seems as if there should be some conditional independence relations that hold in this network. For example, it seems as if Weather$^{(2)}$ and Location$^{(2)}$ should be uncorrelated. Unfortunately, they are not: if we examine the unrolled DBN, we see that there is an active trail between them going through Velocity$^{(1)}$ and Weather$^{(0)}$, Weather$^{(1)}$. This path is not blocked by any of the time 2 variables; in particular, Weather$^{(2)}$ and Location$^{(2)}$ are not conditionally independent given Velocity$^{(2)}$. In general, a similar analysis can be used to show that, for $t \geq 2$, no conditional independence assumptions hold in $\sigma^{(t)}$.* ■

entanglement

This phenomenon, known as *entanglement*, has significant implications. As we discussed, there is a direct relationship between conditional independence properties of a distribution and our ability to represent it as a product of factors. Thus, a distribution that has no independence properties does not admit a compact representation in a factored form.

Unfortunately, the entanglement phenomenon is not specific to this example. Indeed, it holds for a very broad class of DBNs. We demonstrate it for a large subclass of DBNs that exhibit a very regular structure. We begin by introducing a few useful concepts.

Definition 15.1 persistent independence

For a DBN over $\mathcal{X}$, and $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \subset \mathcal{X}$, we say that the independence $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z})$ is persistent *if $(\boldsymbol{X}^{(t)} \perp \boldsymbol{Y}^{(t)} \mid \boldsymbol{Z}^{(t)})$ holds for every t.* ■

Persistent independencies are independence properties of the belief state, and are therefore precisely what we need in order to provide a time-invariant factorization of the belief state.

The following concept turns out to be a useful one, in this setting and others.

Definition 15.2 influence graph

Let $\mathcal{B}_{\rightarrow}$ be a 2-TBN over $\mathcal{X}$. We define the influence graph *for $\mathcal{B}_{\rightarrow}$ to be a directed cyclic graph $\mathcal{I}$ over $\mathcal{X}$ whose nodes correspond to $\mathcal{X}$, and that contains a directed arc $X \rightarrow Y$ if $X \rightarrow Y'$ or $X' \rightarrow Y'$ appear in $\mathcal{B}_{\rightarrow}$. Note that a persistence arc $X \rightarrow X'$ induces a self-cycle in the influence graph.* ■

The influence graph corresponds to influence in the unrolled DBN:

Proposition 15.1

Let $\mathcal{I}$ be the influence graph for a 2-TBN $\mathcal{B}_{\rightarrow}$. Then $\mathcal{I}$ contains a directed path from X to Y if and only if, in the unrolled DBN, for every t, there exists a path from $X^{(t)}$ to $Y^{(t')}$ for some $t' \geq t$.

See exercise 15.3.

The following result demonstrates the inevitability of the entanglement phenomenon, by proving that it holds in a broad class of networks. A DBN is called *fully persistent* if it encodes a state-observation model, and, for each state variable $X \in \boldsymbol{X}$, the 2-TBN contains a *persistence edge* $X \rightarrow X'$.

fully persistent

persistence edge

Theorem 15.1

Let $\langle \mathcal{G}_0, \mathcal{G}_{\rightarrow} \rangle$ be a fully persistent DBN structure over $\mathcal{X} = \boldsymbol{X} \cup \boldsymbol{O}$, where the state variables $\boldsymbol{X}^{(t)}$ are hidden in every time slice, and the observation variables $\boldsymbol{O}^{(t)}$ are observed in every time slice. Furthermore, assume that, in the influence graph for $\mathcal{G}_{\rightarrow}$:

- *there is a trail (not necessarily a directed path) between every pair of nodes, that is, the graph is connected;*
- *every state variable X has some directed path to some evidence variable in $\boldsymbol{O}$.*

Then there is no persistent independence $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z})$ that holds for every DBN $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$ over this DBN structure.

The proof is left as an exercise (see exercise 15.4). Note that, as in every other setting, there may be spurious independencies that hold due to specific choices of the parameters. But, for almost all choices of the parameters, there will be no independence that holds persistently.

In fully persistent DBNs, the tracking problem is precisely one of maintaining a belief state — a distribution over $\boldsymbol{X}^{(t)}$. The entanglement theorem shows that the only exact representation for this belief state is as a full joint distribution, rendering any belief-state-algorithm computationally infeasible except in very small networks.

More generally, if we want to track the system as it evolves, we need to maintain a representation that summarizes all of our information about the past. Specifically, as we showed in theorem 10.2, any sepset in a clique tree must render the two parts of the tree conditionally independent. In a temporal setting, we cannot allow the sepsets to grow unboundedly with the number of time slices. Therefore, there must exist some sepset over a scope $\mathcal{Y}$ that cuts across the network, in that any path that starts from a time 0 variable and continues to infinity must intersect $\mathcal{Y}$. In fully persistent networks, the set of state variables $\boldsymbol{X}^{(t)}$ is a minimal set satisfying this condition. The entanglement theorem states that this set exhibits no persistent independencies, and therefore the message over this sepset can only be represented as an explicit joint distribution. The resulting sepsets are therefore very large — exponential in the number of state variables. Moreover, as these large messages must be incorporated into some clique in the clique tree, the cliques also become exponentially large.

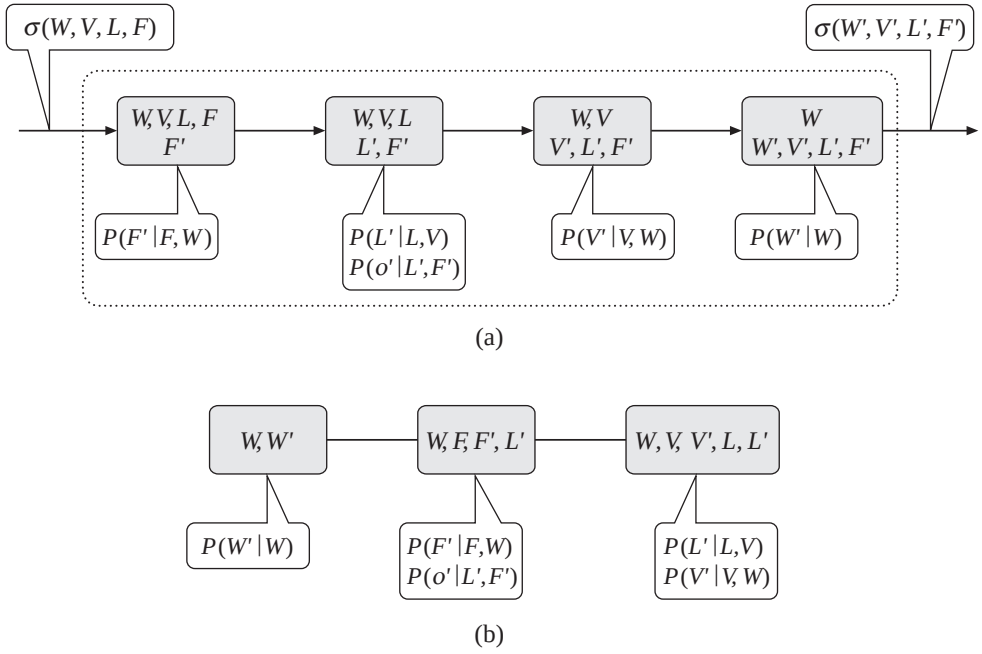


Figure 15.2 Different clique trees for the Car **DBN of figure 6.1.** (a) A template clique tree that allows exact filtering for the Car DBN of figure 6.1. Because the variable O' is always observed, it is not in the scope of any clique (although the factor $P(o' \mid F', L')$ is associated with a clique). (b) A clique tree for the 2-TBN of the Car network, which does not allow exact filtering.

Example 15.3

Consider again the Car *network of figure 6.1. To support exact belief state propagation, our template clique tree must include a clique containing W, V, L, F, where we can incorporate the previous belief state $\sigma^{(t)}(W^{(t)}, V^{(t)}, L^{(t)}, F^{(t)})$. It must also contain a clique containing W', V', L', F', from which we can extract $\sigma^{(t+1)}(W^{(t+1)}, V^{(t+1)}, L^{(t+1)}, F^{(t+1)})$. A minimally sized clique tree containing these cliques is shown in figure 15.2a. All of the cliques in the tree are of size 5. By contrast, if we were simply to construct a template clique tree over the dependency structure defined by the 2-TBN, we could obtain a clique tree where the maximal clique size is 4, as illustrated in figure 15.2b.*

A clique size of 4 is the minimum we can hope for: In general, all of the cliques for a fully persistent network over n variables contain at least $n+1$ *variables: one representative (either X or X') of each variable X in $\mathcal{X}$, plus an additional variable that we are currently eliminating. (See exercise 15.5.) In many cases, the minimal induced width would actually be higher. For example, if we introduce an arc $L \rightarrow V'$ into our 2-TBN (for example, because different locations have different speed limits), the smallest template clique tree allowing for exact filtering has a clique size of 6.* ■

Even in networks when not all variables are persistent, entanglement is still an issue: We still need to represent a distribution that cuts across the entire width of the network. In most cases

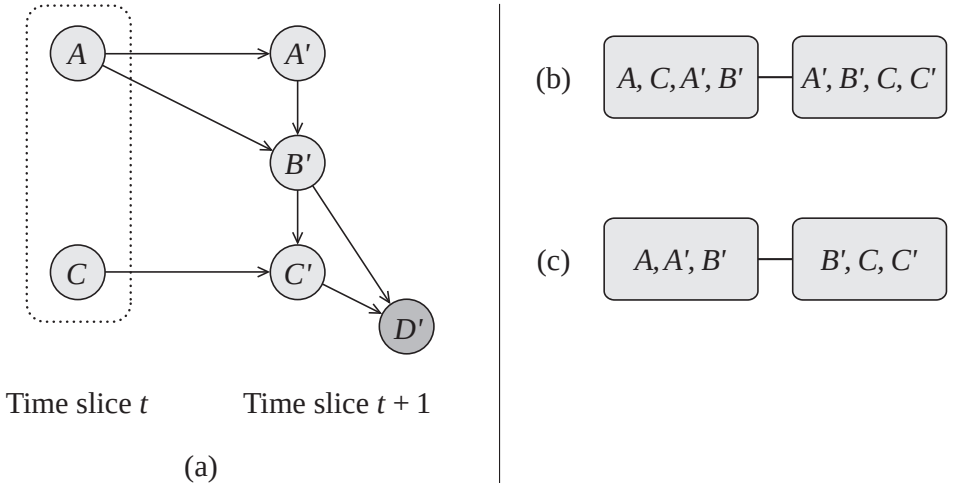


Figure 15.3 Nonpersistent 2-TBN and different possible clique trees: (a) A 2-TBN where not all of the unobserved variables are persistent. (b) A clique tree for this 2-TBN that allows exact filtering; as before, D' is always observed, and hence it is not in the scope of any clique. (c) A clique tree for the 2-TBN, which does not allow exact filtering.

— except for specific structures and observation patterns — these distributions do not exhibit independence structure, and they must therefore be represented as an explicit joint distribution over the interface variables. Because the "width" of the network is often fairly large, this results in large messages and large cliques.

Example 15.4

Consider the 2-TBN shown in figure 15.3, where not all of the unobserved variables are persistent. In this network, our interface variables are A, C. Thus, we can construct a template clique tree over A, C, A', B', C', D' as the basis for our message passing step. To allow exact filtering, we must have a clique whose scope contains A, C and a clique whose scope contains A', C'. A minimally sized clique tree satisfying these constraints is shown in figure 15.3b. It has a maximum clique size of 4; without these constraints, we can construct a clique tree where the maximal clique size is 3 (figure 15.3c). ■

We note that it is sometimes possible to find better clique trees than those constrained to use $\mathcal{X}_I^{(t)}$ as the message. In fact, in some cases, the best sepset actually spans variables in multiple time slices. However, these improvements do not address the fundamental problem, which is that the computational cost of exact inference in DBNs grows exponentially with the "width" of the network. Specifically, we cannot avoid including in our messages at least the set of persistent variables. In many applications, a large fraction of the variables are persistent, rendering this approach intractable.

15.3 Approximate Inference

The computational problems with exact inference force us, in many cases, to fall back on approximate inference. In principle, when viewing the DBN as a large unrolled BN, we can apply any of the approximate inference algorithms that we discussed for BNs. Indeed, there has been significant success, for example, in using a variational approach for reasoning about weakly coupled processes that evolve in parallel. (See exercise 15.6.)

However, several complications arise in this setting. First, as we discussed in section 15.1, the types of tasks that we wish to address in the temporal setting often involve reasoning about arbitrarily large, and possibly unbounded, trajectories. Although we can always address these tasks using inference over the unrolled DBNs, as in the case of exact inference, algorithms that require us to maintain the entire unrolled BN during the inference process may be impractical. In the approximate case, we must also address an additional complication: An approximate inference algorithm that achieves reasonable errors for static networks of bounded size may not work well in arbitrarily large networks. Indeed, the quality of the approximation may degrade with the size of the network.

For the remainder of this section, we focus on the filtering task. As in the case of exact inference, methods developed for filtering extend directly to prediction, and (with a little work) to smoothing with a bounded lookahead. There are many algorithms that have been proposed for approximate tracking in dynamic systems, and one could come up with several others based on the methods described earlier in this book. We begin by providing a high-level overview of a general framework that encompasses these algorithms. We then describe two specific methods that are commonly used in practice, one that uses a message passing approach and the other a sampling approach.

15.3.1 Key Ideas

15.3.1.1 Bounded History Updates

A general approach to addressing the filtering (or prediction) task without maintaining a full history is related to the approach we used for exact filtering. There, we passed messages in the clique tree forward in time, which allowed us to throw away the observations and the cliques in previous time slices once they have been processed.

In principle, the same idea can be applied in the case of approximate inference: We can execute the appropriate inference steps for a time slice and then move forward to the next time slice. However, most approximate inference algorithms require that the same variable in the network be visited multiple times during the course of inference. For example, belief propagation algorithms (as in section 11.3 or section 11.4) send multiple messages through the same cluster. Similarly, structured variational approximation methods (as in section 11.5) are also iterative, running inference on the network multiple times, with different values for the variational parameters. Markov chain Monte Carlo algorithms also require that each node be visited and sampled multiple times. Thus, we cannot just throw away the history and apply our approximate inference to the current time slice alone.

One common solution is to use a form of "limited history" in the inference steps. The various steps associated with the inference algorithm are executed not over the whole network, but over a subnetwork covering only the recent history. Most simply, the subnetwork for time t

is simply a bounded window covering some predetermined number k of previous time slices $t-k,\ldots,t-1,t$. More generally, the subnetwork can be determined in a dynamic fashion, using a variety of techniques.

We will describe the two methods most commonly used in practice, one based on importance sampling, and the other on approximate message propagation. Both take $k=1$, using only the current approximate belief state $\hat{\sigma}^{(t)}$ and the current time slice in estimating the next approximate belief state $\hat{\sigma}^{(t+1)}$. In effect, these methods perform a type of approximate message propagation, as in equation (15.1) and equation (15.2). Although this type of approximation is clearly weak in some cases, it turns out to work fairly well in practice.

15.3.1.2 Analysis of Convergence

The idea of running approximate inference with some bounded history over networks of increasing size immediately raises concerns about the quality of the approximation obtained. Consider, for example, the simplest approximate belief-state filtering process. Here, we maintain an approximate belief state $\hat{\sigma}^{(t)}$, which is (hopefully) similar to our true belief state $\sigma^{(t)}$. We use $\hat{\sigma}^{(t)}$ to compute the subsequent belief state $\hat{\sigma}^{(t+1)}$. This step uses approximate inference and therefore introduces some additional error into our approximation. Therefore, as time evolves, our approximation appears to be accumulating more and more errors. In principle, it might be the case that, at some point, our approximate belief state $\hat{\sigma}^{(t)}$ bears no resemblance to the true belief state $\sigma^{(t)}$.

☞ **Although unbounded errors can occur, it turns out that such situations are rare in practice (for algorithms that are carefully designed). The main reason is that the dynamic system itself is typically stochastic. Thus, the effect of approximations that occur far in the past tends to diminish over time, and the overall error (for well-designed algorithms) tends to remain bounded indefinitely.** For several algorithms (including the two described in more detail later), one can prove a formal result along these lines. All of these results make some assumptions about the stochasticity of the system — the rate at which it "forgets" the past. Somewhat more formally, assume that propagating two distributions through the system dynamics (equation (15.1) and 15.2) reduces some notion of distance between them. In this case, discrepancies between $\hat{\sigma}^{(t)}$ and $\sigma^{(t)}$, which result from approximations in previous time slices, decay over time. Of course, new errors are introduced by subsequent approximations, but, in stochastic systems, we can show that they do not accumulate unboundedly.

Formal theorems proving a uniform bound on the distance between the approximate and true belief state — a bound that holds for all time points t — exist for a few algorithms. These theorems are quite technical, and the actual bounds obtained on the error are fairly large. For this reason, we do not present them here. However, in practice, when the underlying system is stochastic, we do see a bounded error for approximate propagation algorithms. Conversely, when the system evolution includes a deterministic component — for example, when the state contains a variable that (once chosen) does not evolve over time — the errors of the approximate inference algorithms often do diverge over time. Thus, while the specific bounds obtain in the theoretical analyses may not be directly useful, they do provide a theoretical explanation for the behavior of the approximate inference algorithms in practice.

15.3.2 Factored Belief State Methods

The issue underlying the entanglement result is that, over time, all variables in a belief state slice eventually become correlated via active trails through past time slices. In many cases, however, these trails can be fairly long, and, as a consequence, the resulting correlations can be quite weak. This raises the idea of replacing the exact, fully correlated, belief state, with an approximate, factorized belief state that imposes some independence assumptions. For a carefully chosen factorization structure, these independence assumptions may be a reasonable approximation to the structure in the belief state.

This idea gives rise to the following general structure for a filtering algorithm: At each time point t, we have a factored representation $\hat{\sigma}^{(t)}$ of our time t belief state. We then compute the correct update of this time t belief state to produce a new time $t+1$ belief state $\sigma^{(\cdot t+1)}$. The update step consists of propagating the belief state forward through the system dynamics and conditioning on the time $t+1$ observations. Owing to the correlations induced by the system dynamics (as in section 15.2.4), $\sigma^{(\cdot t+1)}$ has more correlations than $\hat{\sigma}^{(t)}$, and therefore it requires larger factors to represent correctly. If we continue this process, we rapidly end up with a belief state that has no independence structure and must be represented as a full joint distribution. Therefore, we introduce a *projection* step, where we approximation $\sigma^{(\cdot t+1)}$ using a more factored representation, giving rise to a new $\hat{\sigma}^{(t+1)}$, with which we continue the process. This update-project cycle ensures that our approximate belief state remains in a class of distributions that we can tractably maintain and update.

belief-state projection

Most simply, we can represent the approximate belief state $\hat{\sigma}^{(t)}$ in terms of a set of factors $\Phi^{(t)} = \{\beta_r^{(t)}(\boldsymbol{X}_r^{(t)})\}$, where we assume (for simplicity) that the factorization of the messages (that is, the choice of scopes $\boldsymbol{X}_r$) does not change over time. Most simply, the scopes of the different factors are disjoint, in which case the belief state is simply a product of marginals over disjoint variables or subsets of variables. As a richer but more complex representation, we can represent $\hat{\sigma}^{(t)}$ using a calibrated cluster tree, or even a calibrated cluster graph $\mathcal{U}$. Indeed, we can even use a general representation that uses overlapping regions and associated counting numbers:

$$\hat{\sigma}^{(t)}(\mathcal{X}^{(t)}) = \prod_r (\beta_r^{(t)}(\boldsymbol{X}_r^{(t)}))^{\kappa_r}.$$

Example 15.5

Consider the task of monitoring a freeway with k cars. As we discussed, after a certain amount of time, the states of the different cars become entangled, so our only option for representing the belief state is as a joint distribution over the states of all *the cars. An obvious approximation is to assume that the correlations between the different cars are not very strong. Thus, although the cars do influence each other, the current state of one car does not tell us too much about the current state of another. Thus, we can choose to approximate the belief state over the entire system using an approximate belief state that ignores or approximates these weak correlations. Specifically, let $\boldsymbol{Y}_i$ be the set of variables representing the state of car i, and let $\boldsymbol{Z}$ be a set of variables that encode global conditions, such as the weather or the current traffic density. Most simply, we can represent the belief state $\hat{\sigma}^{(t)}$ as a product of marginals*

$$\beta_g^{(t)}(\boldsymbol{Z}^{(t)}) \prod_{i=1}^{k} \beta_i^{(t)}(\boldsymbol{Y}_i^{(t)}).$$

In a better approximation, we might preserve the correlations between the state of each individual vehicle and the global system state, by selecting as our factorization

$$\left(\beta_g^{(t)}(\boldsymbol{Z}^{(t)})\right)^{-(k-1)} \prod_{i=1}^{k} \beta_i^{(t)}(\boldsymbol{Z}^{(t)}, \boldsymbol{Y}_i^{(t)}),$$

where the initial term compensates for the multiple counting of the probability of $\boldsymbol{Z}^{(t)}$ in the other factors. Here, the representation of the approximate belief state makes the assumption that the states of the different cars are conditionally independent given the global state variables. ■

We showed that exact filtering is equivalent to a forward pass of message passing in a clique tree, with the belief states playing the role of messages. Hence, filtering with factored belief states is simply a form of message passing with approximate messages. The use of an approximate belief state in a particular parametric class is also known as *assumed density filtering*. This algorithm is a special case of the more general algorithm that we developed in the context of the *expectation propagation* (EP) algorithm of section 11.4. Viewed abstractly, each *slice-cluster* in our monolithic DBN clique tree (one that captures the entire trajectory) corresponds to a pair of adjacent time slices $t-1, t$ and contains the variables $\mathcal{X}_I^{(t)} \cup \mathcal{X}^{(t+1)}$. As in the general EP algorithm, the potential in a slice-cluster is never represented explicitly, but in a decomposed form, as a product over factors. Each slice-cluster is connected to its predecessor and successor slice-clusters. The messages between these slice-clusters correspond to approximate belief states $\hat{\sigma}^{(t)}(\mathcal{X}_I^{(t)})$, which are represented in a factorized form. For uniformity of exposition, we assume that the initial state distribution — the time 0 belief state — also takes (or is approximated as) the same form. Thus, when propagating messages in this chain, each slice-cluster takes messages in this factorized form and produces messages in this form.

assumed density filter

expectation propagation

As we discussed in section 11.4.2, the use of factorized messages allows us to perform the operations in each cluster much more efficiently, by using a nested clique tree or cluster graph that exploits the joint structure of the messages and cluster potential. For example, if the belief state is fully factored as a product over the variables in the interface, the message structure imposes no constraints on the nested data structure used for inference within a time slice. In particular, we can use any clique tree over the 2-TBN structure; for instance, in example 15.3, we can use the structure of figure 15.2b. By contrast, for exact filtering, the messages are full belief states over the interface variables, requiring the use of a nested clique tree with very large cliques. Of course, a fully factorized belief state generally provides a fairly poor approximation to the belief state. As we discussed in the context of the EP algorithm, we can also use much more refined approximations, which use a clique tree or even a general region-based approximation to the belief state.

The algorithm used for the message passing is precisely as we described in section 11.4.2, and we do not repeat it here. We make only three important observations. First, unlike a traditional application of EP, when doing filtering, we generally do only a single upward pass of message propagation, starting at time 0 and propagating toward higher time slices. Because we do not have a backward pass, the distinctions between the sum-product algorithm (section 11.4.3.1) and the belief update algorithm (section 11.4.3.2) are irrelevant in this setting, since the difference arises only in the backward pass. Second, without a backward pass, we do not need to keep track of a clique once it has propagated its message forward. Thus, as in exact inference for

DBNs, we can keep only a single (factored) message and single (factored) slice-cluster in memory at each point in time and perform the message propagation in space that is constant in the number of time slices.

If we continue to assume that the belief state representation is the same for every time slice, then the factorization structure used in each of the message passing steps is identical. In this case, we can perform all the message passing steps using the same *template cluster graph* that has a fixed cluster structure and fixed initial factors (those derived from the 2-TBN); at each time t, the factors representing $\hat{\sigma}^{(t)}$ are introduced into the template cluster graph, which is then calibrated and used to produce the factors representing $\hat{\sigma}^{(t+1)}$. The reuse of the template can reduce the cost of the message propagation step. An alternative approach allows the structure used in our approximation to change over time. This flexibility allows us to adapt our structure to reflect the strengths of the interactions between the variables in our domain. For example, in example 15.5, we might expect the variables associated with cars that are directly adjacent to be highly correlated; but the pairs of cars that are close to each other change over time. Section 15.6 describes some methods for dynamically adapting the representation to the current distribution.

template cluster graph

15.3.3 Particle Filtering

Of the different particle-based methods that we discussed, forward sampling appears best suited to the temporal setting, since it generates samples incrementally, starting from the root of the network. In the temporal setting, this would correspond to generating trajectories starting from the beginning of time, and going forward. This type of sampling, we might hope, is more amenable to a setting where we do not have to keep sampled trajectories that go indefinitely far back. Obviously, rejection sampling is not an appropriate basis for a temporal sampling algorithm. For an indefinitely long trajectory, all samples will eventually be inconsistent with our observations, so we will end up rejecting all samples. In this section, we present a family of filtering algorithms based on importance sampling and analyze their behavior.

15.3.3.1 Naive Likelihood Weighting

It is fairly straightforward to generalize likelihood weighting to the temporal setting. Recall that LW generates samples by sampling nodes that are not observed from their appropriate distribution, and instantiating nodes that are observed to their observed values. Every node that is instantiated in this way causes the weight of the sample to be changed. However, LW generates samples one at a time, starting from the root and continuing until a full assignment is generated. In an online setting, we can have arbitrarily many variables, so there is no natural end to this sampling process. Moreover, in the filtering problem, we want to be able to answer queries online as the system evolves. Therefore, we first adapt our sampling process to return intermediate answers.

The LW algorithm for the temporal setting maintains a set of samples, each of which is a trajectory up to the current time slice t: $\xi^{(t)}[1], \ldots, \xi^{(t)}[M]$. Each sampled trajectory is associated with a weight $w[m]$. At each time slice, the algorithm takes each of the samples, propagates it forward to sample the variables at time t, and adjusts its weight to suit the new evidence at time t. The algorithm uses a likelihood-weighting algorithm as a subroutine to

propagate a time t sample to time $t+1$. The version of the algorithm for 2-TBNs is almost identical to the algorithm 12.2; it is shown in algorithm 15.2 primarily as a reminder.

Algorithm 15.2 Likelihood-weighted particle generation for a 2-TBN

Procedure LW-2TBN (
  $\mathcal{B}_{\rightarrow}$ // 2-TBN
  ξ // Instantiation to time $t-1$ variables
  $\boldsymbol{O}^{(t)} = \boldsymbol{o}^{(t)}$ // time t evidence
)
  Let $X'_1, \ldots, X'_n$ be a topological ordering of $\mathcal{X}'$ in $\mathcal{B}_{\rightarrow}$
  $w \leftarrow 1$
  for $i = 1, \ldots, n$
    $\boldsymbol{u}_i \leftarrow (\xi, \boldsymbol{x}')\langle \mathrm{Pa}_{X'_i} \rangle$
      // Assignment to $\mathrm{Pa}_{X'_i}$ in $x_1, \ldots, x_n, x'_1, \ldots, x'_{i-1}$
    if $X'_i \notin \boldsymbol{O}^{(t)}$ **then**
      Sample x'_i from $P(X'_i \mid \boldsymbol{u}_i)$
    else
      $x'_i \leftarrow \boldsymbol{o}^{(t)}\langle X'_i \rangle$ // Assignment to X'_i in $\boldsymbol{o}^{(t)}$
      $w \leftarrow w \cdot P(x'_i \mid \boldsymbol{u}_i)$ // Multiply weight by probability of desired value
  return $(x'_1, \ldots, x'_n), w$

Algorithm 15.3 Likelihood weighting for filtering in DBNs

Procedure LW-DBN (
  $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$, // DBN
  M // Number of samples
  $\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)}, \ldots$ // Observation sequence
)
  for $m = 1, \ldots, M$
    Sample $\xi^{(0)}[m]$ from $\mathcal{B}_0$
    $w[m] \leftarrow 1$
  for $t = 1, 2, \ldots$
    for $m = 1, \ldots, M$
      $(\xi^{(t)}[m], w) \leftarrow$ LW-2TBN$(\mathcal{B}_{\rightarrow}, \xi^{(t-1)}[m], \boldsymbol{o}^{(t)})$
        // Sample time t variables starting from time $t-1$ sample
      $w[m] \leftarrow w[m] \cdot w$
        // Multiply weight of m'th sample with weight of time t evidence
    $\hat{\sigma}^{(t)}(\xi) \leftarrow \frac{\sum_{m=1}^{M} w[m] \mathbf{I}\{\xi^{(t)}[m] = \xi\}}{\sum_{m=1}^{M} w[m]}$

Unfortunately, this extension of the basic LW algorithm is generally a very poor algorithm for DBNs. To understand why, consider the application of this algorithm to any state-observation

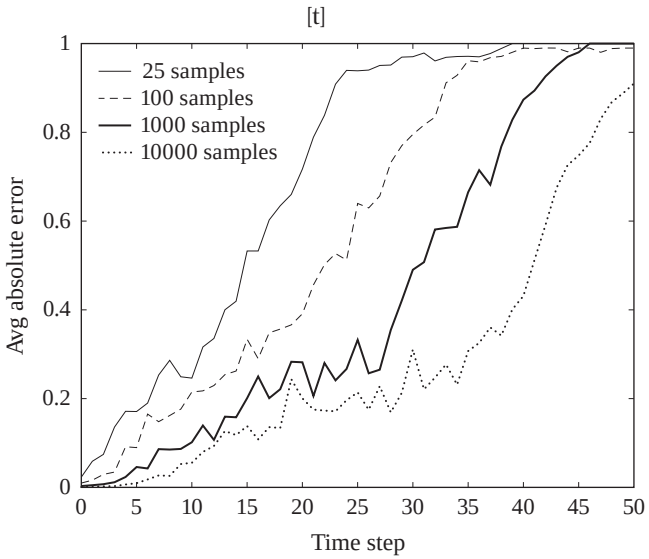


Figure 15.4 Performance of likelihood weighting over time with different numbers of samples, for a state-observation model with one state and one observation variable.

model. In this case, we have a very long network, where all of the evidence is at the leaves. Unfortunately, as we discussed, in such networks, LW generates samples according to the prior distribution, with the evidence affecting only the weights. In other words, the algorithm generates completely random state trajectories, which "match" the evidence only by chance. For example, in our Car example, the algorithm would generate completely random trajectories for the car, and check whether one of them happens to match the observed sensor readings for the car's location. Clearly, the probability that such a match occurs — which is precisely the weight of the sample — decreases exponentially (and quite quickly) with time. This problem can also arise in a static BN, but it is particularly severe in this setting, where the network size grows unboundedly. In this case, as time evolves, more and more evidence is ignored in the sample-generation process (affecting only the weight), so that the samples become less and less relevant. Indeed, we can see in figure 15.4 that, in practice, the samples generated get increasingly irrelevant as t grows, so that LW diverges rapidly as time goes by. From a technical perspective, this occurs because, over time, the variance of the weights of the samples grows very quickly, and unboundedly. Thus, the quality of the estimator obtained from this procedure — the probability that it returns an answer within a certain error tolerance — gets increasingly worse.

particle filter

sequential importance sampling

One approach called *particle filtering* (or *sequential importance sampling*) for addressing this problem is based on the key observation that not all samples are equally "good." In particular, **samples that have higher weight explain the evidence observed so far much better, and are likely to be closer to the current state. Thus, rather than propagate all samples forward to the next time step, we should preferentially select "good" samples for propagation, where "good" samples are ones that have high weight.** There are many ways of implementing this basic intuition: We can select samples for propagation deterministically or stochastically.

We can use a fixed number of samples, or vary the number of samples to achieve a certain quality of approximation (estimated heuristically).

15.3.3.2 The Bootstrap Filter

bootstrap filter

The simplest and most common variant of particle filtering is called the *bootstrap filter*. It maintains a set $\mathcal{D}^{(t)}$ of M time t trajectories $\bar{\boldsymbol{x}}^{(0:t)}[m]$, each associated with its own weight $w^{(t)}[m]$. When propagating samples to the next time slice, each sample is chosen randomly for propagation, proportionately to its current weight. The higher the weight of the sample, the more likely it is to be selected for propagation; thus, higher-weight samples may "spawn" multiple copies, whereas lower-weight ones "die off" to make space for the others.

More formally, consider a data set $\mathcal{D}^{(t)}$ consisting of M weighted sample trajectories $(\bar{\boldsymbol{x}}^{(0:t)}[m], w^{(t)}[m])$. We can define the empirical distribution generated by the data set:

$$\hat{P}_{\mathcal{D}^{(t)}}(\boldsymbol{x}^{(0:t)}) \propto \sum_{m=1}^{M} w^{(t)}[m] \mathbf{I}\{\bar{\boldsymbol{x}}^{(0:t)}[m] = \boldsymbol{x}^{(0:t)}\}.$$

This distribution is a weighted sum of delta distributions, where the probability of each assignment is its total weight in $\mathcal{D}^{(t)}$, renormalized to sum to 1.

The algorithm then generates M new samples for time $t+1$ as follows: For each sample m, it selects a time t sample for propagation by randomly sampling from $\hat{P}_{\mathcal{D}^{(t)}}$. Each of the M selected samples is used to generate a new time $t+1$ sample using the transition model, which is weighted using the observation model. Note that the weight of the sample $w^{(t)}[m]$ manifests in the relative proportion with which the mth sample is propagated. Thus, we do not need to account for its previous weight when determining the weight of the time $t+1$ sample generated from it. If we did include its weight, we would effectively be double-counting it. The algorithm is shown in algorithm 15.4 and illustrated in figure 15.5.

We can view $\hat{P}_{\mathcal{D}^{(t)}}$ as an approximation to the time t belief state (one where only the sampled states have nonzero probability), and the sampling step as using it to generate an approximate belief state for time $t+1$. Thus, this algorithm can be viewed as performing a stochastic version of the belief-state filtering process.

Note that we view the algorithm as maintaining entire trajectories $\bar{\boldsymbol{x}}^{(0:t)}$, rather than simply the current state. In fact, each sample generated does correspond to an entire trajectory. However, for the purpose of filtering, the earlier parts of the trajectory are not relevant, and we can throw out all but the current state $\bar{\boldsymbol{x}}^{(t)}$.

The bootstrap particle filter works much better than likelihood weighting, as illustrated in figure 15.6a. Indeed, the error seems to remain bounded indefinitely over time (b).

We can generalize the basic bootstrap filter along two dimensions. The first modifies the forward sampling procedure — the process by which we extend a partial trajectory $\bar{\boldsymbol{x}}^{(0:t-1)}$ to include a time t state variable assignment $\bar{\boldsymbol{x}}^{(t)}$. The second modifies the particle selection scheme, by which we take a set of weighted time t samples $\mathcal{D}^{(t)}$ and use their weights to select a new set of time t samples. We will describe these two extensions in more detail.

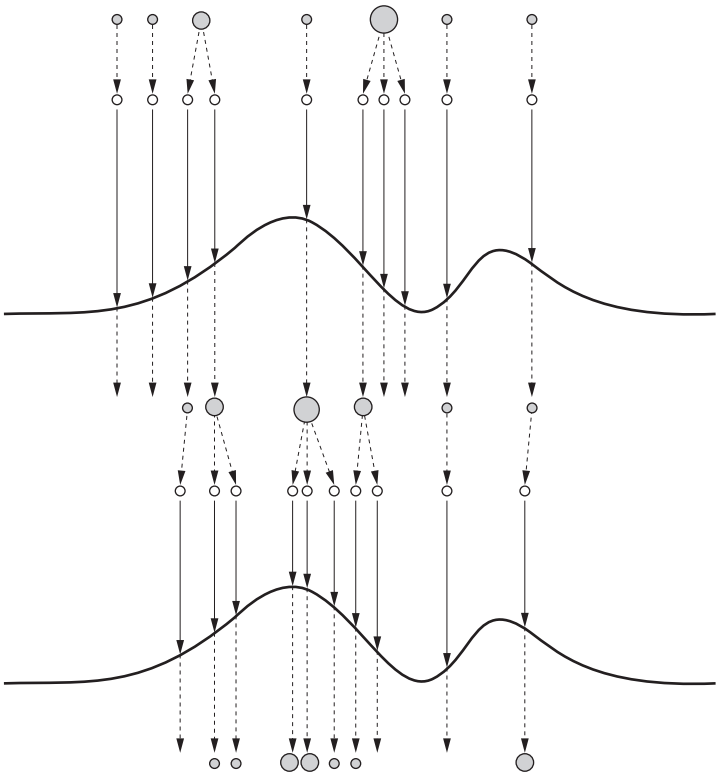


Figure 15.5 Illustration of the particle filtering algorithm. (Adapted with permission from van der Merwe et al. (2000a).) At each time slice, we begin with a set of weighted samples (dark circles), we sample from them to generate a set of unweighted samples (light circles). We propagate each sample forward through the system dynamics, and we update the weight of each sample to reflect the likelihood of the evidence (black line), producing a new set of weighted samples (some of which have weight so small as to be invisible). The process then repeats for the next time slice.

15.3.3.3 Sequential Importance Sampling

We can generalize our forward sampling process by viewing it in terms of importance sampling, as in section 12.2.2. Here, however, we are sampling entire trajectories rather than static states. Our goal is to sample a trajectory $\bar{\boldsymbol{x}}^{(0:t)}$ from the distribution $P(\boldsymbol{x}^{(0:t)} \mid \boldsymbol{o}^{(0:t)})$. To use importance sampling, we must construct a proposal distribution α for trajectories and then use importance weights to correct for the difference between our proposal distribution and our target distribution.

To maintain the ability to execute our filtering algorithm in an online fashion, we must construct our proposal distribution so that trajectories are constructed incrementally. Assume that, at time t, we have sampled some set of partial trajectories $\mathcal{D}^{(t)}$, each possibly associated with some weight. If we want to avoid the need to maintain full trajectories and a full observation

Algorithm 15.4 Particle filtering for DBNs

Procedure Particle-Filter-DBN (
  $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$, // DBN
  M // Number of samples
  $\boldsymbol{o}^{(1)}, \boldsymbol{o}^{(2)}, \ldots$ // Observation sequence
)

for $m = 1, \ldots, M$
  Sample $\bar{\boldsymbol{x}}^{(0)}[m]$ from $\mathcal{B}_0$
  $w^{(0)}[m] \leftarrow 1/M$
for $t = 1, 2, \ldots$
  for $m = 1, \ldots, M$
    Sample $\bar{\boldsymbol{x}}^{(0:t-1)}$ from the distribution $\hat{P}_{\mathcal{D}^{(t-1)}}$.
      // Select sample for propagation
    $(\bar{\boldsymbol{x}}^{(t)}[m], w^{(t)}[m]) \leftarrow$ LW-2TBN$(\mathcal{B}_{\rightarrow}, \bar{\boldsymbol{x}}^{(t-1)}, \boldsymbol{o}^{(t)})$
      // Generate time t sample and weight from selected sample $\bar{\boldsymbol{x}}^{(t-1)}$
  $\mathcal{D}^{(t)} \leftarrow \{(\bar{\boldsymbol{x}}^{(0:t)}[m], w^{(t)}[m]) : m = 1, \ldots, M\}$
  $\hat{\sigma}^{(t)}(\boldsymbol{x}) \leftarrow \hat{P}_{\mathcal{D}^{(t)}}$

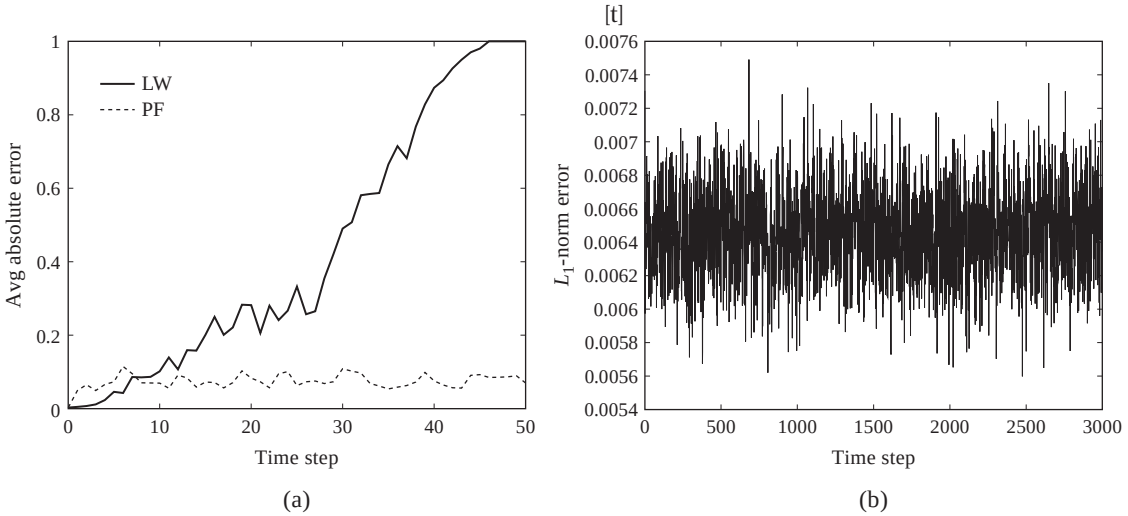


Figure 15.6 Likelihood weighting and particle filtering over time. (a) A comparison for 1,000 time slices. (b) A very long run of particle filtering.

history, each of our proposed sample trajectories for time $t+1$ must be an extension of one of our time t sample trajectories. More precisely, each proposed trajectory at time $t+1$ must have the form $\bar{\boldsymbol{x}}^{(0:t)}, \boldsymbol{x}^{(t+1)}$, for some $\bar{\boldsymbol{x}}^{(0:t)}[m] \in \mathcal{D}^{(t)}$.

Note that this requirement, while desirable from a computational perspective, does have disadvantages. In certain cases, our set of time t sample trajectories might be unrepresentative of the true underlying distribution; this might occur simply because of bad luck in sampling, or because our evidence sequence up to time t was misleading, causing us to select for trajectories that turn out to be a bad match to later observations. Thus, it might be desirable to *rejuvenate* our sample trajectories, allowing the states prior to time t to be modified based on evidence observed later on. However, this type of process is difficult to execute efficiently, and is not often done in practice.

If we proceed under the previous assumption, we can compute the appropriate importance weights for our importance sampling process incrementally:

$$\begin{aligned} w(\bar{\boldsymbol{x}}^{(0:t)}) &= \frac{P(\boldsymbol{x}^{(0:t)} \mid \boldsymbol{o}^{(0:t)})}{\alpha^{(t)}(\boldsymbol{x}^{(0:t)})} \\ &= \frac{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t)})}{\alpha^{(t-1)}(\boldsymbol{x}^{(0:t-1)})} \frac{P(\bar{\boldsymbol{x}}^{(t)} \mid \boldsymbol{x}^{(0:t-1)}, \boldsymbol{o}^{(0:t)})}{\alpha^{(t)}(\bar{\boldsymbol{x}}^{(t)} \mid \boldsymbol{x}^{(0:t-1)})}. \end{aligned}$$

As we have discussed, the quality of an importance sampler is a function of the variance of the weights: the lower the variance, the better the sampler. Thus, we aim to choose our proposal distribution $\alpha^{(t)}(\bar{\boldsymbol{x}}^{(t)} \mid \bar{\boldsymbol{x}}^{(0:t-1)})$ so as to reduce the variance of the preceding expression. Note that only the second of the two terms in this product depends on our time t proposal distribution $\alpha^{(t)}(\bar{\boldsymbol{x}}^{(t)} \mid \bar{\boldsymbol{x}}^{(0:t-1)})$. By assumption, the samples $\boldsymbol{x}^{(0:t-1)}$ are fixed, and hence so is the first term. It is now not difficult to show that the time t proposal distribution that minimizes the overall variance is

$$\alpha^{(t)}(\boldsymbol{X}^{(t)} \mid \boldsymbol{x}^{(0:t-1)}) = P(\boldsymbol{X}^{(t)} \mid \boldsymbol{x}^{(0:t-1)}, \boldsymbol{o}^{(0:t)}), \tag{15.3}$$

making the second term uniformly 1. In words, we should sample the time t state variable assignment $\bar{\boldsymbol{x}}^{(t)}$ from its posterior distribution given the chosen sample from the previous state and the time t observations.

Using this proposal, the appropriate importance weight for our time t trajectory is

$$\frac{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t)})}{\alpha^{(t-1)}(\boldsymbol{x}^{(0:t-1)})}.$$

What is the proposal distribution we use for the time $t-1$ trajectories? If we use this idea in combination with resampling, we can make the approximation that our uniformly sampled particles at time $t-1$ are an approximation to $P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t-1)})$. In this case, we have

$$\begin{aligned} \frac{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t)})}{\alpha^{(t-1)}(\bar{\boldsymbol{x}}^{(0:t-1)})} &\approx \frac{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t)})}{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t-1)})} \\ &\propto \frac{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t-1)}) P(o^{(t)} \mid \boldsymbol{x}^{(0:t-1)}, \boldsymbol{o}^{(0:t-1)})}{P(\boldsymbol{x}^{(0:t-1)} \mid \boldsymbol{o}^{(0:t-1)})} \\ &= P(o^{(t)} \mid \bar{\boldsymbol{x}}^{(t-1)}), \end{aligned}$$

where the last step uses the Markov independence properties. Thus, our importance weights here are proportional to the probability of the time t observation given the time $t-1$ particle $\boldsymbol{x}^{(t-1)}$, marginalizing out the time t state variables. We call this approach *posterior particle filtering*, because the samples are generated using our *posterior* over the time t state, given the time t observation, rather than using our prior.

posterior particle filter

However, sampling from the posterior over the state variables given the time t observations may not be tractable. Indeed, the whole purpose of the (static) likelihood-weighting algorithm is to address this problem, defining a proposal distribution that is (perhaps) closer to the posterior while still allowing forward sampling according to the network structure. However, likelihood weighting is only one of many importance distributions that can be used in this setting. In many cases, significant advantages can be gained by constructing proposal distributions that are even closer to this posterior; we describe some ideas in section 15.3.3.5 below.

15.3.3.4 Sample Selection Scheme

We can also generalize the particle selection scheme. A general selection procedure associates with each particle $\bar{\boldsymbol{x}}^{(0:t)}[m]$ a number of *offspring* $K_m^{(t)}$. Each of the offspring of this particle is a (possibly weighted) copy of it, which is then propagated independently to the next step, as discussed in the previous section. Let $\mathcal{D}^{(t)}$ be the original sample set, and $\tilde{\mathcal{D}}^{(t)}$ be the new sample set.

Assuming that we want to keep the total number of particles constant, we must have $\sum_{m=1}^{M} K_m^{(t)} = M$. There are many approaches to selecting the number of offspring $K_m^{(t)}$ for each particle $\bar{\boldsymbol{x}}^{(0:t)}[m]$. The bootstrap filter implicitly selects $K_m^{(t)}$ using a multinomial distribution, where one performs M IID trials, in each of which we obtain the outcome m with probability $w(\bar{\boldsymbol{x}}^{(0:t)}[m])$ (assuming the weights have been renormalized). This distribution guarantees that the expectation of $K_m^{(t)}$ is $M \cdot w(\bar{\boldsymbol{x}}^{(0:t)}[m])$. Because each of the particles in $\tilde{\mathcal{D}}^{(t)}$ is weighed equally, this property guarantees that the expectation (relative to our random resampling procedure) of $\hat{P}_{\tilde{\mathcal{D}}^{(t)}}$ is our original distribution $\hat{P}_{\mathcal{D}^{(t)}}$. Thus, the resampling procedure does not introduce any bias into our algorithm, in the sense that the expectation of any estimator relative to $\hat{P}_{\mathcal{D}^{(t)}}$ is the same as its expectation relative to $\hat{P}_{\tilde{\mathcal{D}}^{(t)}}$.

While the multinomial scheme is quite natural, there are many other selection schemes that also satisfy this constraint. In general, we can use some other method to select the number of offspring $K_m^{(t)}$ for each sample m, so long as this number satisfies (perhaps approximately) the constraint on the expectation. Assuming $K_m^{(t)} > 0$, we then assign the weight of each of these $K_m^{(t)}$ offspring to be:

$$\frac{w(\bar{\boldsymbol{x}}^{(0:t)}[m])}{K_m^{(t)} \Pr(K_m^{(t)} > 0)};$$

intuitively, we divide the original weight of the mth sample between its $K_m^{(t)}$ offspring. The second term in the denominator accounts for the fact that the sample was not eliminated entirely. To justify this expression, we observe that the total weight of the sample m offspring *conditioned on the fact that* $K_m^{(t)} > 0$ is precisely $w(\bar{\boldsymbol{x}}^{(0:t)}[m])$. Thus, the unconditional expectation of the total of these weights is $w(\bar{\boldsymbol{x}}^{(0:t)}[m])P(K_m^{(t)} > 0)$, causing us to divide by this latter probability

in the new weights.

There are many possible choices for generating the vector of offspring $(K_1^{(t)}, \ldots, K_M^{(t)})$, which tells us how many copies (if any) of each of the M samples in $\mathcal{D}^{(t)}$ we wish to propagate forward. Although the different schemes all have the same expectation, they can differ significantly in terms of their variance. The higher the variance, the greater the probability of obtaining unrepresentative distributions, leading to poor answers. The multinomial sampling scheme induced by the bootstrap filter tends to have a fairly high variance, and other schemes often perform better in practice. Moreover, it is not necessarily optimal to perform a resampling step at every time slice. For example, one can monitor the weights of the samples, and only resample when the variance exceeds a certain threshold, or, equivalently, when the number of effective samples equation (12.15) goes below a certain minimal amount.

Finally, we note that one can also consider methods that vary the number of samples M over time. In certain cases, such as tracking a system in real time, we may be forced to maintain rigid constraints on the amount of time spent in each time slice. In other cases, however, it may be possible to spend more computational resources in some time slices, at the expense of using less in others. For example, if we have the ability to cache our observations a few time slices back (which we may be doing in any case for the purposes of performing smoothing), we can allow more samples in one time slice (perhaps falling a bit behind), and catch up in a subsequent time slice. If so, we can determine whether additional samples are required for the current time slice by using our estimate of the number of effective samples in the current time slice. Empirically, this flexibility in the number of samples used per time slice can also improve the quality of the results, since it helps reduce the variance of the estimator and thereby reduces the harm done by a poor set of samples obtained at one time slice.

15.3.3.5 Other Extensions

As for importance sampling in the static case, there are multiple ways in which we can improve particle filtering by utilizing other inference methods. For example, a key problem in particle filtering is the fact that the diversity of particles often decreases over time, so that we are only generating samples from a relatively small part of our space. In cases where there are multiple reasonably likely hypotheses, this loss of diversity can result in bad situations, where a surprising observation (surprising relative to our current sample population) can suddenly rule out all or most of our samples.

There are several ways of addressing that problem. For example, we can use MCMC methods within a time slice to obtain a more diverse set of samples. While this cannot regenerate hypotheses that are very far away from our current set of samples, it can build up and maintain a broader set of hypotheses that is less likely to become depleted in subsequent steps. A related approach is to generate a clique tree for the time slice in isolation, and then use forward sampling to generate samples from the clique tree (see exercise 12.3). We note that exact inference for a single time slice may be feasible, even if it is infeasible for the DBN as a whole due to the entanglement phenomenon.

Another use for alternative inference methods is to reduce the variance of the generated samples. Here also, multiple approaches are possible. For example, as we discussed in section 15.3.3.3 (in equation (15.3)), we want to generate our time t state variable assignment from its posterior distribution given the chosen sample from the previous state and the time t ob-

servations. We can generate this posterior using exact inference on the 2-TBN structure. Again, this approach may be feasible even if exact inference on the DBN is not. If exact inference is infeasible even for the 2-TBN, we may still be able to use some intermediate alternative. We might be able to reverse some edges that point to observed variables (see exercise 3.12), making the time t distribution closer to the optimal sampling distribution at time t. Alternatively, we might use approximate inference method (for example, the EP-based approach of the previous section) to generate a proposal distribution that is closer to the true posterior than the one obtained by the simple likelihood-weighting sampling distribution.

Rao-Blackwellized particle filter

Finally, we can also use collapsed particles rather than fully sampled states in particle filtering. This method is often known as *Rao-Blackwellized particle filtering*, or *RBPF*. As we observed in section 12.4, the use of collapsed particles reduces the bias of the estimator. The procedure is based on the collapsed importance-sampling procedure for static networks, as described in section 12.4.1. As there, we partition the state variables $\boldsymbol{X}$ into two disjoint sets: the sampled variables $\boldsymbol{X}_p$, and the variables $\boldsymbol{X}_d$ whose distribution we maintain in closed form. Each particle now consists of three components: $(\boldsymbol{x}_p^{(t)}[m], w^{(t)}[m], q^{(t)}[m](\boldsymbol{X}_d^{(t)}))$. The particle structure is generally chosen so as to allow $q^{(t)}[m](\boldsymbol{X}_d^{(t)})$ to be represented effectively, for example, in a factorized form.

At a high level, we use importance sampling from some appropriate proposal distribution Q (as described earlier) to sample the variables $\boldsymbol{X}_p^{(t)}$ and exact inference to compute the importance weights and the distribution $q^{(t)}[m](\boldsymbol{X}_d^{(t)})$. This process is described in detail in section 12.4.1. When applying this procedure in the context of particle filtering, we generate the time t particle from a distribution defined by a time $t-1$ particle and the 2-TBN.

More precisely, consider a time $t-1$ particle $\boldsymbol{x}_p^{(t-1)}[m], w^{(t-1)}[m], q^{(t-1)}[m](\boldsymbol{X}_d^{(t-1)})$. We define a joint probability distribution $P_m^{(t)}(\boldsymbol{X}^{(t-1)} \cup \mathcal{X}^{(t)})$ by taking the time $t-1$ particle $\boldsymbol{x}_p^{(t-1)}[m], q^{(t-1)}[m](\boldsymbol{X}_d^{(t-1)})$ as a distribution over $\boldsymbol{X}^{(t-1)}$ (one that gives probability 1 to $\boldsymbol{x}_p^{(t-1)}[m]$), and then using the 2-TBN to define $P(\mathcal{X}^{(t)} \mid \boldsymbol{X}^{(t-1)})$. The distribution $P_m^{(t)}$ is represented in a factored form, which is derived from the factorization of $q^{(t-1)}[m](\boldsymbol{X}_d^{(t-1)})$ and from the structure of the 2-TBN. We can now use $P_m^{(t)}$, and the time t observation $o^{(t)}$, as input to the procedure of section 12.4.1. The output is a new particle and weight w; the particle is added to the time t data set $\mathcal{D}^{(t)}$, with a weight $w \cdot w^{(t-1)}[m]$. The resulting data set, consisting now of collapsed particles, is then utilized as in standard particle filtering. In particular, an additional sample selection step may be used to choose which particles are to be propagated to the next time step.

As defined, however, the collapsed importance-sampling procedure over $P_m^{(t)}$ computes a particle over all of the variables in this model: both time t and time $t-1$ variables. For our purpose, we need to extract a particle involving only time t variables. This process is fairly straightforward. The particle specifies as assignment to $\boldsymbol{X}_p^{(t)}$; the marginal distribution over $\boldsymbol{X}_d^{(t)}$ can be extracted using standard probabilistic inference techniques. We must take care, however: in general, the distribution over $\boldsymbol{X}_d$ can be subject to the same entanglement phenomena as the distribution as a whole. Thus, we must select the factorization (if any) of $q^{(t)}[m](\boldsymbol{X}_d^{(t)})$ so as to be sustainable over time; that is, $q^{(t-1)}[m](\boldsymbol{X}_d^{(t-1)})$ factorizes in a certain way, then so does the marginal distribution $q^{(t)}[m](\boldsymbol{X}_d^{(t)})$ induced by $P_m^{(t)}$. Box 15.A

provides an example of such a model for the application of collapsed particle filtering to the task of robot localization and mapping.

15.3.4 Deterministic Search Techniques

Random sampling methods such as particle filtering are not always the best approach for generating particles that search the space of possibilities. In particular, if the transition model is discrete and highly skewed — some successor states have much higher probability than others — then a random sampling of successor states is likely to generate many identical samples. This greatly reduces sample diversity, wastes computational resources, and leads to a poor representation of the space of possibilities. In this case, *search*-based methods may provide a better alternative. Here, we aim to find a set of particles that span the high-probability assignments and that we hope will allow us to keep track of the most likely trajectories through the system.

search

speech recognition

These techniques are commonly used in applications such as *speech recognition* (see box 6.B), where the transitions between phonemes, and even between words, are often highly constrained, with most transitions having probability (close to) 0. Here, we often formulate the problem as that of finding the single highest-probability trajectory through the system. In this case, an exact solution can be found by running a variable elimination algorithm such as that of section 13.2. In the context of HMMs, this algorithm is known as the *Viterbi algorithm*.

Viterbi algorithm

In many cases, however, the HMM for speech recognition does not fit into memory. Moreover, if our task is continuous speech recognition, there is no natural end to the sequence. In such settings, we often resort to approximate techniques that are more memory efficient. A commonly used technique is beam search, which has the advantage that it can be applied in an online fashion as the data sequence is acquired. See exercise 15.10.

Finally, we note that deterministic search in temporal models is often used within the framework of collapsed particles, combining search over a subset of the variables with marginalization over others. This type of approach provides an approximate solution to the marginal MAP problem, which is often a more appropriate formulation of the problem. For example, in the speech-recognition problem, the MAP solution finds the most likely trajectory through the speech HMM. However, this complete trajectory tells us not only which words were spoken in a given utterance, but also which phones and subphones were traversed; we are rarely interested in the trajectory through these finer-grained states. A more appropriate goal is to find the most likely sequence of words when we marginalize over the possible sequences of phonemes and subphones. Methods such as beam search can also be adapted to the marginal MAP problem, allowing it to be applied in this setting; see exercise 15.10.

15.4 Hybrid DBNs

So far, we have focused on inference in the context of discrete models. However, many (perhaps even most) dynamical systems tend to include continuous, as well as discrete, variables. From a representational perspective, there is no difficulty incorporating continuous variables into the network model, using the techniques described in section 5.5. However, as usual, inference in models incorporating continuous variables poses new challenges. In general, the techniques developed in chapter 14 for the case of static networks also extend to the case of DBNs, in the

same way that we extended inference techniques for static discrete networks in earlier sections in this chapter.

We now describe a few of the combinations, focusing on issues that are specific to the combination between DBNs and hybrid models. We emphasize that many of the other techniques described in this chapter and in chapter 14 can be successfully combined. For example, one of the most popular combinations is the application of particle filtering to continuous or hybrid systems; however, the combination does not raise significant new issues, and so we omit a detailed presentation.

15.4.1 Continuous Models

We begin by considering systems composed solely of continuous variables.

15.4.1.1 The Kalman Filter

The simplest such system is the linear dynamical system (see section 6.2.3.2), where the variables are related using linear Gaussian CPDs. These systems can be tracked very efficiently using a set of update equations called the *Kalman filter*.

Recall that the key difficulty with tracking a dynamical system is the entanglement phenomenon, which generally forces us to maintain, as our belief state, a full joint distribution over the state variables at time t. For discrete systems, this distribution has size exponential in the number of variables, which is generally intractably large. By contrast, as a linear Gaussian network defines a joint Gaussian distribution, and Gaussian distributions are closed under conditioning and marginalization, we know that the posterior distribution over any subset of variables given any set of observations is a Gaussian. In particular, the belief state over the state variables $\boldsymbol{X}^{(t)}$ is a multivariate Gaussian. A Gaussian can be represented as a mean vector and covariance matrix, which requires (at most) quadratic space in the number of state variables. Thus, in a Kalman filter, we can represent the belief state fairly compactly.

As we now show, we can also maintain the belief state efficiently, using simple matrix operations over the matrices corresponding to the belief state, the transition model, and the observation model. Specifically, consider a linear Gaussian DBN defined over a set of state variables $\boldsymbol{X}$ with $n = |\boldsymbol{X}|$ and a set of observation variables O with $m = |O|$. Let the probabilistic model be defined as in equation (6.3) and equation (6.4), which we review for convenience:

$$\begin{aligned} P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(t-1)}) &= \mathcal{N}\left(A\boldsymbol{X}^{(t-1)}; Q\right), \\ P(O^{(t)} \mid \boldsymbol{X}^{(t)}) &= \mathcal{N}\left(H\boldsymbol{X}^{(t)}; R\right). \end{aligned}$$

Kalman filter

state transition update

We now show the *Kalman filter* update equations, which provide an efficient implementation for equation (15.1) and equation (15.2). Assume that the Gaussian distribution encoding $\sigma^{(t)}$ is maintained using a mean vector $\mu^{(t)}$ and a covariance distribution $\Sigma^{(t)}$. The *state transition update* equation is easy to implement:

$$\begin{aligned} \mu^{(\cdot t+1)} &= A\mu^{(t)} \\ \Sigma^{(\cdot t+1)} &= A\Sigma^{(t)}A^T + Q, \end{aligned} \tag{15.4}$$

where $\mu^{(\cdot t+1)}$ and $\Sigma^{(\cdot t+1)}$ are the mean and covariance matrix for the prior belief state $\sigma^{(\cdot t+1)}$. Intuitively, the new mean vector is simply the application of the linear transformation A to the mean vector in the previous time step. The new covariance matrix is the transformation of the previous covariance matrix via A, plus the covariance introduced by the noise.

observation update

The *observation update* is somewhat more elaborate:

$$\begin{aligned} K^{(t+1)} &= \Sigma^{(\cdot t+1)} H^T (H\Sigma^{(\cdot t+1)} H^T + R)^{-1} \\ \mu^{(t+1)} &= \mu^{(\cdot t+1)} + K^{(t+1)}(o^{(t+1)} - H\mu^{(\cdot t+1)}) \\ \Sigma^{(t+1)} &= (I - K^{(t+1)} H)\Sigma^{(\cdot t+1)}. \end{aligned} \tag{15.5}$$

This update rule can be obtained using tedious but straightforward algebraic manipulations, by forming the joint Gaussian distribution over $\boldsymbol{X}^{(t+1)}, \boldsymbol{O}^{(t+1)}$ defined by the prior belief state $\sigma^{(\cdot t+1)}$ and the observation model $P(\boldsymbol{O}^{(t+1)} \mid \boldsymbol{X}^{(t+1)})$, and then conditioning the resulting joint Gaussian on the observation $\boldsymbol{o}^{(t+1)}$.

To understand the intuition behind this rule, note first that the mean of $\sigma^{(t+1)}$ is simply the mean of $\sigma^{(\cdot t+1)}$, plus a correction term arising from the evidence. The correction term involves the *observation residual* — the difference between our *expected observation* $H\mu^{(\cdot t+1)}$ and the actual observation $o^{(t+1)}$. This residual is multiplied by a matrix called the *Kalman gain* $K^{(t+1)}$, which dictates the importance that we ascribe to the observation. We can see, for example, that when the measurement error covariance R approaches 0, the Kalman gain approaches H^{-1}; in this case, we are exactly "reverse engineering" the residual in the observation and using it to correct the belief state mean. Thus, the actual observation is trusted more and more, and the predicted observation $H\mu^{(\cdot t+1)}$ is trusted less. We also then have that the covariance of the new belief state approaches 0, corresponding to the fact that the observation tells us the current state with close to certainty. Conversely, when the covariance in our belief state $\Sigma^{(\cdot t+1)}$ tends to 0, the Kalman gain approaches 0 as well. In this case, we trust our predicted distribution, and pay less and less attention to the observation: both the mean and the covariance of the posterior belief state $\sigma^{(t+1)}$ are the same as those of the prior belief state $\sigma^{(\cdot t+1)}$. Finally, it can be shown that the posterior covariance matrix of our estimate approaches some limiting value as $T \longrightarrow \infty$, which reflects our "steady state" uncertainty about the system state. We note that both the time t covariance and its limiting value do not depend on the data. On one hand, this fact offers computational savings, since it allows the covariance matrix to be computed offline. However, it also points to a fundamental weakness of linear-Gaussian models, since we would naturally expect our uncertainty to depend on what we have seen.

The Kalman filtering process maintains the belief state as a mean and covariance matrix. An alternative is to maintain the belief state using information matrices (that is, a canonical form representation, as in equation (14.1)). The resulting set of update equations, called the *information form* of the Kalman filter, can be derived in a straightforward way from the basic operations on canonical forms described in section 14.2.1.2; the details are left as an exercise (exercise 15.11). We note that, in the Kalman filter, which maintains covariance matrices, the state transition update (equation (15.4)) is straightforward, and the observation update (equation (15.5)) is complex, requiring the inversion of an $n \times n$ matrix. In the information filter, which maintains information matrices, the situation is precisely the reverse.

information form

15.4.1.2 Nonlinear Systems

The Kalman filter can also be extended to deal with nonlinear continuous dynamics, using the techniques described in section 14.4. In these methods, we maintain all of the intermediate factors arising in the course of inference as multivariate Gaussian distributions. When encountering a nonlinear CPD, which would give rise to a non-Gaussian factor, we simply linearize the result to produce a new Gaussian. We described two main methods for performing the linearization, either by taking the Taylor series expansion of the nonlinear function, or by using one of several numerical integration techniques. The same methods apply without change to the setting of tracking nonlinear continuous DBNs. In this case, the application is particularly straightforward, as the factors that we wish to manipulate in the course of tracking are all distributions; in a general clique tree, some factors do not represent distributions, preventing us from applying these linearization techniques and constraining the order in which messages are passed.

Concretely, assume that our nonlinear system has the model:

$$\begin{aligned} P(\boldsymbol{X}^{(t)} \mid \boldsymbol{X}^{(t-1)}) &= f(\boldsymbol{X}^{(t-1)}, \boldsymbol{U}^{(t-1)}) \\ P(O^{(t)} \mid \boldsymbol{X}^{(t)}) &= g(\boldsymbol{X}^{(t)}, \boldsymbol{W}^{(t)}), \end{aligned}$$

where f and g are deterministic nonlinear (continuous) functions, and $\boldsymbol{U}^{(t)}, \boldsymbol{W}^{(t)}$ are Gaussian random variables, which explicitly encode the noise in the transition and observation models, respectively. (In other words, rather than modeling the system in terms of stochastic CPDs, we use an equivalent representation that partitions the model into a deterministic function and a noise component.)

extended Kalman filter

unscented Kalman filter

To address the filtering task here, we can apply either of the linearization methods described earlier. The Taylor-series linearization of section 14.4.1.1 gives rise to a method called the *extended Kalman filter*. The unscented transformation of section 14.4.1.2 gives rise to a method called the *unscented Kalman filter*. In this latter approach, we maintain our belief state using the same representation as in the Kalman filter: $\sigma^{(t)} = \mathcal{N}\left(\mu^{(t)}; \Sigma^{(t)}\right)$. To perform the transition update, we construct a joint Gaussian distribution $p(\boldsymbol{X}^{(t)}, \boldsymbol{U}^{(t)})$ by multiplying the Gaussians for $\sigma^{(t)}$ and $\boldsymbol{U}^{(t)}$. The result is a Gaussian distribution and a nonlinear function f, to which we can now apply the unscented transformation of section 14.4.1.2. The result is a mean vector $\mu^{(\cdot t+1)}$ and covariance matrix $\Sigma^{(\cdot t+1)}$ for the prior belief state $\sigma^{(\cdot t+1)}$.

To obtain the posterior belief state, we must perform the observation update. We construct a joint Gaussian distribution $p(\boldsymbol{X}^{(t+1)}, \boldsymbol{W}^{(t+1)})$ by multiplying the Gaussians for $\sigma^{(\cdot t+1)}$ and $\boldsymbol{W}^{(t+1)}$. We then estimate a *joint* Gaussian distribution over $\boldsymbol{X}^{(t+1)}, O^{(t+1)}$, using the unscented transformation of equation (14.18) to estimate the integrals required for computing the mean and covariance matrix of this joint distribution. We now have a Gaussian joint distribution over $\boldsymbol{X}^{(t+1)}, O^{(t+1)}$, which we can condition on our observation $o^{(t+1)}$ in the usual way. The resulting posterior over $\boldsymbol{X}^{(t+1)}$ is the new belief state $\sigma^{(t+1)}$. Note that this approach computes a full joint covariance matrix over $\boldsymbol{X}^{(t+1)}, O^{(t+1)}$. When the dependency model of the observation on the state is factored, where we have individual observation variables each of which depends only on a few state variables, we can perform this computation in a more structured way (see exercise 15.12).

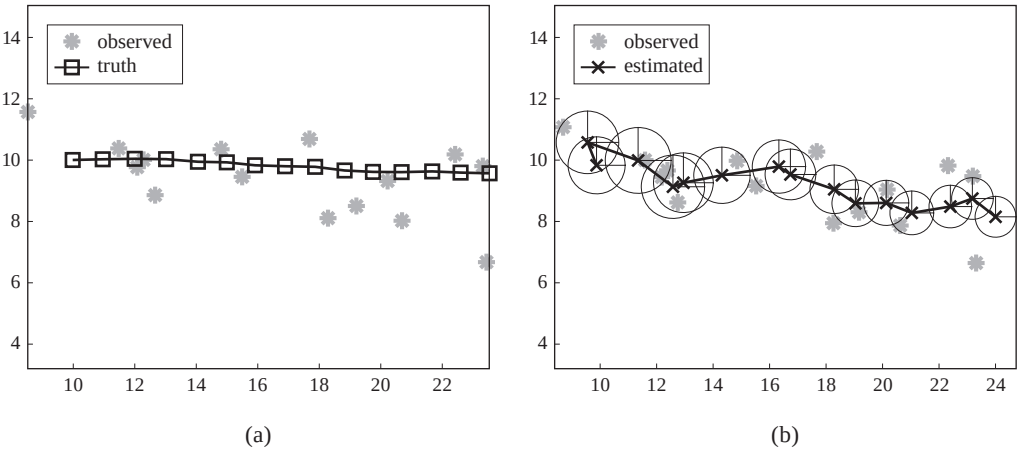


Figure 15.A.1 — Illustration of Kalman filtering for tracking (a) Raw data (dots) generated by an object moving to the right (line). (b) Estimated location of object: crosses are the posterior mean, circles are 95 percent confidence ellipsoids.

Box 15.A — Case Study: Tracking, Localization, and Mapping. *A key application of probabilistic models is to the task of tracking moving objects from noisy measurements. One example is* target tracking*, where we measure the location of an object, such as an airplane, using an external sensor. Another example is* robot localization*, where the moving object itself collects measurements, such as sonar or laser data, that can help it localize itself on a map.*

target tracking

robot localization

Kalman filtering was applied to this problem as early as the 1960s. Here, we give a simplified example to illustrate the key ideas. Consider an object moving in a two-dimensional plane. Let $X_1^{(t)}$ *and* $X_2^{(t)}$ *be the horizontal and vertical locations of the object, and* $\dot{X}_1^{(t)}$ *and* $\dot{X}_2^{(t)}$ *be the corresponding velocity. We can represent this as a state vector* $\boldsymbol{X}^{(t)} \in \mathbb{R}^4$*. Let us assume that the object is moving at constant velocity, but is "perturbed" by random Gaussian noise (for example, due to the wind). Thus we can define* $X_i' \sim \mathcal{N}\left(X_i + \dot{X}_i; \sigma^2\right)$ *for* $i = 1, 2$*. Assume we can obtain a (noisy) measurement of the location of the object but not its velocity. Let* $\boldsymbol{Y}^{(t)} \in \mathbb{R}^2$ *represent our observation, where* $\boldsymbol{Y}^{(t)} \sim \mathcal{N}\left((X_1^{(t)}, X_2^{(t)}); \Sigma_o\right)$*, where* Σ_o *is the covariance matrix that governs our observation noise model. Here, we do not necessarily assume that noise is added separately to each dimension of the object location. Finally, we need to specify our initial (prior) beliefs about the state of the object,* $p(\boldsymbol{X}^{(0)})$*. We assume that this distribution is also a Gaussian* $p(\boldsymbol{X}^{(0)}) = \mathcal{N}\left(\mu^{(0)}; \Sigma^{(0)}\right)$*. We can represent prior ignorance by making* $\Sigma^{(0)}$ *suitably "broad." These parameters fully specify the model, allowing us to apply the Kalman filter, as described in section 15.4.1.1.*

Figure 15.A.1 gives an example. The object moves to the right and generates an observation at each time step (such as a "blip" on a radar screen). We observe these blips and filter out the noise by using the Kalman filter; the resulting posterior distribution is plotted on the right. Our best guess

about the location of the object is the posterior mean, $\mathbf{E}_{\sigma^{(t)}}\left[\boldsymbol{X}_{1:2}^{(t)} \mid \boldsymbol{y}^{(1:t)}\right]$*, denoted as a cross. Our uncertainty associated with this estimate is represented as an ellipse that contains 95 percent of the probability mass. We see that our uncertainty goes down over time, as the effects of the initial uncertainty get "washed out." As we discussed, the covariance converges to some steady-state value, which will remain indefinitely.*

We have demonstrated this approach in the setting where the measurement is of the object's location, from an external measurement device. It is also applicable when the measurements are collected by the moving object, and estimate, for example, the distance of the robot to various landmarks on a map. If the error in the measured distance to the landmark has Gaussian noise, the Kalman filter approach still applies.

In practice, it is rarely possible to apply the Kalman filter in its simplest form. First, the dynamics and/or observation models are often nonlinear. A common example is when we get to observe the range and bearing to an object, but not its $X_1^{(t)}$ *and* $X_2^{(t)}$ *coordinates. In this case, the observation model contains some trigonometric functions, which are nonlinear. If the Gaussian noise assumption is still reasonable, we apply either the extended Kalman filter or the unscented Kalman filter to linearize the model. Another problem arises when the noise is non-Gaussian, for example, when we have clutter or outliers. In this case, we might use the multivariate T distribution; this solution gains robustness at the cost of computational tractability. An alternative is to assume that each observation comes from a mixture model; one component corresponds to the observation being generated by the object, and another corresponds to the observation being generated by the background. However, this model is now an instance of a conditional linear Gaussian, raising all of the computational issues associated with multiple modes (see section 15.4.2). The same difficulty arises if we are tracking multiple objects, where we are uncertain which observation was generated by which object; this problem is an instance of the* data association *problem (see box 12.D).*

data association

In nonlinear settings, and particularly in those involving multiple modes, another very popular alternative is to use particle filtering. This approach is particularly appealing in an online setting such as robot motion, where computations need to happen in real time, and using limited computational resources. As a simple example, assume that we have a known map, $\mathcal{M}$*. The map can be encoded as a* occupancy grid *— a discretized grid of the environment, where each square is* 1 *if the environment contains an obstacle at that location, and* 0 *otherwise. Or we can encode it using a more geometric representation, such as a set of line segments representing walls. We can represent the robot's location in the environment either in continuous coordinates, or in terms of the discretized grid (if we use that representation). In addition, the robot needs to keep track of its pose, or orientation, which we may also choose to discretize into an appropriate number of bins. The measurement* $\boldsymbol{y}^{(t)}$ *is a vector of measured distances to the nearest obstacles, as described in box 5.E. Our goal is to maintain* $P(\boldsymbol{X}^{(t)} \mid \boldsymbol{y}^{(1:t)}, \mathcal{M})$*, which is our posterior over the robot location.*

Note that our motion model is nonlinear. Moreover, although the error model on the measured distance is a Gaussian around the true distance, the true distance to the nearest obstacle (in any given direction) is not even a continuous function of the robot's position. In fact, belief states can easily become multimodal owing to perceptual aliasing*, that is, when the robot's percepts can match two or more locations. Thus, a Gaussian model is a very poor approximation here.*

Thrun et al. (2000) propose the use of particle filtering for localization, giving rise to an algorithm called Monte Carlo localization*. Figure 15.A.2 demonstrates one sample trajectory of the particles over time. We see that, as the robot acquires more measurements, its belief state becomes more*

Monte Carlo localization

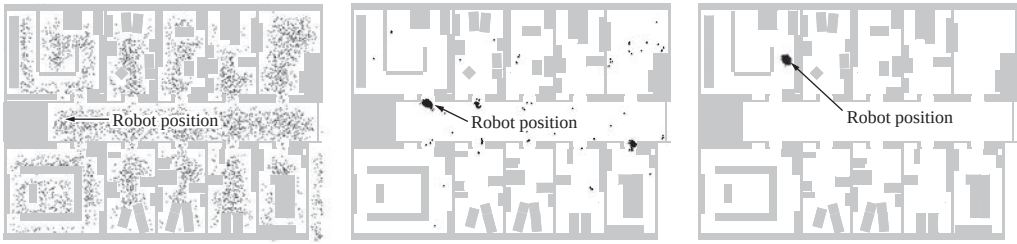


Figure 15.A.2 — Sample trajectory of particle filtering for robot localization

sharply peaked. More importantly, we see that the use of a particle-based belief state makes it easy to model multimodal posteriors.

One important issue when implementing a particle filter is the choice of proposal distribution. The simplest method is to use the standard bootstrap filter, where we propose directly from the dynamics model, and weight the proposals by how closely they match the evidence. However, when the robot is lost, so that our current belief state is very diffuse, this approach will not work very well, since the proposals will literally be all over the map but will then get "killed off" (given essentially zero weight) if they do not match the (high-quality) measurements. As we discussed, the ideal solution is to use posterior particle filtering, where we sample a particle $\boldsymbol{x}^{(t+1)}[m]$ from the posterior distribution $P(\mathcal{X}^{(t+1)} \mid \boldsymbol{x}^{(t)}[m], \boldsymbol{y}^{(t+1)})$. However, this solution requires that we be able to invert the evidence model using Bayes rule, a process that is not always feasible for complex, nonlinear models. One ad hoc fix is to inflate the noise level in the measurement model artificially, giving particles an artificially high chance of surviving until the belief state has the chance to adapt to the evidence. A better approach is to use a proposal that takes the evidence into account; for example, we can compare $\boldsymbol{y}^{(t)}$ with the map and then use a proposal that is a mixture of the bootstrap proposal $P(\mathcal{X}^{(t+1)} \mid \boldsymbol{x}^{(t)}[m])$ and some set of candidate locations that are most consistent with the recent observations.

robot mapping

SLAM

We now turn to the harder problem of localizing a robot in an unknown environment, while mapping *the environment at the same time. This problem is known as* simultaneous localization and mapping (SLAM). *In our previous terminology, this task corresponds to computing $p(\boldsymbol{X}^{(t)}, \mathcal{M} \mid \boldsymbol{y}^{(1:t)})$. Here, again, our task is much easier in the linear setting, where we represent the map in terms of K landmarks whose locations, denoted $L_1, \ldots, L_k$, are now unknown. Assume that we have a Gaussian prior over the location of each landmark, and that our observations $Y_k^{(t)}$ measure the Euclidean distance between the robot position $\boldsymbol{X}^{(t)}$ and the kth landmark location L_k, with some Gaussian noise. It is not difficult to see that $P(Y_k^{(t)} \mid \boldsymbol{X}^{(t)}, L_k)$ is a Gaussian distribution, so that our entire model is now a linear dynamical system. Therefore, we can naturally apply a Kalman filter to this task, where now our belief state represents $P(\boldsymbol{X}^{(t)}, L_1, \ldots, L_k \mid \boldsymbol{y}^{(1:t)})$. Figure 15.A.3a demonstrates this process for a simple two-dimensional map. We can see that the uncertainty of the landmark locations is larger for landmarks encountered later in the process, owing to the accumulation of uncertainty about the robot location. However, when the robot closes the loop and reencounters the first landmark, the uncertainty about its position reduces dramatically; the*

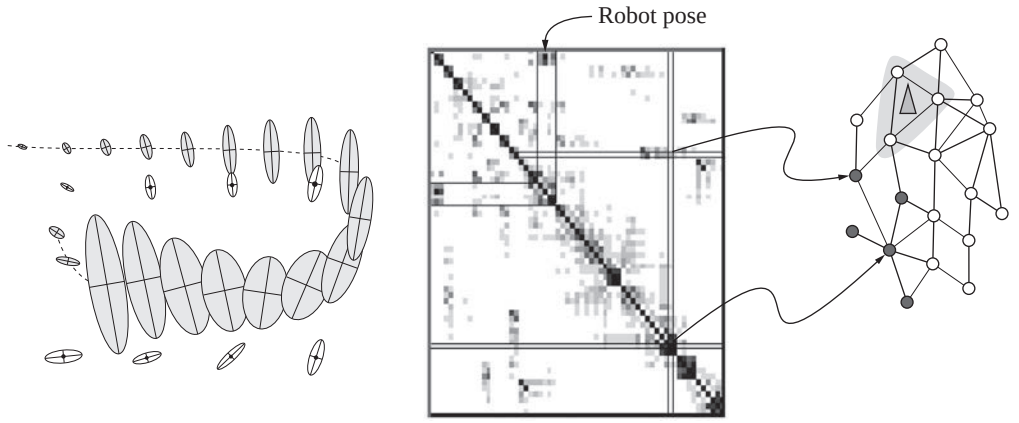


Figure 15.A.3 — Kalman filters for the SLAM problem (a) Visualization of marginals in Gaussian SLAM. (b) Visualization of information (inverse covariance) matrix in Gaussian SLAM, and its Markov network structure; very few entries have high values.

landmark uncertainty reduces at the same point. If we were to use a smoothing algorithm, we would also be able to reduce much of the uncertainty about the robot's intermediate locations, and hence also about the intermediate landmarks.

Gaussian inference is attractive in this setting because even representing a posterior over the landmark positions would require exponential space in the discrete case. Here, because of our ability to use Gaussians, the belief state grows quadratically only in the number of landmarks. However, for large maps, even quadratic growth can be too inefficient, particularly in an online setting. To address this computational burden, two classes of methods based on approximate inference have been proposed.

The first is based on factored belief-state idea of section 15.3.2. Here, we utilize the observation that, although the variables representing the different landmark locations become correlated due to entanglement, the correlations between them are often rather weak. In particular, two landmarks that were just observed at the same time become correlated due to uncertainty about the robot position. But, as the robot moves, the direct correlation often decays, and two landmarks often become almost conditionally independent given some subset of other landmarks; this conditional independence manifests as sparsity in the information (inverse covariance) matrix, as illustrated in figure 15.A.3b. Thus, these approaches approximate the belief state by using a sparser representation that maintains only the strong correlations. We note that the set of strongly correlated landmark pairs changes over time, and hence the structure of our approximation must be adaptive. We can consider a range of sparser representations for a Gaussian distribution. One approach is to use a clique tree, which admits exact M-projection operations but grows quadratically in the maximum size of cliques in our clique tree. Another is to use the Markov network representation of the Gaussian

(or, equivalently, its inverse covariance). The two main challenges are to determine dynamically the approximation to use at each step, and to perform the (approximate) M-projection in an efficient way, essential in this real-time setting.

A second approach, and one that is more generally applicable, is to use collapsed particles. This approach is based on the important observation that the robot makes independent observations of the different landmarks. Thus, as we can see in figure 15.A.4a, the landmarks are conditionally independent given the robot's trajectory. Importantly, the landmarks are not independent given the robot's current location, owing to entanglement, but if we sample an entire robot trajectory $\boldsymbol{x}^{(1:t)}$*, we can now maintain the conditional distribution over the landmark positions in a fully factored form. In this approach, known as* FastSLAM*, each particle is associated with a (factorized) distribution over maps. In this approach, rather than maintaining a Gaussian of dimension* $2K+2$ *(two coordinates for each landmark and two for the robot), we maintain a set of particles, each associated with* K *two-dimensional Gaussians. Because the Gaussian representation is quadratic in the dimension and the matrix inversion operations are cubic in the dimension, this approach provides considerable savings. (See exercise 15.13.) Experimental results show that, in the settings where Kalman filtering is suitable, this approximation achieves almost the same performance with a small number of particles (as few as ten); see figure 15.A.4b.*

This approach is far more general than the sparse Kalman-filtering approach we have described, since it also allows us to deal with other map representations, such as the occupancy-grid map described earlier. Moreover, it also provides an integrated solution in cases where we have uncertainty about data association; here, we can sample over data-association hypotheses and still maintain a factored distribution over the map representation. Overall, by combining the different techniques we have described, we can effectively address most instances of the SLAM problem, so that this problem is now essentially considered solved.

15.4.2 Hybrid Models

We now move to considering a system that includes both discrete and continuous variables. As in the static case, inference in such a model is very challenging, even in cases where the continuous dynamics are simple.

CLG dynamics

switching linear dynamical system

Consider a conditional linear Gaussian DBN, where all of the continuous variables have *CLG dynamics*, and the discrete variables have no continuous parents. This type of model is known as a *switching linear dynamical system*, as it can be viewed as a dynamical system where changes in the discrete state cause the continuous (linear) dynamics to switch. The unrolled DBN is a standard CLG network, and, as in any CLG network, given a full assignment to the discrete variables, the distribution over the continuous variables (or any subset of them) is a multivariate Gaussian. However, if we are not given an assignment to the discrete variables, the marginal distribution over the continuous variables is a mixture of Gaussians, where the number of mixture components is exponential in the number of discrete variables. (This phenomenon is precisely what underlies the $\mathcal{NP}$-hardness proof for inference in CLG networks.)

In the case of a temporal model, this problem is even more severe, since the number of mixture components grows exponentially, and unboundedly, over time.

Example 15.6

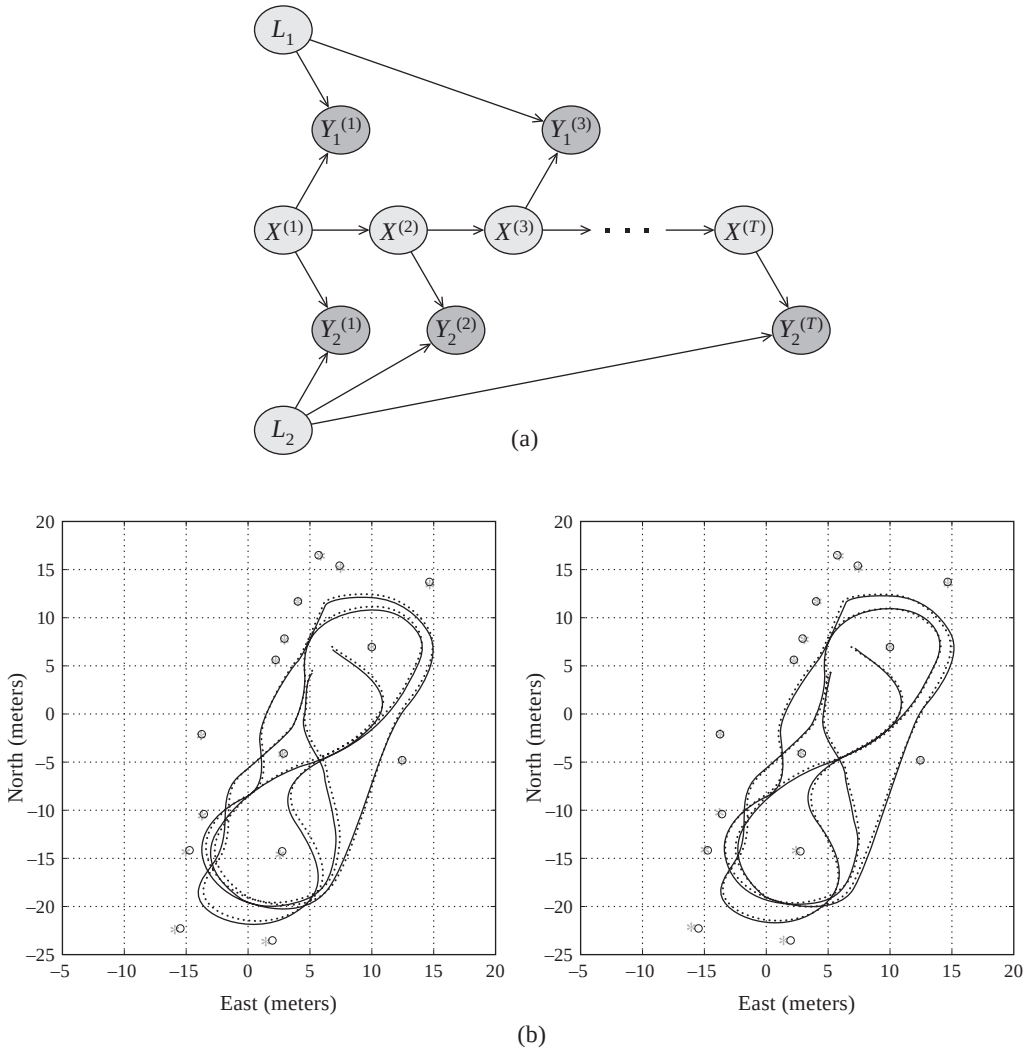


Figure 15.A.4 — Collapsed particle filtering for SLAM (a) DBN illustrating the conditional independencies in the SLAM probabilistic model. (b) Sample predictions using a SLAM algorithm (solid path, star landmarks) in comparison to ground-truth measurements obtained from a GPS sensor, not given as input to the algorithm (dotted lines, circle landmarks): left — EKF; right — FastSLAM. The results show that both approaches achieve excellent results, and that there is very little difference in the quality of the estimates between these two approaches.

Consider a one-dimensional switching linear-dynamical system whose state consists of a single discrete variable D and a single continuous variable X. In the SLDS, both X and D are persistent, and in addition we have $D' \rightarrow X'$. The transition model is defined by a CLG model for X', and a standard discrete CPD for D. In this example, the belief state at time t, marginalized over $X^{(t)}$, is a mixture distribution with a number of mixture components that is exponential in t. ■

In order to make the propagation algorithm tractable, we must make sure that the mixture of Gaussians represented by the belief state does not get too large. The two main techniques to do so are *pruning* and *collapsing*.

mixture pruning

Pruning algorithms reduce the size of the mixture distribution in the belief state by discarding some of its Gaussians. The standard pruning algorithm simply keeps the N Gaussians with the highest probabilities, discards the rest, and then renormalizes the probabilities of the remaining Gaussian to sum to 1. This is a form of *beam search* for marginal MAP, as described in exercise 15.10.

beam search

mixture collapsing

Collapsing algorithms partition the mixture of Gaussians into subsets, and then they collapse all the Gaussians in each subset into one Gaussian. Thus, if the belief state were partitioned into N subsets, the result would be a belief state with exactly N Gaussians. The different collapsing algorithms can differ in the choice of subsets of Gaussians to be collapsed, and on exactly when the collapsing is performed. The collapsed Gaussian $\hat{p}$ for a mixture p is generally chosen to be its M-projection — the one that minimizes $\boldsymbol{I}\boldsymbol{D}(p\|\hat{p})$. Recall from proposition 14.3 that the M-projection can be computed simply and efficiently by matching moments.

We now describe some commonly used collapsing algorithms in this setting and show how they can be viewed within the framework of expectation propagation, as described in section 14.3.3. More precisely, consider a CLG DBN containing the discrete state variables $\boldsymbol{A}$ and the continuous state variables $\boldsymbol{X}$. Let $M = |Val(\boldsymbol{A})|$ — the number of discrete states at any given time slice. Assume, for simplicity, that the state at time 0 is fully observed. We note that, throughout this section, we focus solely on techniques for CLG networks. It is not difficult to combine these techniques with linearization methods (as described earlier), allowing us to accommodate both nonlinear dynamics and discrete children of continuous parents (see section 14.4.2.5).

As we discussed, after t time slices, the total number of Gaussian mixture components in the belief state is M^t — one for every assignment to $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(t)}$. One common approximation is the class of *general pseudo-Bayesian* algorithms, which lump together components whose assignment in the recent past is identical. That is, for a positive integer k, $\sigma^{(t)}$ has M^{k-1} mixture components — one for every assignment to $\boldsymbol{A}^{((t-(k-2)):t)}$ (which for $k = 1$ we take to be a single "vacuous" assignment). If $\sigma^{(t)}$ has M^{k-1} mixture components, then one step of forward propagation results in a distribution with M^k components. Each of these Gaussian components is conditioned on the evidence, and its weight in the mixture is multiplied with the likelihood it gives the evidence. The resulting mixture is now partitioned into M^{k-1} subsets, and each subset is collapsed separately, producing a new belief state.

general pseudo-Bayes

GPB1

For $k = 1$, the algorithm, known as *GPB1*, maintains exactly one Gaussian in the belief state; it also maintains a distribution over $\boldsymbol{A}^{(t)}$. Thus, $\sigma^{(t)}$ is essentially a product of a discrete distribution $\sigma^{(t)}(\boldsymbol{A}^{(t)})$ and a Gaussian $\sigma^{(t)}(\boldsymbol{X}^{(t)})$. In the forward-propagation step (corresponding to equation (15.1)), we obtain a mixture of M Gaussians: one Gaussian $p^{(t+1)}_{\boldsymbol{a}^{(t+1)}}$

for each assignment $\boldsymbol{a}^{(t+1)}$, whose weight is

$$\sum_{\boldsymbol{A}^{(t)}} P(\boldsymbol{a}^{(t+1)} \mid \boldsymbol{A}^{(t)})\sigma^{(t)}(\boldsymbol{a}^{(t)}).$$

In the conditioning step, each of these Gaussian components is conditioned on the evidence $o^{(t+1)}$, and its weight is multiplied by the likelihood of the evidence relative to this Gaussian. The resulting weighted mixture is then collapsed into a single Gaussian, using the M-projection operation described in proposition 14.3.

expectation propagation

We can view this algorithm as an instance of the *expectation propagation* (EP) algorithm applied to the clique tree for the DBN described in section 15.2.3. The messages, which correspond to the belief states, are approximated using a factorization that decouples the discrete variables $\boldsymbol{A}^{(t)}$ and the continuous variables $\boldsymbol{X}^{(t)}$; the distribution over the continuous variables is maintained as a Gaussian. This approximation is illustrated in figure 15.7a. It is not hard to verify that the GPBl propagation update is precisely equivalent to the operation of incorporating the time t message into the $(t, t+1)$ clique, performing the in-clique computation, and then doing the M-projection to produce the time $t+1$ message.

GPB2

The *GPB2* algorithm maintains M Gaussians $\{p^{(t)}_{\boldsymbol{a}^{(t)}}\}$ in the time t belief state, each one corresponding to an assignment $\boldsymbol{a}^{(t)}$. After propagation, we get M^2 Gaussians, each one corresponding to an assignment to both $\boldsymbol{a}^{(t)}$ and $\boldsymbol{a}^{(t+1)}$. We now partition this mixture into M subsets, one for each assignment to $\boldsymbol{a}^{(t+1)}$. Each subset is collapsed, resulting in a new belief state with M Gaussians. Once again, we can view this algorithm as an instance of EP, using the message structure $\boldsymbol{A}^{(t)} \rightarrow \boldsymbol{X}^{(t)}$, where the distribution of $\boldsymbol{X}^{(t)}$ given $\boldsymbol{A}^{(t)}$ in the message has the form of a conditional Gaussian. This EP formulation is illustrated in figure 15.7b.

A limitation of the GPB2 algorithm is that, at every propagation step we must generate M^2 Gaussians, a computation that may be too expensive. An alternative approach is called the *interacting multiple model (IMM)* algorithm. Like GPB2, it maintains a belief state with M Gaussians $p^{(t)}_{\boldsymbol{a}^{(t)}}$; but like GPBl, it collapses all of the Gaussians in the mixture into a single Gaussian, prior to incorporating them into the transition model $\mathcal{P}(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)})$. Importantly, however, it performs the collapsing step after incorporating the discrete transition model $P(\boldsymbol{A}^{(t+1)} \mid \boldsymbol{A}^{(t)})$. This produces a different mixture distribution for each assignment $\boldsymbol{a}^{(t+1)}$ — these distributions are all mixtures of the same set of time t Gaussians, but with different mixture weights, generally producing a better approximation. Each of these components, representing a conditional distribution over $\boldsymbol{X}^{(t)}$ given $\boldsymbol{A}^{(t+1)}$, is then propagated using the continuous dynamics $P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)}, \boldsymbol{A}^{(t+1)})$, and conditioned on the evidence, producing the time $t+1$ belief state. The IMM algorithm can also be formulated in the EP framework, as illustrated in figure 15.7c.

Although we collapse our belief state in M different ways when using the IMM algorithm, we only create M Gaussians at time $t+1$ (as opposed to M^2 Gaussians in GPB2). The extra work compared to GPBl is the computation of the new mixture probabilities and collapsing M mixtures instead of just one, but usually this extra computational cost is small relative to the cost of computing the M Gaussians at time $t+1$. Therefore, the computational cost of the IMM algorithm is only slightly higher than the GPBl algorithm, since both algorithms generate only M Gaussians at every propagation step, and it is significantly lower than GPB2. In practice, it seems that IMM often performs significantly better than GPBl and almost as well as GPB2. Thus,

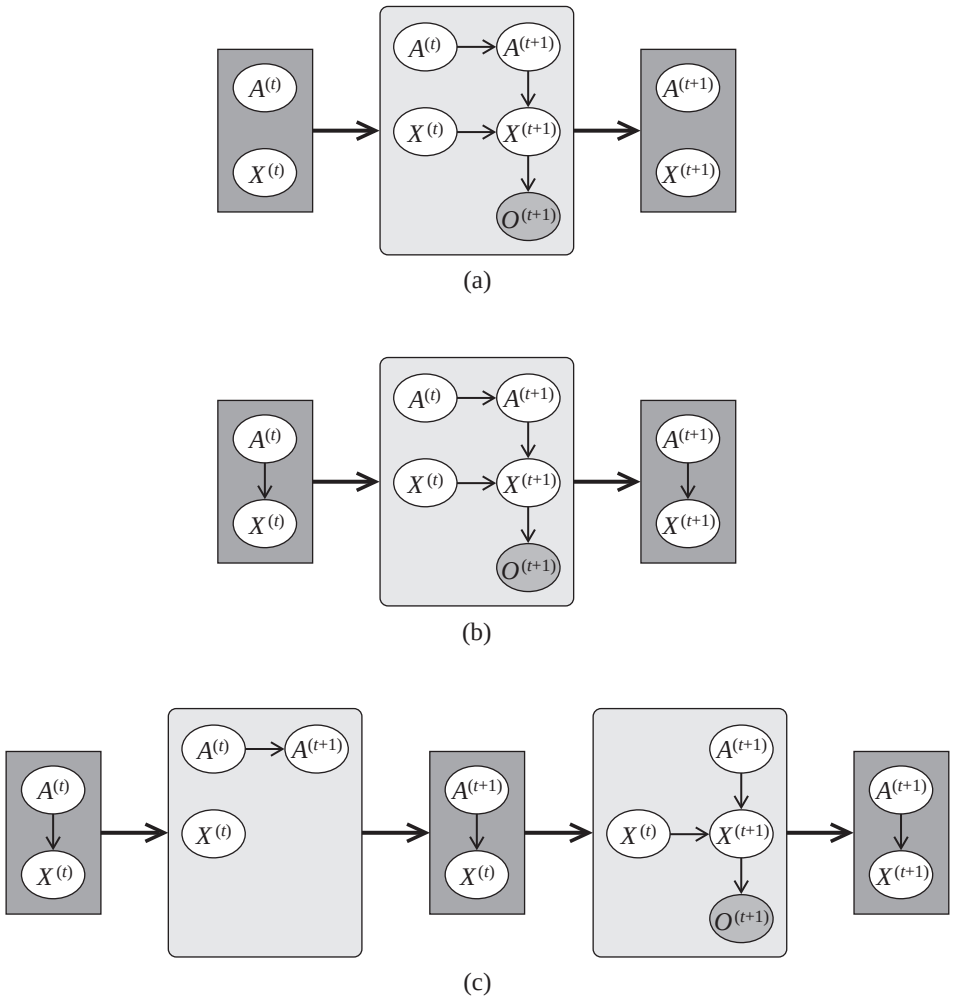


Figure 15.7 Three approaches to collapsing the Gaussian mixture obtained when tracking in a hybrid CLG DBN, viewed as instances of the EP algorithm. The figure illustrates the case where the network contains a single discrete variable A and a single continuous variable X: (a) the GPB1 algorithm; (b) the GPB2 algorithm; (c) the IMM algorithm.

the IMM algorithm appears to be a good compromise between complexity and performance.

We note that all of these clustering methods define the set of mixture components to correspond to assignments of discrete variables in the network. For example, in both GPB1 and IMM, each component in the mixture for $\sigma^{(t)}$ corresponds to an assignment to $\boldsymbol{A}^{(t)}$. In general, this approach may not be optimal, as Gaussians that correspond to different discrete assignments of $\boldsymbol{A}^{(t)}$ may be more similar to each other than Gaussians that correspond to the same assignment. In this case, the collapsed Gaussian would have a variance that is unnecessarily large, leading to a poorer approximation to the belief state. The solution to this problem is to select dynamically a partition of Gaussians in the current belief state, where the Gaussian components in the same partition are collapsed.

Our discussion has focused on cases where the number of discrete states at each time step is tractably small. How do we extend these ideas to more complex systems, where the number of discrete states at each time step is too large to represent explicitly? The EP formulation of these collapsing strategies provides an easy route to generalization. In section 14.3.3, we discussed the application of the EP algorithm to CLG networks, using either a clique tree or a cluster-graph message passing scheme. When faced with a more complex DBN, we can construct a cluster graph or a clique tree for the 2-TBN, where the clusters contain both discrete and continuous variables. These clusters pass messages to each other, using M-projection to the appropriate parametric family chosen for the messages. When a cluster contains one or more discrete variables, the M-projection operation may involve one of the collapsing procedures described. We omit details.

15.5 Summary

In this chapter, we discussed the problem of performing inference in a dynamic Bayesian network. We showed how the most natural inference tasks in this setting map directly onto probabilistic queries in the ground DBN. Thus, at a high level, the inference task here is not different from that of any other Bayesian network: we can simply unroll the network, instantiate our observations, and run inference to compute the answers to our queries. A key challenge lies in the fact that the networks that are produced in this approach can be very (or even unboundedly) large, preventing us from applying many of our exact and approximate inference schemes.

We showed that the tracking problem can naturally be formulated as a single upward pass of clique tree propagation, sending messages from time 0 toward later time slices. The messages naturally represent a *belief state*: our current beliefs about the state of the system. Importantly, this forward-propagation pass can be done in a way that does not require that the clique tree for the entire unrolled network be maintained.

Unfortunately, the entanglement property that arises in all but the most degenerate DBNs typically implies that the belief state has no conditional independence structure, and therefore it does not admit any factorization. This fact generally prevents the use of exact inference, except in DBNs over a fairly small state space.

As for exact inference, approximate inference techniques can be mapped directly to the unrolled DBN. Here also, however, we wish to avoid maintaining the entire DBN in memory during the course of inference. Some algorithms lend themselves more naturally than others to

this "online" message passing. Two methods that have been specifically adapted to this setting are the factored message passing that lies at the heart of the expectation propagation algorithm, and the likelihood-weighting (importance-sampling) algorithm.

The application of the factored message passing is straightforward: We represent the message as a product of factors, possibly with counting number corrections to avoid double-counting; we employ a nested clique tree or cluster graph within each time slice to perform approximate message propagation, mapping a time t approximate belief state to a time $t + 1$ approximate belief state. In the case of filtering, this application is even simpler than the original EP, since no backward message passing is used.

The adaptation of the likelihood-weighting algorithm is somewhat more radical. Here, if we simply propagate particles forward, using the evidence to adjust their weights, the weight of the particles will generally rapidly go to zero, leaving us with a set of particles that has little or no relevance to the true state. From a technical perspective, the variance of this estimator rapidly grows as a function of t. We therefore introduce a resampling step, which allows "good" samples to duplicate while removing poor samples. We discussed several approaches for selecting new samples, including the simple (but very common) bootstrap filter, as well as more complex schemes that use MCMC or other approaches to generate a better proposal distribution for new samples. As in static networks, hybrid schemes that combine sampling with some form of (exact or approximate) global inference can be very useful in DBNs. Indeed, we saw examples of practical applications where this type of approach has been used successfully.

Finally, we discussed the task of inference in continuous or hybrid models. The issues here are generally very similar to the ones we already tackled in the case of static networks. In purely Gaussian networks, a straightforward application of the basic Gaussian message propagation algorithm gives rise to the famous Kalman filter. For nonlinear systems, we can apply the techniques of chapter 14 to derive the extended Kalman filter and the unscented filter. More interesting is the case where we have a hybrid system that contains both discrete and continuous variables. Here, in order to avoid an unbounded blowup in the number of components in the Gaussian mixture representing the belief state, we must collapse multiple Gaussians into one. Various standard approaches for performing this collapsing procedure have been developed in the tracking community; these approaches use a window length in the trajectory to determine which Gaussians to collapse. Interestingly, these approaches can be naturally viewed as variants of the EP algorithm, with different message approximation schemes.

Our presentation in this chapter has focused on inference methods that exploit in some way the specific properties of inference in temporal models. However, all of the inference methods that we have discussed earlier in this book (as well as others) can be adapted to this setting.

If we are willing to unroll the network fully, we can use any inference algorithm that has been developed for static networks. But even in the temporal setting, we can adapt other inference algorithms to the task of inference within a pair of consecutive time slices for the purpose of propagating a belief state. Thus, for example, we can consider variational approximation over a pair of time slices, or MCMC sampling, or various combinations of different algorithms. Many such variants have been proposed; see section 15.6 for references to some of them.

All of our analysis in this chapter has assumed that the structure of the different time slices is the same, so that a fixed approximation structure could be reasonably used for all time slices t. In practice, we might have processes that are not homogeneous, whether because the process itself is nonstationary or because of sparse events that can radically change the momentary

system dynamics. The problem of dynamically adapting the approximation structure to changing circumstances is an exciting direction of research. Also related is the question of dealing with systems whose very structure changes over time, for example, a road where the number of cars can change dynamically.

This last point is related, naturally, to the problem of inference in other types of template-based models, some of which we described in chapter 6 but whose inference properties we did not tackle here. Of course, one option is to construct the full ground network and perform standard inference. However, this approach can be quite costly and even intractable. An important goal is to develop methods that somehow exploit structure in the template-based model to reduce the computational burden. Ideally, we would run our entire inference at the template level, avoiding the step of generating a ground network. This process is called *lifted inference*, in analogy to terms used in first-order logic theorem proving. As a somewhat less lofty goal, we would like to develop algorithms that use properties of the ground network, for example, the fact that it has repeated substructures due to the use of templates. These directions provide an important trajectory for current and future research.

lifted inference

15.6 Relevant Literature

The earliest instances of inference in temporal models were also some of the first applications of dynamic programming for probabilistic inference in graphical models: the forward-backward algorithm of Rabiner and Juang (1986) for hidden Markov models, and the Kalman filtering algorithm.

Kjærulff (1992, 1995a) provided the first algorithm for probabilistic inference in DBNs, based on a clique tree formulation applied to a moving window in the DBNs. Darwiche (2001a) studies the concept of slice-by-slice triangulation in a DBN, and suggests some new elimination strategies. Bilmes and Bartels (2003) extend on this work by providing a triangulation algorithm specifically designed for DBNs; they also show that it can be beneficial to allow the inference data structure to span a window larger than two time slices. Binder, Murphy, and Russell (1997) show how exact clique tree propagation in a DBN can be performed in a space-efficient manner by using a time-space trade-off.

Simple variants of particle-based filtering were first introduced by Handschin and Mayne (1969); Akashi and Kumamoto (1977). The popularity of these methods dates to the mid-1990s, where the resampling step was first introduced to avoid the degeneracy problems inherent to the naive approaches. This idea was introduced independently in several communities under several names, including: dynamic mixture models (West 1993), bootstrap filters (Gordon et al. 1993), survival of the fittest (Kanazawa et al. 1995), condensation (Isard and Blake 1998a), Monte Carlo filters (Kitagawa 1996), and sequential importance sampling (SIS) with resampling (SIR) (Doucet 1998). Kanazawa et al. (1995) propose the use of arc reversal for generating a better sampling distribution. Particle smoothing was first proposed by Isard and Blake (1998b) and Godsill, Doucet, and West (2000).

The success of these methods on a range of practical applications led to the development of multiple improvements, as well as some significant analysis of the theoretical properties of these methods. Doucet, de Freitas, and Gordon (2001) and Ristic, Arulampalam, and Gordon (2004) provide an excellent overview of some of the key developments.

The Viterbi algorithm for finding the MAP assignment in an HMM was proposed by Viterbi (1967); it is the first incarnation of the variable elimination algorithm for MAP algorithm.

The application of collapsed particles to switching linear systems were proposed by Akashi and Kumamoto (1977); Chen and Liu (2000); Doucet et al. (2000). Lerner et al. (2002) propose deterministic particle methods as an alternative to the sampling based approach, and demonstrate significant advantages in cases (such as fault diagnosis) where the distribution is highly peaked, so that sampling would generate the same particle multiple times. Marthi et al. (2002) describe an alternative sampling algorithm based on an MCMC approach with a decaying time window.

BK algorithm

Boyen and Koller (1998b) were the first to study the entanglement phenomenon explicitly and to propose factorization of belief states as an approximation in DBN inference, describing an algorithm that came to be known as the *BK algorithm*. They also provided a theoretical analysis demonstrating that, under certain conditions, the error accumulated over time remains bounded. Boyen and Koller (1998a) extend these ideas to the smoothing task. Murphy and Weiss (2001) suggested an algorithm that uses belief propagation within a time slice rather than a clique tree algorithm, as well as an iterated variant of the BK algorithm. Boyen and Koller (1999) study the properties of different belief state factorizations and offer some guidance on how to select a factorization that leads to a good approximation; Paskin (2003b); Frogner and Pfeffer (2007) suggest concrete heuristics for this adaptive process. Several methods that combine factorization with particle-based methods have also been proposed (Koller and Fratkina 1998; Murphy 1999; Lerner et al. 2000; Montemerlo et al. 2002; Ng et al. 2002).

The Kalman filtering algorithm was proposed by (Kalman 1960; Kalman and Bucy 1961); a simpler version was proposed as early as the nineteenth century (Thiele 1880). Normand and Tritchler (1992) provide a derivation of the Kalman filtering equations from a Bayesian network perspective. Many extensions and improvements have been developed over the years. Bar-Shalom, Li, and Kirubarajan (2001) provide a good overview of these methods, including the extended Kalman filter, and a variety of methods for collapsing the Gaussian mixture in a switching linear-dynamical system. Kim and Nelson (1998) reviews a range of deterministic and MCMC-based methods for these systems.

Lerner et al. (2000) and Lerner (2002) describe an alternative collapsing algorithm that provides more flexibility in defining the Gaussian mixture; they also show how collapsing can be applied in a factored system, where discrete variables are present in multiple clusters. Heskes and Zoeter (2002) apply the EP algorithm to switching linear systems. Zoeter and Heskes (2006) describe the relationship between the GPB algorithms and expectation propagation and provide an experimental comparison of various collapsing methods. The unscented filter, which we described in chapter 14, was first developed in the context of a filtering task by Julier and Uhlmann (1997). It has also been used as a proposal distribution for particle filtering (van der Merwe et al. 2000b,a), producing a filter of higher accuracy and asymptotic correctness guarantees.

When a temporal model is viewed in terms of the ground network it generates, it is amenable to the application of a range of other approximate inference methods. In particular, the global variational methods of section 11.5 have been applied to various classes of sequence-based models (Ghahramani and Jordan 1997; Ghahramani and Hinton 1998).

Temporal models have been applied to very many real-world problems, too numerous to list. Bar-Shalom, Li, and Kirubarajan (2001) describe the key role of these methods to target tracking. Isard and Blake (1998a) first proposed the use of particle filtering for visual tracking tasks; this approach, often called *condensation*, has been highly influential in the computer-

vision community and has led to much follow-on work. The use of probabilistic models has also revolutionized the field of mobile robotics, providing greatly improved solutions to the tasks of navigation and mapping (Fox et al. 1999; Thrun et al. 2000); see Thrun et al. (2005) for a detailed overview of this area. The use of particle filtering, under the name *Monte Carlo localization*, has been particularly influential (Thrun et al. 2000; Montemerlo et al. 2002). However, factored models, both separately and in combination with particle-based methods, have also played a role in these applications (Murphy 1999; Paskin 2003b; Thrun et al. 2004,?), in particular as a representation of complex maps. Dynamic Bayesian models are also playing a role in speech-recognition systems (Zweig and Russell 1998; Bilmes and Bartels 2005) and in fault monitoring and diagnosis of complex systems (Lerner et al. 2002).

Monte Carlo localization

Finally, we also refer the reader to Murphy (2002), who provides an excellent tutorial on inference in DBNs.

15.7 Exercises

Exercise 15.1

Consider clique-tree message passing, described in section 15.2.2, where the messages in the clique tree of figure 15.1 are passed first from the beginning of the clique tree chain toward the end.

a. Show that if sum-product message passing is used, the backward messages (in the "downward" pass) represent $P(o^{((t+1):T)} \mid S^{(t+1)})$.
b. Show that if belief-update message passing is used, the backward messages (in the "downward" pass) represent $P(S^{(t+1)} \mid o^{(1:T)})$.

Exercise 15.2⋆

Consider the smoothing task for HMMs, implemented using the clique tree algorithm described in section 15.2.2. As discussed, the $O(NT)$ space requirement may be computationally prohibitive in certain settings. Let K be the space required to store the evidence at each time slice. Show how, by caching a certain subset of messages, we can trade off time for space, resulting in an algorithm whose time requirements are $O(N^2 T \log T)$ and space requirements are $O(N \log T)$.

Exercise 15.3

Prove proposition 15.1.

Exercise 15.4⋆

a. Prove the entanglement theorem, theorem 15.1.
b. Is there any 2-TBN (not necessarily fully persistent) where $(X \perp Y \mid \boldsymbol{Z})$ holds persistently but $(X \perp Y \mid \emptyset)$ does not? If so, give an example. If not, explain formally why not.

Exercise 15.5

Consider a fully persistent DBN over n state variables $\boldsymbol{X}$. Show that any clique tree over $\boldsymbol{X}^{(t)}, \boldsymbol{X}^{(t+1)}$ that we can construct for performing the belief-state propagation step has induced width at least $n + 1$.

Exercise 15.6⋆

Recall that a hidden Markov model!factorialfactorial HMMfactorial HMM (see figure 6.3) is a DBN over $X_1, \ldots, X_n, O$ such that the only parent of X'_i in the 2-TBN is X_i, and the parents of O' are $X'_1, \ldots, X'_n$. (Typically, some structured model is used to encode this CPD compactly.) Consider the problem of using a structured variational approximation (as in section 11.5) to perform inference over the unrolled network for a fixed number T of time slices.

a. Consider a space of approximate distributions Q composed of disjoint clusters $\{X_i^{(0)}, \ldots, X_i^{(T)}\}$ for $i = 1, \ldots, n$. Show the variational update equations, describe the use of inference to compute the messages, and analyze the computational complexity of the resulting algorithm.

b. Consider a space of approximate distributions Q composed of disjoint clusters $\{X_1^{(1)}, \ldots, X_n^{(t)}\}$ for $t = 1 \ldots, T$. Show the variational update equations, describe the use of inference to compute the messages, and analyze the computational complexity of the resulting algorithm.

c. Discuss the circumstances when you would use one approximation over the other.

Exercise 15.7⋆

particle smoothing

In this question, you will extend the particle filtering algorithm to address the *smoothing* task, that is, computing $P(\boldsymbol{X}^{(t)} \mid \boldsymbol{o}^{(1:T)})$ where $T > t$ is the "end" of the sequence.

a. Prove using formal probabilistic reasoning that

$$P(\boldsymbol{X}^{(t)} \mid \boldsymbol{o}^{(1:T)}) = P(\boldsymbol{X}^{(t)} \mid \boldsymbol{o}^{(1:t)}) \cdot \sum_{\boldsymbol{X}^{(t+1)}} \frac{P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{o}^{(1:T)}) P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{X}^{(t)})}{P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{o}^{(1:t)})}.$$

b. Based on this formula, construct an extension to the particle filtering algorithm that:

- Does a first pass that is identical to particle filtering, but where it keeps the samples $\bar{\boldsymbol{x}}^{(0:t)}[m], w^{(t)}[m]$ generated for each time slice t.
- Has a second phase that updates the weights of the samples at the different time slices based on the formula in part 1, with the goal of getting an approximation to $P(\boldsymbol{X}^{(t)} \mid o^{(1:T)})$.

Write your algorithm using a few lines of pseudo-code. Provide a brief description of how you would use your algorithm to estimate the probability of $P(\boldsymbol{X}^{(t)} = \boldsymbol{x} \mid o^{(1:T)})$ for some assignment $\boldsymbol{x}$.

Exercise 15.8⋆

One of the problems with the particle filtering algorithm is that it uses only the observations obtained so far to select the samples that we continue to propagate. In many cases, the "right" choice of samples at time t is not clear based on the evidence up to time t, but it manifests a small number of time slices into the future. By that time, however, the relevant samples may have been eliminated in favor of ones that appeared (based on the limited evidence available) to be better.

In this problem, your task is to extend the particle filtering algorithm to deal with this problem. More precisely, assume that the relevant evidence usually manifests within k time slices (for some small k). Consider performing the particle-filtering algorithm with a lookahead of k time slices rather than a single time slice. Present the algorithm clearly and mathematically, specifying exactly how the weights are computed and how the next-state samples are generated. Briefly explain why your algorithm is sampling from (roughly) the right distribution (right in the same sense that standard particle filtering is sampling from the right distribution).

For simplicity of notation, assume that the process is structured as a state-observation model.

(Hint: Use your answer to exercise 15.7.)

Exercise 15.9⋆⋆

collapsed particle filtering

In this chapter, we have discussed only the application of particle filtering that samples all of the network variables. In this exercise, you will construct a collapsed-particle-filtering method. Consider a DBN where we have observed variables $\boldsymbol{O}$, and the unobserved variables are divided into two disjoint groups $\boldsymbol{X}$ and $\boldsymbol{Y}$. Assume that, in the 2-TBN, the parents of $\boldsymbol{X}'$ are only variables in $\boldsymbol{X}$, and that we can efficiently perform exact inference on $P(\boldsymbol{Y}' \mid \boldsymbol{X}, \boldsymbol{X}', \boldsymbol{o}')$. Describe an algorithm that uses *collapsed particle filtering* for this type of DBN, where we represent the approximate belief state $\hat{\sigma}^{(t)}$ using a set of weighted collapsed particles, where the variables $\boldsymbol{X}^{(t)}$ are sampled, and for each sample $\boldsymbol{X}^{(t)}[m]$, we have an exact distribution over the variables $\boldsymbol{Y}^{(t)}$. Specifically, describe how each sample $\boldsymbol{x}^{(t+1)}[m]$ is

generated from the time t samples, how to compute the associated distribution over $\mathbf{Y}^{(t)}$, and how to compute the appropriate importance weights. Make sure your derivation is consistent with the analysis used in section 12.4.1.

Exercise 15.10

beam search

In this exercise, we apply the *beam-search* methods described in section 13.7 to the task of finding high-probability assignments in an HMM. Assume that our hidden state in the HMM is denoted $X^{(t)}$ and the observations are denoted $O^{(t)}$. Our goal is to find $x^{*(1:T)} = \arg\max_{x^{(1:T)}} P(X^{(1:T)} \mid O^{(1:t)})$. Importantly, however, we want to perform the beam search in an online fashion, adapting our set of candidates as new observations arrive.

a. Describe how beam search can be applied in the context of an HMM. Specify the search procedure and suggest a number of pruning strategies.

b. Now, assume we have an additional set of variables $Y^{(t)}$ that are part of our model and whose value we do not care about. That is, our goal has not changed. Assume that our 2-TBN model does not contain an arc $Y \rightarrow X'$. Describe how beam search can be applied in this setting.

Exercise 15.11

Construct an algorithm that performs tracking in linear-Gaussian dynamical systems, maintaining the belief state in terms of the canonical form described in section 14.2.1.2.

Exercise 15.12⋆

Consider a nonlinear 2-TBN where we have a set of observation variables $O_1, \ldots, O_k$, where each O'_i is a leaf in the 2-TBN and has the parents $\boldsymbol{U}_i \in \boldsymbol{X}'$. Show how we can use the methods of exercise 14.9 to perform the step of conditioning on the observation $\boldsymbol{O}' = \boldsymbol{o}'$, without forming an entire joint distribution over $\boldsymbol{X}' \cup O$. Your method should perform the numerical integration over a space with as small a dimension as possible.

Exercise 15.13⋆

a. Write down the probabilistic model for the Gaussian SLAM problem with K landmarks.

b. Derive the equations for a collapsed bootstrap-sampling particle-filtering algorithm in FastSLAM. Show how the samples are generated, how the importance weights are computed, and how the posterior is maintained.

c. Derive the equations for the posterior collapsed-particle-filtering algorithm, where $\boldsymbol{x}^{(t+1)}[m]$ is generated from $P(\boldsymbol{X}^{(t+1)} \mid \boldsymbol{x}^{(t)}[m], \boldsymbol{y}^{(t+1)})$. Show how the samples are generated, how the importance weights are computed, and how the posterior is maintained.

d. Now, consider a different approach of applied collapsed-particle filtering to this problem. Here, we select the landmark positions $\boldsymbol{L} = \{L_1, \ldots, L_k\}$ as our set of sampled variables, and for each particle $\boldsymbol{l}[m]$, we maintain a distribution over the robot's state at time t. Without describing the details of this algorithm, explain qualitatively what will happen to the particles and their weights eventually (as T grows). Are there conditions under which this algorithm will converge to the correct map?

Part III

Learning

16 *Learning Graphical Models: Overview*

16.1 Motivation

In most of our discussions so far, our starting point has been a given graphical model. For example, in our discussions of conditional independencies and of inference, we assumed that the model — structure as well as parameters — was part of the input.

There are two approaches to the task of acquiring a model. The first, as we discussed in box 3.C, is to construct the network by hand, typically with the help of an expert. However, as we saw, knowledge acquisition from experts is a nontrivial task. The construction of even a modestly sized network requires a skilled knowledge engineer who spends several hours (at least) with one or more domain experts. Larger networks can require weeks or even months of work. This process also generally involves significant testing of the model by evaluating results of some "typical" queries in order to see whether the model returns plausible answers.

Such "manual" network construction is problematic for several reasons. **In some domains, the amount of knowledge required is just too large or the expert's time is too valuable. In others, there are simply no experts who have sufficient understanding of the domain. In many domains, the properties of the distribution change from one application site to another or over time, and we cannot expect an expert to sit and redesign the network every few weeks.** In many settings, however, we may have access to a set of examples generated from the distribution we wish to model. In fact, in the Information Age, it is often easier to obtain even large amounts of data in electronic form than it is to obtain human expertise. For example, in the setting of medical diagnosis (such as box 3.D), we may have access to a large collection of patient records, each listing various attributes such as the patient's history (age, sex, smoking, previous medical complications, and so on), reported symptoms, results of various tests, the physician's initial diagnosis and prescribed treatment, and the treatment's outcome. We may hope to use these data to learn a model of the distribution of patients in our population. In the case of pedigree analysis (box 3.B), we may have some set of family trees where a particular disease (for example, breast cancer) occurs frequently. We can use these family trees to learn the parameters of the genetics of the disease: the transmission model, which describes how often a disease genotype is passed from the parents to a child, and the penetrance model, which defines the probability of different phenotypes given the genotype. In an image segmentation application (box 4.B), we might have a set of images segmented by a person, and we might wish to learn the parameters of the MRF that define the characteristics of different regions, or those that define how strongly we believe that two neighboring pixels should be in the same segment.

It seems clear that such instances can be of use in constructing a good model for the underlying distribution, either in isolation or in conjunction with some prior knowledge acquired from a human. This task of constructing a model from a set of instances is generally called *model learning*. In this part of the book, we focus on methods for addressing different variants of this task. In the remainder of this chapter, we describe some of these variants and some of the issues that they raise.

To make this discussion more concrete, let us assume that the domain is governed by some underlying distribution P^*, which is induced by some (directed or undirected) network model $\mathcal{M}^* = (\mathcal{K}^*, \boldsymbol{\theta}^*)$. We are given a data set $\mathcal{D} = \{\boldsymbol{d}[1], \ldots, \boldsymbol{d}[M]\}$ of M samples from P^*. The standard assumption, whose statistical implications were briefly discussed in appendix A.2, is that the data instances are sampled independently from P^*; as we discussed, such data instances are called *independent and identically distributed (IID)*. We are also given some family of models, and our task is to learn some model $\tilde{\mathcal{M}}$ in this family that defines a distribution $P_{\tilde{\mathcal{M}}}$ (or $\tilde{P}$ when $\tilde{\mathcal{M}}$ is clear from the context). We may want to learn only model parameters for a fixed structure, or some or all of the structure of the model. In some cases, we might wish to present a spectrum of different hypotheses, and so we might return not a single model but rather a probability distribution over models, or perhaps some estimate of our confidence in the model learned.

IID samples

We first describe the set of goals that we might have when learning a model and the different evaluation metrics to which they give rise. We then discuss how learning can be viewed as an optimization problem and the issues raised by the design of that problem. Finally, we provide a detailed taxonomy of the different types of learning tasks and discuss some of their computational ramifications.

16.2 Goals of Learning

To evaluate the merits of different learning methods, it is important to consider our goal in learning a probabilistic model from data. A priori, it is not clear why the goal of the learning is important. After all, our ideal solution is to return a model $\tilde{\mathcal{M}}$ that precisely captures the distribution P^* from which our data were sampled. Unfortunately, this goal is not generally achievable, because of computational reasons and (more importantly) because a limited data set provides only a rough approximation of the true underlying distribution. **In practice, the amount of data we have is rarely sufficient to obtain an accurate representation of a high-dimensional distribution involving many variables. Thus, we have to select $\tilde{\mathcal{M}}$ so as to construct the "best" approximation to $\mathcal{M}^*$. The notion of "best" depends on our goals. Different models will generally embody different trade-offs.** One approximate model may be better according to one performance metric but worse according to another. Therefore, to guide our development of learning algorithms, we must define the goals of our learning task and the corresponding metrics by which different results will be evaluated.

16.2.1 Density Estimation

The most common reason for learning a network model is to use it for some inference task that we wish to perform. Most obviously, as we have discussed throughout most of this book so far, a graphical model can be used to answer a range of probabilistic inference queries. In this

density estimation

setting, we can formulate our learning goal as one of *density estimation*: constructing a model $\tilde{\mathcal{M}}$ such that $\tilde{P}$ is "close" to the generating distribution P^*.

How do we evaluate the quality of an approximation $\tilde{\mathcal{M}}$? One commonly used option is to use the relative entropy distance measure defined in definition A.5:

$$\boldsymbol{D}(P^* \| \tilde{P}) = \boldsymbol{E}_{\xi \sim P^*} \left[\log \left(\frac{P^*(\xi)}{\tilde{P}(\xi)} \right) \right].$$

Recall that this measure is zero when $\tilde{P} = P^*$ and positive otherwise. Intuitively, it measures the extent of the compression loss (in bits) of using $\tilde{P}$ rather than P^*.

To evaluate this metric, we need to know P^*. In some cases, we are evaluating a learning algorithm on synthetically generated data, and so P^* may be known. In real-world applications, however, P^* is not known. (If it were, we would not need to learn a model for it from a data set.) However, we can simplify this metric to obtain one that is easier to evaluate:

Proposition 16.1 *For any distributions P, P' over $\mathcal{X}$:*

$$\boldsymbol{D}(P \| P') = -\boldsymbol{H}_P(\mathcal{X}) - \boldsymbol{E}_{\xi \sim P}[\log P'(\xi)].$$

Proof

$$\begin{aligned} \boldsymbol{D}(P \| P') &= \boldsymbol{E}_{\xi \sim P} \left[\log \left(\frac{P(\xi)}{P'(\xi)} \right) \right] \\ &= \boldsymbol{E}_{\xi \sim P}[\log P(\xi) - \log P'(\xi)] \\ &= \boldsymbol{E}_{\xi \sim P}[\log P(\xi)] - \boldsymbol{E}_{\xi \sim P}[\log P'(\xi)] \\ &= -\boldsymbol{H}_P(\mathcal{X}) - \boldsymbol{E}_{\xi \sim P}[\log P'(\xi)]. \end{aligned}$$

■

Applying this derivation to $P^*, \tilde{P}$, we see that the first of these two terms is the negative entropy of P^*; because it does not depend on $\tilde{P}$, it does not affect the comparison between different approximate models. We can therefore focus our evaluation metric on the second term $\boldsymbol{E}_{\xi \sim P^*} \left[\log \tilde{P}(\xi) \right]$ and prefer models that make this term as large as possible. This term is called an *expected log-likelihood*. It encodes our preference for models that assign high probability to instances sampled from P^*. Intuitively, the higher the probability that $\tilde{\mathcal{M}}$ gives to points sampled from the true distribution, the more reflective it is of this distribution. We note that, in moving from the relative entropy to the expected log-likelihood, we have lost our baseline $\boldsymbol{E}_{P^*}[\log P^*]$, an inevitable loss since we do not know P^*. As a consequence, although we can use the log-likelihood as a metric for comparing one learned model to another, we cannot evaluate a particular $\tilde{\mathcal{M}}$ in how close it is to the unknown optimum.

expected log-likelihood

likelihood

log-likelihood

More generally, in our discussion of learning we will be interested in the *likelihood* of the data, given a model $\mathcal{M}$, which is $P(\mathcal{D} : \mathcal{M})$, or for convenience using the *log-likelihood* $\ell(\mathcal{D} : \mathcal{M}) = \log P(\mathcal{D} : \mathcal{M})$.

log-loss

loss function

It is also customary to consider the negated form of the log-likelihood, called the *log-loss*. The log-loss reflects our cost (in bits) per instance of using the model $\tilde{P}$. The log-loss is our first example of a *loss function*, a key component in the statistical machine-learning paradigm. A loss function $loss(\xi : \mathcal{M})$ measures the loss that a model $\mathcal{M}$ makes on a particular instance

ξ. When instances are sampled from some distribution P^*, our goal is to find a model that minimizes the *expected loss*, or the *risk*:

risk

$$\mathbf{E}_{\xi \sim P^*}[loss(\xi \ : \ \mathcal{M})].$$

In general, of course, P^* is unknown. However, we can approximate the expectation using an empirical average and estimate the risk relative to P^* with an *empirical risk* averaged over a set $\mathcal{D}$ of instances sampled from P^*:

empirical risk

$$\mathbf{E}_{\mathcal{D}}[loss(\xi \ : \ \mathcal{M})] = \frac{1}{|\mathcal{D}|} \sum_{\xi \in \mathcal{D}} loss(\xi \ : \ \mathcal{M}). \tag{16.1}$$

In the case of the log-loss, this expression has a very intuitive interpretation. Consider a data set $\mathcal{D} = \{\xi[1], \ldots, \xi[M]\}$ composed of IID instances. The probability that $\mathcal{M}$ ascribes to $\mathcal{D}$ is

$$P(\mathcal{D} \ : \ \mathcal{M}) = \prod_{m=1}^{M} P(\xi[m] \ : \ \mathcal{M}).$$

Taking the logarithm, we obtain

$$\log P(\mathcal{D} \ : \ \mathcal{M}) = \sum_{m=1}^{M} \log P(\xi[m] \ : \ \mathcal{M}),$$

which is precisely the negative of the empirical log-loss that appears inside the summation of equation (16.1).

The risk can be used both as a metric for evaluating the quality of a particular model and as a factor for selecting a model among a given class, given a training set $\mathcal{D}$. We return to these ideas in section 16.3.1 and box 16.A.

16.2.2 Specific Prediction Tasks

In the preceding discussion, we assumed that our goal was to use the learning model to perform probabilistic inference. With that assumption, we jumped to the conclusion that we wish to fit the overall distribution P^* as well as possible. However, that objective measures only our ability to evaluate the overall probability of a full instance ξ. In reality, the model can be used for answering a whole range of queries of the form $P(\boldsymbol{Y} \mid \boldsymbol{X})$. In general, we can devise a test suite of queries for our learned model, which allows us to evaluate its performance on a range of queries. Most attention, however, has been paid to the special case where we have a particular set of variables $\boldsymbol{Y}$ that we are interested in predicting, given a certain set of variables $\boldsymbol{X}$.

classification task

Most simply, we may want to solve a simple *classification task*, the goal of a large fraction of the work in machine learning. For example, consider the task of document classification, where we have a set $\boldsymbol{X}$ of words and other features characterizing the document, and a variable Y that labels the document topic. In image segmentation, we are interested in predicting the class labels for all of the pixels in the image ($\boldsymbol{Y}$), *given* the image features ($\boldsymbol{X}$). There are many other such examples.

A model trained for a prediction task should be able to produce for any instance characterized by $\boldsymbol{x}$, the probability distribution $\tilde{P}(Y \mid \boldsymbol{x})$. We might also wish to select the MAP assignment of this conditional distribution to produce a specific prediction:

$$h_{\tilde{P}}(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \tilde{P}(\boldsymbol{y} \mid \boldsymbol{x}).$$

classification error

What loss function do we want to use for evaluating a model designed for a prediction task? We can, for example, use the *classification error*, also called the *0/1 loss*:

0/1 loss

$$\boldsymbol{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\tilde{P}}[\boldsymbol{1}\{h_{\tilde{P}}(\boldsymbol{x}) \neq \boldsymbol{y}\}], \tag{16.2}$$

which is simply the probability, over all $(\boldsymbol{x}, \boldsymbol{y})$ pairs sampled from $\tilde{P}$, that our classifier selects the wrong label. While this metric is suitable for labeling a single variable, it is not well suited to situations, such as image segmentation, where we simultaneously provide labels to a large number of variables. In this case, we do not want to penalize an entire prediction with an error of 1 if we make a mistake on only a few of the target variables. Thus, in this case, we might consider performance metrics such as the *Hamming loss*, which, instead of using the indicator function $\boldsymbol{1}\{h_{\tilde{P}}(\boldsymbol{x}) \neq \boldsymbol{y}\}$, counts the number of variables $\boldsymbol{Y}$ in which $h_{\tilde{P}}(\boldsymbol{x})$ differs from the ground truth $\boldsymbol{y}$.

Hamming loss

We might also wish to take into account the confidence of the prediction. One such criterion is the *conditional likelihood* or its logarithm (sometimes called the *conditional log-likelihood*):

conditional likelihood

$$\boldsymbol{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P^*}\left[\log \tilde{P}(\boldsymbol{y} \mid \boldsymbol{x})\right]. \tag{16.3}$$

Like the log-likelihood criterion, this metric evaluates the extent to which our learned model is able to predict data generated from the distribution. However, it requires the model to predict only $\boldsymbol{Y}$ given $\boldsymbol{X}$, and not the distribution of the $\boldsymbol{X}$ variables. As before, we can negate this expression to define a loss function and compute an empirical estimate by taking the average relative to a data set $\mathcal{D}$.

If we determine, in advance, that the model will be used only to perform a particular task, we may want to train the model to make trade-offs that make it more suited to that task. In particular, if the model is never evaluated on predictions of the variables $\boldsymbol{X}$, we may want to design our training regime to optimize the quality of its answers for $\boldsymbol{Y}$. We return to this issue in section 16.3.2.

16.2.3 Knowledge Discovery

knowledge discovery

Finally, a very different motivation for learning a model for a distribution P^* is as a tool for *discovering knowledge* about P^*. We may hope that an examination of the learned model can reveal some important properties of the domain: what are the direct and indirect dependencies, what characterizes the nature of the dependencies (for example, positive or negative correlation), and so forth. For example, in the genetic inheritance domain, it may be of great interest to discover the parameter governing the inheritance of a certain property, since this parameter can provide significant biological insight regarding the inheritance mechanism for the allele(s) governing the disease. In a medical diagnosis domain, we may want to learn the structure of the model to discover which predisposing factors lead to certain diseases and which symptoms

are associated with different diseases. Of course, simpler statistical methods can be used to explore the data, for example, by highlighting the most significant correlations between pairs of variables. However, a learned network model can provide parameters that have direct causal interpretation and can also reveal much finer structure, for example, by distinguishing between direct and indirect dependencies, both of which lead to correlations in the resulting distribution.

The knowledge discovery task calls for a very different evaluation criterion and a different set of compromises from a prediction task. In this setting, we really do care about reconstructing the correct model $\mathcal{M}^*$, rather than some other model $\tilde{\mathcal{M}}$ that induces a distribution similar to $\mathcal{M}^*$. Thus, in contrast to density estimation, where our metric was on the distribution defined by the model (for example, $\boldsymbol{D}(P^* \| \tilde{P})$), here our measure of success is in terms of the model, that is, differences between $\mathcal{M}^*$ and $\tilde{\mathcal{M}}$. Unfortunately, this goal is often not achievable, for several reasons.

identifiability

First, even with large amounts of data, the true model may not be *identifiable*. Consider, for example, the task of recovering aspects of the correct network structure $\mathcal{K}^*$. As one obvious difficulty, recall that a given Bayesian network structure often has several I-equivalent structures. If such is the case for our target distribution $\mathcal{K}^*$, the best we can hope for is to recover an I-equivalent structure. The problems are significantly greater when the data are limited. Here, for example, if X and Y are directly related in $\mathcal{K}^*$ but the parameters relating them induce only a weak relationship, it may be very difficult to detect the correlation in the data and distinguish it from a random fluctuations. This limitation is less of a problem for a density estimation task, where ignoring such weak correlations often has very few repercussions on the quality of our learned density; however, if our task focuses on correct reconstruction of structure, examples such as this reduce our accuracy. Conversely, when the number of variables is large relative to the amount of the training data, there may well be pairs of variables that appear strongly correlated simply by chance. Thus, we are also likely, in such settings, to infer the presence of edges that do not exist in the underlying model. Similar issues arise when attempting to infer other aspects of the model.

The relatively high probability of making model identification errors can be significant if the goal is to discover the correct structure of the underlying distribution. For example, if our goal is to infer which genes regulate which other genes (as in box 21.D), and if we plan to use the results of our analysis for a set of (expensive and time-consuming) wet-lab experiments, we may want to have some confidence in the inferred relationship.  **Thus, in a knowledge discovery application, it is far more critical to assess the confidence in a prediction, taking into account the extent to which it can be identified given the available data and the number of hypotheses that would give rise to similar observed behavior.** We return to these issues more concretely later on in the book (see, in particular, section 18.5).

16.3 Learning as Optimization

The previous section discussed different ways in which we can evaluate our learned model. In many of the cases, we defined a numerical criterion — a loss function — that we would like to optimize. This perspective suggests that the learning task should be viewed as an optimization problem: we have a *hypothesis space*, that is, a set of candidate models, and an *objective function*, a criterion for quantifying our preference for different models. Thus, our learning task can be

hypothesis space

objective function

formulated as one of finding a high-scoring model within our model class. The view of learning as optimization is currently the predominant approach to learning (not only of probabilistic models).

In this section, we discuss different choices of objective functions and their ramification on the results of our learning procedure. This discussion raises important points that will accompany us throughout this part of the book. We note that the formal foundations for this discussion will be established in later chapters, but the discussion is of fundamental importance to our entire discussion of learning, and therefore we introduce these concepts here.

16.3.1 Empirical Risk and Overfitting

Consider the task of constructing a model $\mathcal{M}$ that optimizes the expectation of a particular loss function $\mathbf{E}_{\xi \sim P^*}[loss(\xi : \mathcal{M})]$. Of course, we generally do not know P^*, but, as we have discussed, we can use a data set $\mathcal{D}$ sampled from P^* to produce an empirical estimate for this expectation. More formally, we can use the data $\mathcal{D}$ to define an *empirical distribution* $\hat{P}_{\mathcal{D}}$, as follows:

empirical distribution

$$\hat{P}_{\mathcal{D}}(A) = \frac{1}{M} \sum_m \mathbf{1}\{\xi[m] \in A\}. \tag{16.4}$$

That is, the probability of the event A is simply the fraction of training examples that satisfy A. It is clear that $\hat{P}_{\mathcal{D}}(A)$ is a probability distribution. Moreover, as the number of training examples grows, the empirical distribution approaches the true distribution.

Theorem 16.1 *Let $\xi[1], \xi[2], \ldots$ be a sequence of IID samples from $P^*(\mathcal{X})$, and let $\mathcal{D}_M = \langle \xi[1], \ldots, \xi[M] \rangle$, then*

$$\lim_{M \to \infty} \hat{P}_{\mathcal{D}_M}(A) = P^*(A)$$

almost surely.

Thus, for a sufficiently large training set, $\hat{P}_{\mathcal{D}}$ will be quite close to the original distribution P^* with high probability (one that converges to 1 as $M \to \infty$). Since we do not have direct access to P^*, we can use $\hat{P}_{\mathcal{D}}$ as the best proxy and try to minimize our loss function relative to $\hat{P}_{\mathcal{D}}$. Unfortunately, a naive application of this optimization objective can easily lead to very poor results.

Consider, for example, the use of the empirical log-loss (or log-likelihood) as the objective. It is not difficult to show (see section 17.1) that the distribution that maximizes the likelihood of a data set $\mathcal{D}$ is the empirical distribution $\hat{P}_{\mathcal{D}}$ itself. Now, assume that we have a distribution over a probability space defined by 100 binary random variables, for a total of 2^{100} possible joint assignments. If our data set $\mathcal{D}$ contains 1,000 instances (most likely distinct from each other), the empirical distribution will give probability 0.001 to each of the assignments that appear in $\mathcal{D}$, and probability 0 to all $2^{100} - 1,000$ other assignments. While this example is obviously extreme, the phenomenon is quite general. For example, assume that $\mathcal{M}^*$ is a Bayesian network where some variables, such as *Fever*, have a large number of parents $X_1, \ldots, X_k$. In a table-CPD, the number of parameters grows exponentially with the number of parents k. For large k, we are highly unlikely to encounter, in $\mathcal{D}$, instances that are relevant to all possible parent instantiations, that is, all possible combinations of diseases $X_1, \ldots, X_k$. If we do not have

enough data, many of the cases arising in our CPD will have very little (or no) relevant training data, leading to very poor estimates of the conditional probability of *Fever* in this context. In general, as we will see in later chapters, the amount of data required to estimate parameters reliably grows linearly with the number of parameters, so that the amount of data required can grow exponentially with the network connectivity.

As we see in this example, there is a significant problem with using the empirical risk (the loss on our training data) as a surrogate for our true risk. In particular, this type of objective tends to *overfit* the learned model to the training data. However, our goal is to answer queries about examples that were not in our training set. Thus, for example, in our medical diagnosis example, the patients to which the learned network will be applied are new patients, not the ones on whose data the network was trained. In our image-segmentation example, the model will be applied to new (unsegmented) images, not the (segmented) images on which the model was trained. Thus, it is critical that the network *generalize* to perform well on unseen data.

overfitting

generalization

The need to generalize to unseen instances and the risk of overfitting to the training set raise an important trade-off that will accompany us throughout our discussion. On one hand, if our hypothesis space is very limited, it might not be able to represent our true target distribution P^*. Thus, even with unlimited data, we may be unable to capture P^*, and thereby remain with a suboptimal model. This type of limitation in a hypothesis space introduces inherent error in the result of the learning procedure, which is called *bias*, since the learning procedure is limited in how close it can approximate the target distribution. Conversely, if we select a hypothesis space that is highly expressive, we are more likely to be able to represent correctly the target distribution P^*. However, given a small data set, we may not have the ability to select the "right" model among the large number of models in the hypothesis space, many of which may provide equal or perhaps even better loss on our limited (and thereby unrepresentative) training set $\mathcal{D}$. Intuitively, when we have a rich hypothesis space and limited number of samples, small random fluctuations in the choice of $\mathcal{D}$ can radically change the properties of the selected model, often resulting in models that have little relationship to P^*. As a result, the learning procedure will suffer from a high *variance* — running it on multiple data sets from the same P^* will lead to highly variable results. Conversely, if we have a more limited hypothesis space, we are less likely to find, by chance, a model that provides a good fit to $\mathcal{D}$. Thus, a high-scoring model within our limited hypothesis space is more likely to be a good fit to P^*, and thereby is more likely to generalize to unseen data.

bias-variance trade-off

This *bias-variance trade-off* underlies many of our design choices in learning. When selecting a hypothesis space of different models, we must take care not to allow too rich a class of possible models. Indeed, with limited data, the error introduced by variance may be larger than the potential error introduced by bias, and we may choose to restrict our learning to models that are too simple to correctly encode P^*. Although the learned model is guaranteed to be incorrect, our ability to estimate its parameters more reliably may well compensate for the error arising from incorrect structural assumptions. Moreover, when learning structure, although we will not correctly learn all of the edges, this restriction may allow us to more reliably learn the most important edges. In other words, restricting the space of possible models leads us to select models $\tilde{\mathcal{M}}$ whose performance on the training objective is poorer, but whose distance to P^* is better.

Restricting our model class is one way to reduce overfitting. In effect, it imposes a hard constraint that prevents us from selecting a model that precisely captures the training data. A

second, more refined approach is to change our training objective so as to incorporate a soft preference for simpler models. Thus, our learning objective will usually incorporate competing components: some components will tend to move us toward models that fit well with our observed data; others will provide *regularization* that prevents us from taking the specifics of the data to extremes. **In many cases, we adopt a combination of the two approaches, utilizing both a hard constraint over the model class and an optimization objective that leads us away from overfitting.**

regularization

The preceding discussion described the phenomenon of overfitting and the importance of ensuring that our learned model generalizes to unseen data. However, we did not discuss how to tell whether our model generalizes, and how to design our hypothesis space and/or objective function so as to reduce this risk. Box 16.A discusses some of the basic experimental protocols that one uses in the design and evaluation of machine learning procedures. Box 16.B discusses a basic theoretical framework that one can use to try and answer questions regarding the appropriate complexity of our model class.

Box 16.A — Skill: Design and Evaluation of Learning Procedures. *A basic question in learning is to evaluate the performance of our learning procedure. We might ask this question in a relative sense, to compare two or more alternatives (for example, different hypothesis spaces, or different training objectives), or in an absolute sense, when we want to test whether the model we have learned "captures" the distribution. Both questions are nontrivial, and there is a large literature on how to address them. We briefly summarize some of the main ideas here.*

In both cases, we would ideally like to compare the learned model to the real underlying distribution that generated the data. This is indeed the strategy we use for evaluating performance when learning from synthetic data sets where we know (by design) the generating distribution (see, for example, box 17.C). Unfortunately, this strategy is infeasible when we learn from real-life data sets where we do not have access to the true generating distribution. And while synthetic studies can help us understand the properties of learning procedures, they are limited in that they are often not representative of the properties of the actual data we are interested in.

Evaluating Generalization Performance *We start with the first question of evaluating the performance of a given model, or a set of models, on unseen data. One approach is to use* holdout testing. *In this approach, we divide our data set into two disjoint sets: the* training set $\mathcal{D}_{train}$ *and* test set $\mathcal{D}_{test}$. *To avoid artifacts, we usually use a randomized procedure to decide on this partition. We then learn a model using $\mathcal{D}_{train}$ (with some appropriate objective function), and we measure our performance (using some appropriate loss function) on $\mathcal{D}_{test}$. Because $\mathcal{D}_{test}$ is also sampled from P^*, it provides us with an empirical estimate of the risk. Importantly, however, because $\mathcal{D}_{test}$ is disjoint from $\mathcal{D}_{train}$, we are measuring our loss using instances that were unseen during the training, and not on ones for which we optimized our performance. Thus, this approach provides us with an unbiased estimate of the performance on new instances.*

holdout testing

training set

test set

Holdout testing can be used to compare the performance of different learning procedures. It can also be used to obtain insight into the performance of a single learning procedure. In particular, we can compare the performance of the procedure (say the empirical log-loss per instance, or the classification error) on the training set and on the held-out test set. Naturally, the training set performance will be better, but if the difference is very large, we are probably overfitting to the

training data and may want to consider a less expressive model class, or some other method for discouraging overfitting.

Holdout testing poses a dilemma. To get better estimates of our performance, we want to increase the size of the test set. Such an increase, however, decreases the size of the training set, which results in degradation of quality of the learned model. When we have ample training data, we can find reasonable compromises between these two considerations. When we have few training samples, there is no good compromise, since decreasing either the training or the test set has large ramifications either on the quality of the learned model or the ability to evaluate it.

An alternative solution is to attempt to use available data for both training and testing. Of course, we cannot test on our training data; the trick is to combine our estimates of performance from repeated rounds of holdout testing. That is, in each iteration we train on some subset of the instances and test on the remaining ones. If we perform multiple iterations, we can use a relatively small test set in each iteration, and pool the performance counts from all of them to estimate performance. The question is how to allocate the training and test data sets. A commonly used procedure is k-fold cross-validation, *where we use each instance once for testing. This is done by partitioning the original data into k equally sized sets, and then in each iteration holding as test data one partition and training from all the remaining instances; see algorithm 16.A.1. An extreme case of cross-validation is* leave one out cross-validation, *where we set $k = M$, that is, in each iteration we remove one instance and use it as a testing case. Both cross-validation schemes allow us to estimate not only the average performance of our learning algorithm, but also the extent to which this performance varies across the different folds.*

cross-validation

Both holdout testing and cross-validation are primarily used as methods for evaluating a learning procedure. In particular, a cross-validation procedure constructs k different models, one for each partition of the training set into training/test folds, and therefore does not result in even a single model that we can subsequently use. If we want to write a paper on a new learning procedure, the results of cross-validation provide a good regime for evaluating our procedure and comparing it to others. If we actually want to end up with a real model that we can use in a given domain, we would probably use cross-validation or holdout testing to select an algorithm and ensure that it is working satisfactorily, but then train our model on the entire data set $\mathcal{D}$, thereby making use of the maximum amount of available data to learn a single model.

Selecting a Learning Procedure *One common use for these evaluation procedures is as a mechanism for selecting a learning algorithm that is likely to perform well on a particular application. That is, we often want to choose among a (possibly large) set of different options for our learning procedure: different learning algorithms, or different algorithmic parameters for the same algorithm (for example, different constraints on the complexity of the learned network structures). At first glance, it is straightforward to use holdout testing or cross-validation for this purpose: we take each option* LearnProc_j, *evaluate its performance, and select the algorithm whose estimated loss is smallest. While this use is legitimate, it is also tempting to use the performance estimate that we obtained using this procedure as a measure for how well our algorithm will generalize to unseen data. This use is likely to lead to misleading and overly optimistic estimates of performance, since we have selected our particular learning procedure to optimize for this particular performance metric. Thus, if we use cross-validation or a holdout set to* select *a learning procedure, and we want to have an unbiased estimate of how well our selected procedure will perform on unseen data, we must hold back a completely separate test set that is never used in selecting any aspect of the*

Algorithm 16.A.1 — Algorithms for holdout and cross-validation tests.

```
Procedure Evaluate (
    M,    // parameters to evaluate
    D     // test data set
)
1   loss ← 0
2   for m = 1, ..., M
3     loss ← loss + loss(ξ[m] : M)
4   return loss/M

Procedure Train-And-Test (
    LearnProc,    // Learning procedure
    D_train,      // Training data
    D_test,       // Test data
)
1   M ← LearnProc(D_train)
2   return Evaluate(M, D_test)

Procedure Holdout-Test (
    LearnProc,    // Learning procedure
    D,            // Data Set
    p_test        // Fraction of data for testing
)
1   Randomly reshuffle instances in D
2   M_train ← round(M · (1 − p_test))
3   D_train ← {ξ[1], ..., ξ[M_train]}
4   D_test ← {ξ[M_train + 1], ..., ξ[M]}
5   return Train-And-Test(LearnProc, D_train, D_test)

Procedure Cross-Validation (
    LearnProc,    // Learning procedure
    D,            // Data Set
    K,            // number of cross-validation folds
)
1   Randomly reshuffle instances in D
2   Partition D into K disjoint data sets D_1, ..., D_K
3   loss ← 0
4   for k = 1, ..., K
5     D_−k ← D − D_k
6     loss ← loss + Train-And-Test(LearnProc, D_−k, D_k)
7   return loss/K
```

validation set

model, on which our model's final performance will be evaluated. In this setting, we might have: a training set, using which we learn the model; a validation set*, which we use to evaluate different variants of our learning procedure and select among them; and a separate test set, on which our final performance is actually evaluated. This approach, of course, only exacerbates the problem of fragmenting our training data, and so one can develop nested cross-validation schemes that achieve the same goal.*

goodness of fit

Goodness of Fit *Cross-validation and holdout tests allow us to evaluate performance of different learning procedures on unseen data. However, without a "gold standard" for comparison, they do not allow us to evaluate whether our learned model really captures everything there is to capture about the distribution. This question is inherently harder to answer. In statistics, methods for answering such questions fall under the category of* goodness of fit *tests. The general idea is the following. After learning the parameters, we have a hypothesis about a distribution that generated the data. Now we can ask whether the data behave as though they were sampled from this distribution. To do this, we compare properties of the training data set to properties of* simulated *data sets of the same size that we generate according to the learned distribution. If the training data behave in a manner that deviates significantly from what we observed in the majority of the simulations, we have reason to believe that the data were not generated from the learned distribution.*

More precisely, we consider some property f of data sets, and evaluate $f(\mathcal{D}_{train})$ for the training set. We then generate new data sets $\mathcal{D}$ from our learned model $\mathcal{M}$ and evaluate $f(\mathcal{D})$ for these randomly generated data sets. If $f(\mathcal{D}_{train})$ deviates significantly from the distribution of the values $f(\mathcal{D})$ among our randomly sampled data sets, we would probably reject the hypothesis that $\mathcal{D}_{train}$ was generated from $\mathcal{M}$. Of course, there are many choices regarding which properties f we should evaluate. One natural choice is to define f as the empirical log-loss in the data set, $\mathbf{E}_{\mathcal{D}}[loss(\xi : \mathcal{M})]$, as per equation (16.1). We can then ask whether the empirical log-loss for $\mathcal{D}_{train}$ differs significantly from the expected empirical log-loss for data set $\mathcal{D}$ sampled from $\mathcal{M}$. Note that the expected value of this last expression is simply the entropy of $\mathcal{M}$, and, as we saw in section 8.4, we can compute the entropy of a Bayesian network fairly efficiently. To check for significance, we also need to consider the tail distribution of the log-loss, which is more involved. However, we can approximate that computation by computing the variance of the log-loss as a function of M. Alternatively, because generating samples from a Bayesian network is relatively inexpensive (as in section 12.1.1), we might find it easier to generate a large number of data sets $\mathcal{D}$ of size M sampled from the model and use those to estimate the distribution over $\mathbf{E}_{\mathcal{D}}[loss(\xi : \mathcal{M})]$.

Box 16.B — Concept: PAC-bounds. *As we discussed, given a target loss function, we can estimate the empirical risk on our training set $\mathcal{D}_{train}$. However, because of possible overfitting to the training data, the performance of our learned model on the training set might not be representative of its performance on unseen data. One might hope, however, that these two quantities are related, so that a model that achieves low training loss also achieves low expected loss (risk).*

Before we tackle a proof of this type, however, we must realize that we cannot guarantee with certainty the quality of our learned model. Recall that the data set $\mathcal{D}$ is sampled stochastically from P^*, so there is always a chance that we would have "bad luck" and sample a very unrepresentative

data set from P^*. For example, we might sample a data set where we get the same joint assignment in all of the instances. It is clear that we cannot expect to learn useful parameters from such a data set (assuming, of course, that P^* is not degenerate). The probability of getting such a data set is very low, but it is not zero. Thus, our analysis must allow for the chance that our data set will be highly unrepresentative, in which case our learned model (which presumably performed well on the training set) may not perform well on expectation.

probably approximately correct

Our goal is then to prove that our learning procedure is probably approximately correct*: that is, for most training sets $\mathcal{D}$, the learning procedure will return a model whose error is low. Making this discussion concrete, assume we use relative entropy to the true distribution as our loss function. Let P^*_M be the distribution over data sets $\mathcal{D}$ of size M sampled IID from P^*. Now, assume that we have a learning procedure L that, given a data set $\mathcal{D}$, returns a model $\mathcal{M}_{L(\mathcal{D})}$. We want to prove results of the form:*

> *Let $\epsilon > 0$ be our* approximation parameter *and $\delta > 0$ our* confidence parameter. *Then, for M "large enough," we have that*
>
> $$P^*_M(\{\mathcal{D} : \boldsymbol{D}(P^* \| P_{\mathcal{M}_{L(\mathcal{D})}}) \leq \epsilon\}) \geq 1 - \delta.$$

That is, for sufficiently large M, we have that, for most data sets $\mathcal{D}$ of size M sampled from P^*, the learning procedure, applied to $\mathcal{D}$, will learn a close approximation to P^*. The number of samples M required to achieve such a bound is called the sample complexity. *This type of result is called a* PAC-bound.

sample complexity

PAC-bound

This type of bound can only be obtained if the hypothesis space contains a model that can correctly represent P^*. In many cases, however, we are learning with a hypothesis space that is not guaranteed to be able to express P^*. In this case, we cannot expect to learn a model whose relative entropy to P^* is guaranteed to be low. In such a setting, the best we can hope for is to get a model whose error is at most ϵ worse than the lowest error found within our hypothesis space. The expected loss beyond the minimal possible error is called the excess risk. *See section 17.6.2.2 for one example of a generalization bound for this case.*

excess risk

16.3.2 Discriminative versus Generative Training

In the previous discussion, we implicitly assumed that our goal is to get the learned model $\tilde{\mathcal{M}}$ to be a good approximation to P^*. However, as we discussed in section 16.2.2, we often know in advance that we want the model to perform well on a particular task, such as predicting $\boldsymbol{Y}$ from $\boldsymbol{X}$. The training regime that we described would aim to get $\tilde{\mathcal{M}}$ close to the overall joint distribution $P^*(\boldsymbol{Y}, \boldsymbol{X})$. This type of objective is known as *generative training*, because we are training the model to *generate* all of the variables, both the ones that we care to predict and the features that we use for the prediction. Alternatively, we can train the model *discriminatively*, where our goal is to get $\tilde{P}(\boldsymbol{Y} \mid \boldsymbol{X})$ to be close to $P^*(\boldsymbol{Y} \mid \boldsymbol{X})$. The same model class can be trained in these two different ways, producing different results.

generative training

discriminative training

Example 16.1

As the simplest example, consider a simple "star" Markov network structure with a single target variable Y connected by edges to each of a set of features $X_1, \ldots, X_n$. If we train the model generatively, we are learning a naive Markov *model, which, because the network is singly connected, is equivalent to a naive Bayes model. On the other hand, we can train the same network structure discriminatively, to obtain a good fit to $P^*(Y \mid X_1, \ldots, X_n)$. In this case, as we showed in example 4.20, we are learning a model that is a logistic regression model for Y given its features.* ■

naive Markov

Note that a model that is trained generatively can still be used for specific prediction tasks. For example, we often train a naive Bayes model generatively but use it for classification. However, a model that is trained for a particular prediction task $P(\boldsymbol{Y} \mid \boldsymbol{X})$ does not encode a distribution over $\boldsymbol{X}$, and hence it cannot be used to reach any conclusions about these variables.

Discriminative training can be used for any class of models. However, its application in the context of Bayesian networks is less appealing, since this form of training changes the interpretation of the parameters in the learned model. For example, if we discriminatively train a (directed) naive Bayes model, as in example 16.1, the resulting model would essentially represent the same logistic regression as before, except that the pairwise potentials between Y and each X_i would be locally normalized to look like a CPD. Moreover, most of the computational properties that facilitate Bayesian network learning do not carry through to discriminative training. For this reason, discriminative training is usually performed in the context of undirected models. In this setting, we are essentially training a *conditional random field* (CRF), as in section 4.6.1: a model that directly encodes a conditional distribution $P(\boldsymbol{Y} \mid \boldsymbol{X})$.

conditional random field

There are various trade-offs between generative and discriminative training, both statistical and computational, and the question of which to use has been the topic of heated debates. We now briefly enumerate some of these trade-offs.

bias

Generally speaking, generative models have a higher *bias* — they make more assumptions about the form of the distribution. First, they encode independence assumptions about the feature variables $\boldsymbol{X}$, whereas discriminative models make independence assumptions only about $\boldsymbol{Y}$ and about their dependence on $\boldsymbol{X}$. An alternative intuition arises from the following view. A generative model defines $\tilde{P}(\boldsymbol{Y}, \boldsymbol{X})$, and thereby also induces $\tilde{P}(\boldsymbol{Y} \mid \boldsymbol{X})$ and $\tilde{P}(\boldsymbol{X})$, using the same overall model for both. To obtain a good fit to P^*, we must therefore tune our model to get good fits to both $P^*(\boldsymbol{Y} \mid \boldsymbol{X})$ and $P^*(\boldsymbol{X})$. Conversely, a discriminative model aims to get a good fit only to $P^*(\boldsymbol{Y} \mid \boldsymbol{X})$, without constraining the same model to provide a good fit to $P^*(\boldsymbol{X})$ as well.

☞ **The additional bias in the setting offers a standard trade-off. On one hand, it can help regularize and constrain the learned model, thereby reducing its ability to overfit the data. Therefore, generative training often works better when we are learning from limited amounts of data.** However, imposing constraints can hurt us when the constraints are wrong, by preventing us from learning the correct model. In practice, the class of models we use always imposes some constraints that do not hold in the true generating distribution P^*. For limited amounts of data, the constraints might still help reduce overfitting, giving rise to better generalization. **However, as the amount of data grows, the bias imposed by the constraints** ☞ **starts to dominate the error of our learned model. Because discriminative models make fewer assumptions, they will tend to be less affected by incorrect model assumptions and will often outperform the generatively trained models for larger data sets.**

Example 16.2

Consider the problem of optical character recognition — identifying letters from handwritten images. Here, the target variable Y is the character label (for example, "A"). Most obviously, we can use the individual pixels as our feature variables $X_1, \ldots, X_n$. We can then either generatively train a naive Markov model or discriminatively train a logistic regression model. The naive Bayes (or Markov) model separately learns the distribution over the 256 pixel values given each of the 26 labels; each of these is estimated independently, giving rise to a set of fairly low-dimensional estimation problems. Conversely, the discriminative model is jointly optimizing all of the approximately 26×256 parameters of the multinomial logit distribution, a much higher-dimensional estimation problem. Thus, for sparse data, the naive Bayes model may often perform better.

However, even in this simple setting, the independence assumption made by the naive Bayes model — that pixels are independent given the image label — is clearly false. As a consequence, the naive Bayes model may be counting, as independent, features that are actually correlated, leading to errors in the estimation. The discriminative model is not making these assumptions; by fitting the parameters jointly, it can compensate for redundancy and other correlations between the features. Thus, as we get enough data to fit the logistic model reasonably well, we would expect it to perform better. ■

A related benefit of discriminative models is that they are able to make use a much richer feature set, where independence assumptions are clearly violated. These richer features can often greatly improve classification accuracy.

Example 16.3

Continuing our example, the raw pixels are fairly poor features to use for the image classification task. Much work has been spent by researchers in computer vision and image processing in developing richer feature sets, such as the direction of the edge at a given image pixel, the value of a certain filter applied to an image patch centered at the pixel, and many other features that are even more refined. In general, we would expect to be able to classify images much better using these features than using the raw pixels directly. However, each of these features depends on the values of multiple pixels, and the same pixels are used in computing the values of many different features. Therefore, these features are certainly not independent, and using them in the context of a naive Bayes classifier is likely to lead to fairly poor answers. However, there is no reason not to include such correlated features within a logistic regression or other discriminative classifier. ■

Conversely, generative models have their own advantages. They often offer a more natural interpretation of a domain. And they are better able to deal with missing values and unlabeled data. Thus, the appropriate choice of model is application dependent, and often a combination of different training regimes may be the best choice.

16.4 Learning Tasks

We now discuss in greater detail the different variants of the learning task. As we briefly mentioned, the input of a learning procedure is:

- Some prior knowledge or constraints about $\tilde{\mathcal{M}}$.
- A set $\mathcal{D}$ of data instances $\{\boldsymbol{d}[1], \ldots, \boldsymbol{d}[M]\}$, which are independent and identically distributed (IID) samples from P^*.

The output is a model $\tilde{\mathcal{M}}$, which may include the structure, the parameters, or both.

There are many variants of this fairly abstract learning problem; roughly speaking, they vary along three axes, representing the two types of input and the type of output. First, and most obviously, the problem formulation depends on our output — the type of graphical model we are trying to learn — a Bayesian network or a Markov network. The other two axes summarize the input of the learning procedure. The first of these two characterizes the extent of the constraints that we are given about $\tilde{\mathcal{M}}$, and the second characterizes the extent to which the data in our training set are fully observed. We now discuss each of these in turn. We then present a taxonomy of the different tasks that are defined by these axes, and we review some of their computational implications.

16.4.1 Model Constraints

hypothesis space

The first question is the extent to which our input constrains the *hypothesis space* — the class of models that we are allowed to consider as possible outputs of our learning algorithm. There is an almost unbounded set of options here, since we can place various constraints on the structure or on the parameters of the model. Some of the key points along the spectrum are:

- At one extreme, we may be given a graph structure, and we have to learn only (some of) the parameters; note that we generally do not assume that the given structure is necessarily the correct one $\mathcal{K}^*$.
- We may not know the structure, and we have to learn both parameters and structure from the data.
- Even worse, we may not even know the complete set of variables over which the distribution P^* is defined. In other words, we may only observe some subset of the variables in the domain and possibly be unaware of others.

The less prior knowledge we are given, the larger the hypothesis space, and the more possibilities we need to consider when selecting a model. As we discussed in section 16.3.1, the complexity of the hypothesis space defines several important trade-offs. The first is statistical. If we restrict the hypothesis space too much, it may be unable to represent P^* adequately. Conversely, if we leave it too flexible, our chances increase of finding a model within the hypothesis space that accidentally has high score but is a poor fit to P^*. The second trade-off is computational: in many cases (although not always), the richer the hypothesis space, the more difficult the search to find a high-scoring model.

16.4.2 Data Observability

data observability

Along the second input axis, the problem depends on the extent of the *observability* of our training set. Here, there are several options:

complete data

- The data are *complete*, or *fully observed*, so that each of our training instances $\boldsymbol{d}[m]$ is a full instantiation to all of the variables in $\mathcal{X}^*$.

incomplete data

- The data are *incomplete*, or *partially observed*, so that, in each training instance, some variables are not observed.

hidden variable

- The data contain *hidden variables* whose value is never observed in any training instance. This option is the only one compatible with the case where the set of variables $\mathcal{X}^*$ is unknown, but it may also arise if we know of the existence of a hidden variable but never have the opportunity to observe it directly.

As we move along this axis, more and more aspects of our data are unobserved. When data are unobserved, we must hypothesize possible values for these variables. The greater the extent to which data are missing, the less we are able to hypothesize reliably values for the missing entries.

Dealing with partially observed data is critical in many settings. First, in many settings, observing the values of all variables can be difficult or even impossible. For example, in the case of patient records, we may not perform all tests on all patients, and therefore some of the variables may be unobserved in some records. Other variables, such as the disease the patient had, may never be observed with certainty. The ability to deal with partially observed data cases is also crucial to adapting a Bayesian network using data cases obtained after the network is operational. In such situations, the training instances are the ones provided to the network as queries, and as such, are never fully observed (at least when presented as a query).

A particularly difficult case of missing data occurs when we have hidden variables. Why should we worry about learning such variables? For the task of knowledge discovery, these variables may play an important role in the model, and therefore they may be critical for our understanding of the domain. For example, in medical settings, the genetic susceptibility of a patient to a particular disease might be an important variable. This might be true even if we do not know what the genetic cause is, and thus cannot observe it. As another example, the tendency to be an "impulse shopper" can be an important hidden variable in an application to supermarket data mining. In these cases, our domain expert can find it convenient to specify a model that contains these variables, even if we never expect to observe their values directly.

In other cases, we might care about the hidden variable even when it has no predefined semantic meaning. Consider, for example, a naive Bayes model, such as the one shown in figure 3.2, but where we assume that the X_i's are observed but the class variable C is hidden. In this model, we have a *mixture distribution*: Each value of the hidden variable represents a separate distribution over the X_i's, where each such mixture component distribution is "simple" — all of the X_i's are independent in each of the mixture components. Thus, the population is composed of some number of separate subpopulations, each of which is generated by a distinct distribution. If we could learn this model, we could recover the distinct subpopulations, that is, figure out what types of individuals we have in our population. This type of analysis is very useful from the perspective of knowledge discovery.

mixture distribution

Finally, we note that **the inclusion of a hidden variable in the network can greatly simplify the structure, reducing the complexity of the network that needs to be learned.** Even a sparse model over some set of variables can induce a large number of dependencies over a subset of its variables; for example, returning to the earlier naive Bayes example, if the class variable C is hidden and therefore is not included in the model, the distribution over the variables $X_1, \ldots, X_n$ has no independencies and requires a fully connected graph to be represented correctly. Figure 16.1 shows another example. (This figure illustrates another visual convention that will accompany us throughout this part of the book: Variables whose values are always hidden are shown as white ovals.) Thus, in many cases, ignoring the hidden variable

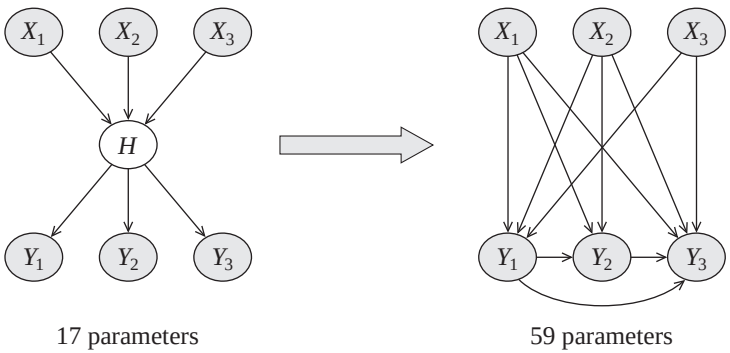


Figure 16.1 The effect of ignoring hidden variables. The model on the right is an I-map for the distribution represented by the model on the left, where the hidden variable is marginalized out. The counts indicate the number of independent parameters, under the assumption that the variables are binary-valued. The variable H is hidden and hence is shown as a white oval.

leads to a significant increase in the complexity of the "true" model (the one that best fits P^*), making it harder to estimate robustly. Conversely, learning a model that usefully incorporates a hidden variable is far from trivial. Thus, the decision of whether to incorporate a hidden variable is far from trivial, and it requires a careful evaluation of the trade-offs.

16.4.3 Taxonomy of Learning Tasks

Based on these three axes, we can provide a taxonomy of different learning tasks and discuss some of the computational issues they raise.

The problem of parameter estimation for a known structure is one of numerical optimization. Although straightforward in principle, this task is an important one, both because numbers are difficult to elicit from people and because parameter estimation forms the basis for the more advanced learning scenarios.

In the case of Bayesian networks, when the data are complete, the parameter estimation problem is generally easily solved, and it often even admits a closed-form solution. Unfortunately, this very convenient property does not hold for Markov networks. Here, the global partition function induces entanglement of the parameters, so that the dependence of the distribution on any single parameter is not straightforward. Nevertheless, for the case of a fixed structure and complete data, the optimization problem is convex and can be solved optimally using simple iterated numerical optimization algorithms. Unfortunately, each step of the optimization algorithm requires inference over the network, which can be expensive for large models.

When the structure is not given, the learning task now incorporates an additional level of complexity: the fact that our hypothesis space now contains an enormous (generally superexponentially large) set of possible structures. In most cases, as we will see, the problem of structure selection is also formulated as an optimization problem, where different network structures are given a score, and we aim to find the network whose score is highest. In the case of Bayesian networks, the same property that allowed a closed-form solution for the parameters also allows

the score for a candidate network to be computed in closed form. In the case of Markov network, most natural scores for a network structure cannot be computed in closed form because of the partition function. However, we can define a convex optimization problem that jointly searches over parameter and structure, allowing for a single global optimum.

The problem of dealing with incomplete data is much more significant. Here, the multiple hypotheses regarding the values of the unobserved variables give rise to a combinatorial range of different alternative models, and induce a nonconvex, multimodal optimization problem even in parameter space. The known algorithms generally work by iteratively using the current parameters to fill in values for the missing data, and then using the completion to reestimate the model parameters. This process requires multiple calls to inference as a subroutine, making this process expensive for large networks. The case where the structure is not known is even harder, since we need to combine a discrete search over network structure with nonconvex optimization over parameter space.

16.5 Relevant Literature

Most of the topics reviewed here are discussed in greater technical depth in subsequent chapters, and so we defer the bibliographic references to the appropriate places. Hastie, Tibshirani, and Friedman (2001) and Bishop (2006) provide an excellent overview of basic concepts in machine learning, many of which are relevant to the discussion in this book.

17 *Parameter Estimation*

In this chapter, we discuss the problem of estimating parameters for a Bayesian network. We assume that the network structure is fixed and that our data set $\mathcal{D}$ consists of fully observed instances of the network variables: $\mathcal{D} = \{\xi[1], \ldots, \xi[M]\}$. This problem arises fairly often in practice, since numerical parameters are harder to elicit from human experts than structure is. It also plays a key role as a building block for both structure learning and learning from incomplete data. As we will see, despite the apparent simplicity of our task definition, there is surprisingly much to say about it.

As we will see, there are two main approaches to dealing with the parameter-estimation task: one based on *maximum likelihood estimation*, and the other using Bayesian approaches. For each of these approaches, we first discuss the general principles, demonstrating their application in the simplest context: a Bayesian network with a single random variable. We then show how the structure of the distribution allows the techniques developed in this very simple case to generalize to arbitrary network structures. Finally, we show how to deal with parameter estimation in the context of structured CPDs.

17.1 Maximum Likelihood Estimation

In this section, we describe the basic principles behind maximum likelihood estimation.

17.1.1 The Thumbtack Example

We start with what may be considered the simplest learning problem: parameter learning for a single variable. This is a classical Statistics 101 problem that illustrates some of the issues that we will encounter in more complex learning problems. Surprisingly, this simple problem already contains some interesting issues that we need to tackle.

Imagine that we have a thumbtack, and we conduct an experiment whereby we flip the thumbtack in the air. It comes to land as either heads or tails, as in figure 17.1. We toss the thumbtack several times, obtaining a data set consisting of *heads* or *tails* outcomes. Based on this data set, we want to estimate the probability with which the next flip will land heads or tails. In this description, we already made the implicit assumption that the thumbtack tosses are controlled by an (unknown) *parameter* θ, which describes the frequency of heads in thumbtack tosses. In addition, we also assume that the data instances are independent and identically distributed (IID).

Figure 17.1 A simple thumbtack tossing experiment

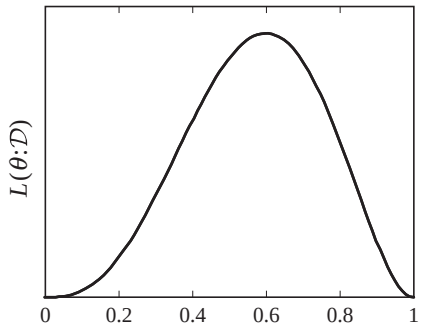


Figure 17.2 The likelihood function for the sequence of tosses H, T, T, H, H

Assume that we toss the thumbtack 100 times, of which 35 come up heads. What is our estimate for θ? Our intuition suggests that the best estimate is 0.35. Had θ been 0.1, for example, our chances of seeing $35/100$ heads would have been much lower. In fact, we examined a similar situation in our discussion of sampling methods in section 12.1, where we used samples from a distribution to estimate the probability of a query. As we discussed, the central limit theorem shows that, as the number of coin tosses grows, it is increasingly unlikely to sample a sequence of IID thumbtack flips where the fraction of tosses that come out heads is very far from θ. Thus, for sufficiently large M, the fraction of heads among the tosses is a good estimate with high probability.

To formalize this intuition, assume that we have a set of thumbtack tosses $x[1], \ldots, x[M]$ that are IID, that is, each is sampled independently from the same distribution in which $X[m]$ is equal to H (heads) or T (tails) with probability θ and $1 - \theta$, respectively. Our task is to find a good value for the parameter θ. As in many formulations of learning tasks, we define a *hypothesis space* Θ — a set of possibilities that we are considering — and an *objective function* that tells us how good different hypotheses in the space are relative to our data set $\mathcal{D}$. In this case, our hypothesis space Θ is the set of all parameters $\theta \in [0, 1]$.

hypothesis space

objective function

How do we score different possible parameters θ? As we discussed in section 16.3.1, one way of evaluating θ is by how well it predicts the data. In other words, if the data are likely given the parameter, the parameter is a good predictor. For example, suppose we observe the sequence of outcomes H, T, T, H, H. If we know θ, we could assign a probability to observing this particular sequence. The probability of the first toss is $P(X[1] = H) = \theta$. The probability of the second toss is $P(X[2] = T \mid X[1] = H)$, but our assumption that the coin tosses are independent allows us to conclude that this probability is simply $P(X[2] = T) = 1 - \theta$. This

is also the probability of the third outcome, and so on. Thus, the probability of the sequence is

$$P(\langle H, T, T, H, H\rangle : \theta) = \theta(1-\theta)(1-\theta)\theta\theta = \theta^3(1-\theta)^2.$$

As expected, this probability depends on the particular value θ. As we consider different values of θ, we get different probabilities for the sequence. Thus, we can examine how the probability of the data changes as a *function* of θ. We thus define the *likelihood function* to be

likelihood function

$$L(\theta : \langle H, T, T, H, H\rangle) = P(\langle H, T, T, H, H\rangle : \theta) = \theta^3(1-\theta)^2.$$

Figure 17.2 plots the likelihood function in our example.

Clearly, parameter values with higher likelihood are more likely to generate the observed sequences. Thus, we can use the likelihood function as our measure of quality for different parameter values and select the parameter value that maximizes the likelihood; this value is called the *maximum likelihood estimator (MLE)*. By viewing figure 17.2 we see that $\hat{\theta} = 0.6 = 3/5$ maximizes the likelihood for the sequence H, T, T, H, H.

maximum likelihood estimator

Can we find the MLE for the general case? Assume that our data set $\mathcal{D}$ of observations contains $M[1]$ heads and $M[0]$ tails. We want to find the value $\hat{\theta}$ that maximizes the likelihood of θ relative to $\mathcal{D}$. The likelihood function in this case is:

$$L(\theta : \mathcal{D}) = \theta^{M[1]}(1-\theta)^{M[0]}.$$

It turns out that it is easier to maximize the logarithm of the likelihood function. In our case, the *log-likelihood* function is:

log-likelihood

$$\ell(\theta : \mathcal{D}) = M[1]\log\theta + M[0]\log(1-\theta).$$

Note that the log-likelihood is monotonically related to the likelihood. Therefore, maximizing the one is equivalent to maximizing the other. However, the log-likelihood is more convenient to work with, since products are converted to summations.

Differentiating the log-likelihood, setting the derivative to 0, and solving for θ, we get that the maximum likelihood parameter, which we denote $\hat{\theta}$, is

$$\hat{\theta} = \frac{M[1]}{M[1] + M[0]}, \tag{17.1}$$

as expected (see exercise 17.1).

As we will see, the maximum likelihood approach has many advantages. However, the approach also has some limitations. For example, if we get 3 heads out of 10 tosses, the MLE estimate is 0.3. We get the same estimate if we get 300 heads out of 1,000 tosses. Clearly, the two experiments are not equivalent. Our intuition is that, in the second experiment, we should be more confident of our estimate. Indeed, statistical estimation theory deals with *confidence intervals*. These are common in news reports, for example, when describing the results of election polls, where we often hear that "61 ± 2 percent" plan to vote for a certain candidate. The 2 percent is a confidence interval — the poll is designed to select enough people so that the MLE estimate will be within 0.02 of the true parameter, with high probability.

confidence interval

17.1.2 The Maximum Likelihood Principle

We now generalize the discussion of maximum likelihood estimation to a broader range of learning problems. We then consider how to apply it to the task of learning the parameters of a Bayesian network.

We start by describing the setting of the learning problem. Assume that we observe several IID samples of a set of random variables $\mathcal{X}$ from an unknown distribution $P^*(\mathcal{X})$. We assume we know in advance the sample space we are dealing with (that is, which random variables, and what values they can take). However, we do not make any additional assumptions about P^*.

training set

We denote the *training set* of samples as $\mathcal{D}$ and assume that it consists of M instances of $\mathcal{X}$: $\xi[1], \ldots \xi[M]$.

parametric model

Next, we need to consider what exactly we want to learn. We assume that we are given a *parametric model* for which we wish to estimate *parameters*. Formally, a *parametric model* (also known as a parametric family; see section 8.2) is defined by a function $P(\xi : \boldsymbol{\theta})$, specified in terms of a set of *parameters*. Given a particular set of parameter values $\boldsymbol{\theta}$ and an instance ξ of $\mathcal{X}$, the model assigns a probability (or density) to ξ. Of course, we require that for each choice of parameters $\boldsymbol{\theta}$, $P(\xi : \boldsymbol{\theta})$ is a legal distribution; that is, it is nonnegative and

$$\sum_{\xi} P(\xi : \boldsymbol{\theta}) = 1.$$

parameter space

In general, for each model, not all parameter values are legal. Thus, we need to define the *parameter space* Θ, which is the set of allowable parameters.

To get some intuition, we consider concrete examples. The model we examined in section 17.1.1 has parameter space $\Theta_{thumbtack} = [0, 1]$ and is defined as

$$P_{thumbtack}(x : \theta) = \begin{cases} \theta & \text{if } x = H \\ 1 - \theta & \text{if } x = T. \end{cases}$$

There are many additional examples.

Example 17.1

multinomial

Suppose that X is a multinomial *variable that can take values $x^1, \ldots, x^K$. The simplest representation of a multinomial distribution is as a vector $\boldsymbol{\theta} \in \mathbb{R}^K$, such that*

$$P_{multinomial}(x : \boldsymbol{\theta}) = \theta_k \text{ if } x = x^k.$$

The parameter space of this model is

$$\Theta_{multinomial} = \left\{ \boldsymbol{\theta} \in [0, 1]^K : \sum_i \theta_i = 1 \right\}.$$

■

Example 17.2

Gaussian

Suppose that X is a continuous variable that can take values in the real line. A Gaussian *model for X is*

$$P_{Gaussian}(x : \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where $\boldsymbol{\theta} = \langle \mu, \sigma \rangle$. The parameter space for this model is $\Theta_{Gaussian} = \mathbb{R} \times \mathbb{R}^+$. That is, we allow any real value of μ and any positive real value for σ. ■

likelihood function

The next step in maximum likelihood estimation is defining the *likelihood function*. As we saw in our example, the likelihood function for a given choice of parameters $\boldsymbol{\theta}$ is the probability (or density) the model assigns the training data:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_m P(\xi[m] : \boldsymbol{\theta}).$$

In the thumbtack example, we saw that we can write the likelihood function using simpler terms. That is, using the counts $M[1]$ and $M[0]$, we managed to have a compact description of the likelihood. More precisely, once we knew the values of $M[1]$ and $M[0]$, we did not need to consider other aspects of training data (for example, the order of tosses). These are the *sufficient statistics* for the thumbtack learning problem. In a more general setting, a sufficient statistic is a function of the data that summarizes the relevant information for computing the likelihood.

Definition 17.1

sufficient statistics

A function $\tau(\xi)$ from instances of $\mathcal{X}$ to $\mathbb{R}^\ell$ (for some ℓ) is a sufficient statistic *if, for any two data sets $\mathcal{D}$ and $\mathcal{D}'$ and any $\boldsymbol{\theta} \in \Theta$, we have that*

$$\sum_{\xi[m] \in \mathcal{D}} \tau(\xi[m]) = \sum_{\xi'[m] \in \mathcal{D}'} \tau(\xi'[m]) \quad \Longrightarrow \quad L(\boldsymbol{\theta} : \mathcal{D}) = L(\boldsymbol{\theta} : \mathcal{D}').$$

■

We often refer to the tuple $\sum_{\xi[m] \in \mathcal{D}} \tau(\xi[m])$ as the sufficient statistics of the data set $\mathcal{D}$.

Example 17.3

Let us reconsider the multinomial model of example 17.1. It is easy to see that a sufficient statistic for the data set is the tuple of counts $\langle M[1], \ldots, M[K] \rangle$, *such that $M[k]$ is number of times the value x^k appears in the training data. To obtain these counts by summing instance-level statistics, we define $\tau(x)$ to be a tuple of dimension K, such that $\tau(x)$ has a 0 in every position, except at the position k for which $x = x^k$, where its value is 1:*

$$\tau(x^k) = (\overbrace{0, \ldots, 0}^{k-1}, 1, \overbrace{0, \ldots, 0}^{n-k}).$$

Given the vector of counts, we can write the likelihood function as

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_k \theta_k^{M[k]}.$$

■

Example 17.4

Let us reconsider the Gaussian model of example 17.2. In this case, it is less obvious how to construct sufficient statistics. However, if we expand the term $(x - \mu)^2$ in the exponent, we can rewrite the model as

$$P_{Gaussian}(x : \mu, \sigma) = e^{-x^2 \frac{1}{2\sigma^2} + x \frac{\mu}{\sigma^2} - \frac{\mu^2}{2\sigma^2} - \frac{1}{2}\log(2\pi) - \log(\sigma)}.$$

We then see that the function

$$s_{Gaussian}(x) = \langle 1, x, x^2 \rangle$$

is a sufficient statistic for this model. Note that the first element in the sufficient statistics tuple is "1," which does not depend on the value of the data item; it serves, as in the multinomial case, to count the number of data items. ■

We venture several comments about the likelihood function. First, we stress that the likelihood function measures the effect of the choice of parameters on the training data. Thus, for example, if we have two sets of parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, so that $L(\boldsymbol{\theta} : \mathcal{D}) = L(\boldsymbol{\theta}' : \mathcal{D})$, then we *cannot*, given only the data, distinguish between the two choices of parameters. Moreover, if $L(\boldsymbol{\theta} : \mathcal{D}) = L(\boldsymbol{\theta}' : \mathcal{D})$ for all possible choices of $\mathcal{D}$, then the two parameters are *indistinguishable* for any outcome. In such a situation, we can say in advance (that is, before seeing the data) that some distinctions cannot be resolved based on the data alone.

Second, since we are maximizing the likelihood function, we usually want it to be continuous (and preferably smooth) function of $\boldsymbol{\theta}$. To ensure these properties, most of the theory of statistical estimation requires that $P(\xi : \boldsymbol{\theta})$ is a continuous and differentiable function of $\boldsymbol{\theta}$, and moreover that Θ is a continuous set of points (which is often assumed to be convex).

maximum likelihood estimation

Once we have defined the likelihood function, we can use *maximum likelihood estimation* to choose the parameter values. Formally, we state this principle as follows.

Maximum Likelihood Estimation: Given a data set $\mathcal{D}$, choose parameters $\hat{\boldsymbol{\theta}}$ that satisfy

$$L(\hat{\boldsymbol{\theta}} : \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta} : \mathcal{D}).$$

Example 17.5

Consider estimating the parameters of the multinomial distribution of example 17.3. As one might guess, the maximum likelihood is attained when

$$\hat{\theta}_k = \frac{M[k]}{M}$$

(see exercise 17.2). That is, the probability of each value of X corresponds to its frequency in the training data. ■

Example 17.6

empirical mean, variance

Consider estimating the parameters of a Gaussian distribution of example 17.4. It turns out that the maximum is attained when μ and σ correspond to the empirical *mean and variance of the training data:*

$$\begin{aligned} \hat{\mu} &= \frac{1}{M} \sum_m x[m] \\ \hat{\sigma} &= \sqrt{\frac{1}{M} \sum_m (x[m] - \hat{\mu})^2} \end{aligned}$$

(see exercise 17.3). ■

17.2 MLE for Bayesian Networks

We now move to the more general problem of estimating parameters for a Bayesian network. It turns out that the structure of the Bayesian network allows us to reduce the parameter estimation problem to a set of unrelated problems, each of which can be addressed using the techniques of the previous section. We begin by considering a simple example to clarify our intuition, and then generalize to more complicated networks.

17.2.1 A Simple Example

The simplest example of a nontrivial network structure is a network consisting of two binary variables, say X and Y, with an arc $X \to Y$. (A network without such an arc trivially reduces to the cases we already discussed.)

As for a single parameter, our goal in maximum likelihood estimation is to maximize the likelihood (or log-likelihood) function. In this case, our network is parameterized by a parameter vector $\boldsymbol{\theta}$, which defines the set of parameters for all the CPDs in the network. In this example, our parameterization would consist of the following parameters: θ_{x^1}, and θ_{x^0} specify the probability of the two values of X; $\theta_{y^1|x^1}$, and $\theta_{y^0|x^1}$ specify the probability of Y given that $X = x^1$; and $\theta_{y^1|x^0}$, and $\theta_{y^0|x^0}$ describe the probability of Y given that $X = x^0$. For brevity, we also use the shorthand $\boldsymbol{\theta}_{Y|x^0}$ to refer to the set $\{\theta_{y^1|x^0}, \theta_{y^0|x^0}\}$, and $\boldsymbol{\theta}_{Y|X}$ to refer to $\boldsymbol{\theta}_{Y|x^1} \cup \boldsymbol{\theta}_{Y|x^0}$.

In this example, each training instance is a tuple $\langle x[m], y[m]\rangle$ that describes a particular assignment to X and Y. Our likelihood function is:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_{m=1}^{M} P(x[m], y[m] : \boldsymbol{\theta}).$$

Our network model specifies that $P(X, Y : \boldsymbol{\theta})$ has a product form. Thus, we can write

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_m P(x[m] : \boldsymbol{\theta}) P(y[m] \mid x[m] : \boldsymbol{\theta}).$$

Exchanging the order of multiplication, we can equivalently write this term as

$$L(\boldsymbol{\theta} : \mathcal{D}) = \left(\prod_m P(x[m] : \boldsymbol{\theta})\right) \left(\prod_m P(y[m] \mid x[m] : \boldsymbol{\theta})\right).$$

That is, the likelihood decomposes into two separate terms, one for each variable. Moreover, each of these terms is a *local likelihood* function that measures how well the variable is predicted given its parents.

Now consider the two individual terms. Clearly, each one depends only on the parameters for that variable's CPD. Thus, the first is $\prod_m P(x[m] : \boldsymbol{\theta}_X)$. This term is identical to the multinomial likelihood function we discussed earlier. The second term is more interesting, since we can decompose it even further:

$$\begin{aligned}
&\prod_m P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|X}) \\
&\quad = \prod_{m:x[m]=x^0} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|X}) \cdot \prod_{m:x[m]=x^1} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|X}) \\
&\quad = \prod_{m:x[m]=x^0} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^0}) \cdot \prod_{m:x[m]=x^1} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^1}).
\end{aligned}$$

Thus, in this example, the likelihood function decomposes into a product of terms, one for each group of parameters in $\boldsymbol{\theta}$. This property is called the *decomposability* of the likelihood function.

likelihood decomposability

We can do one more simplification by using the notion of sufficient statistics. Let us consider one term in this expression:

$$\prod_{m:x[m]=x^0} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^0}). \tag{17.2}$$

Each of the individual terms $P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^0})$ can take one of two values, depending on the value of $y[m]$. If $y[m] = y^1$, it is equal to $\theta_{y^1|x^0}$. If $y[m] = y^0$, it is equal to $\theta_{y^0|x^0}$. How many cases of each type do we get? First, we restrict attention only to those data cases where $x[m] = x^0$. These, in turn, partition into the two categories. Thus, we get $\theta_{y^1|x^0}$ in those data cases where $x[m] = x^0$ and $y[m] = y^1$; we use $M[x^0, y^1]$ to denote their number. We get $\theta_{y^0|x^0}$ in those data cases where $x[m] = x^0$ and $y[m] = x^0$, and use $M[x^0, y^0]$ to denote their number. Thus, the term in equation (17.2) is equal to:

$$\prod_{m:x[m]=x^0} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^0}) \quad = \quad \theta_{y^1|x^0}^{M[x^0,y^1]} \cdot \theta_{y^0|x^0}^{M[x^0,y^0]}.$$

Based on our discussion of the multinomial likelihood in example 17.5, we know that we maximize $\theta_{Y|x^0}$ by setting:

$$\theta_{y^1|x^0} = \frac{M[x^0, y^1]}{M[x^0, y^1] + M[x^0, y^0]} = \frac{M[x^0, y^1]}{M[x^0]},$$

and similarly for $\theta_{y^0|x^0}$. Thus, we can find the maximum likelihood parameters in this CPD by simply counting how many times each of the possible assignments of X and Y appears in the training data. It turns out that these counts of the various assignments for some set of variables are useful in general. We therefore define:

Definition 17.2

Let $\boldsymbol{Z}$ be some set of random variables, and $\boldsymbol{z}$ be some instantiation to these random variables. Let $\mathcal{D}$ be a data set. We define $M[\boldsymbol{z}]$ to be the number of entries in $\mathcal{D}$ that have $\boldsymbol{Z}[m] = \boldsymbol{z}$

$$M[\boldsymbol{z}] = \sum_m \mathbb{1}\{\boldsymbol{Z}[m] = \boldsymbol{z}\}. \tag{17.3}$$

■

17.2.2 Global Likelihood Decomposition

As we can expect, the arguments we used for deriving the MLE of $\theta_{Y|x^0}$ apply for the parameters of other CPDs in that example and indeed for other networks as well. We now develop, in several steps, the formal machinery for proving such properties in Bayesian networks.

We start by examining the likelihood function of a Bayesian network. Suppose we want to learn the parameters for a Bayesian network with structure $\mathcal{G}$ and parameters $\boldsymbol{\theta}$. This means that we agree in advance on the type of CPDs we want to learn (say table-CPDs, or noisy-ors). As we discussed, we are also given a data set $\mathcal{D}$ consisting of samples $\xi[1], \ldots, \xi[M]$. Writing

the likelihood, and repeating the steps we performed in our example, we get

$$\begin{aligned} L(\boldsymbol{\theta} : \mathcal{D}) &= \prod_m P_{\mathcal{G}}(\xi[m] : \boldsymbol{\theta}) \\ &= \prod_m \prod_i P(x_i[m] \mid \mathrm{pa}_{X_i}[m] : \boldsymbol{\theta}) \\ &= \prod_i \left[\prod_m P(x_i[m] \mid \mathrm{pa}_{X_i}[m] : \boldsymbol{\theta}) \right]. \end{aligned}$$

conditional likelihood

Note that each of the terms in the square brackets refers to the *conditional likelihood* of a particular variable given its parents in the network. We use $\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}$ to denote the subset of parameters that determines $P(X_i \mid \mathrm{Pa}_{X_i})$ in our model. Then, we can write

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_i L_i(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} : \mathcal{D}),$$

local likelihood

where the *local likelihood* function for X_i is:

$$L_i(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} : \mathcal{D}) = \prod_m P(x_i[m] \mid \mathrm{pa}_{X_i}[m] : \boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}).$$

This form is particularly useful when the parameter sets $\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}$ are *disjoint*. That is, each CPD is parameterized by a separate set of parameters that do not overlap. This assumption is quite natural in all our examples so far. (Although, as we will see in section 17.5, parameter sharing can be handy in many domains.) **This analysis shows that the likelihood decomposes as a product of independent terms, one for each CPD in the network. This important property is called the *global decomposition* of the likelihood function.**

global decomposability

We can now immediately derive the following result:

Proposition 17.1 *Let $\mathcal{D}$ be a complete data set for $X_1, \ldots, X_n$, let $\mathcal{G}$ be a network structure over these variables, and suppose that the parameters $\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}$ are disjoint from $\boldsymbol{\theta}_{X_j|\mathrm{Pa}_{X_j}}$ for all $j \neq i$. Let $\hat{\boldsymbol{\theta}}_{X_i|\mathrm{Pa}_{X_i}}$ be the parameters that maximize $L_i(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} : \mathcal{D})$. Then, $\hat{\boldsymbol{\theta}} = \langle \hat{\boldsymbol{\theta}}_{X_1|\mathrm{Pa}_1}, \ldots, \hat{\boldsymbol{\theta}}_{X_n|\mathrm{Pa}_n} \rangle$ maximizes $L(\boldsymbol{\theta} : \mathcal{D})$.*

In other words, **we can maximize each local likelihood function *independently* of rest of the network, and then combine the solutions to get an MLE solution.** This decomposition of the global problem to independent subproblems allows us to devise efficient solutions to the MLE problem. Moreover, this decomposition is an immediate consequence of the network structure and does not depend on any particular choice of parameterization for the CPDs.

17.2.3 Table-CPDs

Based on the preceding discussion, we know that the likelihood of a Bayesian network decomposes into local terms that depend on the parameterization of CPDs. The choice of parameters determines how we can maximize each of the local likelihood functions. We now consider what is perhaps the simplest parameterization of the CPD: a *table-CPD*.

table-CPD

Suppose we have a variable X with parents $\boldsymbol{U}$. If we represent that CPD $P(X \mid \boldsymbol{U})$ as a table, then we will have a parameter $\theta_{x|\boldsymbol{u}}$ for each combination of $x \in Val(X)$ and $\boldsymbol{u} \in Val(\boldsymbol{U})$. In this case, we can rewrite the local likelihood function as follows:

$$\begin{aligned} L_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D}) &= \prod_m \theta_{x[m]|\boldsymbol{u}[m]} \\ &= \prod_{\boldsymbol{u} \in Val(\boldsymbol{U})} \left[\prod_{x \in Val(X)} \theta_{x|\boldsymbol{u}}^{M[\boldsymbol{u},x]} \right], \end{aligned} \tag{17.4}$$

where $M[\boldsymbol{u}, x]$ is the number of times $\xi[m] = x$ and $\boldsymbol{u}[m] = \boldsymbol{u}$ in $\mathcal{D}$. That is, we grouped together all the occurrences of $\theta_{x|\boldsymbol{u}}$ in the product over all instances. This provides a further *local decomposition* of the likelihood function.

local decomposability

We need to maximize this term under the constraints that, for each choice of value for the parents $\boldsymbol{U}$, the conditional probability is legal, that is:

$$\sum \theta_{x|\boldsymbol{u}} = 1 \quad \text{for all } \boldsymbol{u}.$$

These constraints imply that the choice of value for $\theta_{x|\boldsymbol{u}}$ can impact the choice of value for $\theta_{x'|\boldsymbol{u}}$. However, the choice of parameters given different values $\boldsymbol{u}$ of $\boldsymbol{U}$ are independent of each other. Thus, we can maximize each of the terms in square brackets in equation (17.4) independently.

We can thus further decompose the local likelihood function for a tabular CPD into a product of simple likelihood functions. Each of these likelihood functions is a *multinomial* likelihood, of the type that we examined in example 17.3. The counts in the data for the different outcomes x are simply $\{M[\boldsymbol{u}, x] : x \in Val(X)\}$. We can then immediately use the maximum likelihood estimation for multinomial likelihood of example 17.5 and see that the MLE parameters are

$$\hat{\theta}_{x|\boldsymbol{u}} = \frac{M[\boldsymbol{u}, x]}{M[\boldsymbol{u}]}, \tag{17.5}$$

where we use the fact that $M[\boldsymbol{u}] = \sum_x M[\boldsymbol{u}, x]$.

This simple formula reveals a key challenge when estimating parameters for a Bayesian networks. Note that the number of data points used to estimate the parameter $\hat{\theta}_{x|\boldsymbol{u}}$ is $M[\boldsymbol{u}]$. Data points that do not agree with the parent assignment $\boldsymbol{u}$ play no role in this computation. As the number of parents $\boldsymbol{U}$ grows, the number of different parent assignments grows exponentially. Therefore, the number of data instances that we expect to have for a single parent assignment shrinks exponentially. This phenomenon is called *data fragmentation*, since the data set is partitioned into a large number of small subsets. Intuitively, when we have a very small number of data instances from which we estimate a parameter, the estimates we get can be very noisy (this intuition is formalized in section 17.6), leading to *overfitting*. We are also more likely to get a large number of zeros in the distribution, which can lead to very poor performance. **Our inability to estimate parameters reliably as the dimensionality of the parent set grows is one of the key limiting factors in learning Bayesian networks from data.** This problem is even more severe when the variables can take on a large number of values, for example, in text applications.

data fragmentation

overfitting

classification

Box 17.A — Concept: Naive Bayes Classifier. *One of the basic tasks of learning is* classification. *In this task, our goal is build a classifier — a procedure that assigns instances into two or more categories, for example, deciding whether an email message is junk mail that should be discarded or a relevant message that should be presented to the user. In the usual setting, we are given a training example of instances from each category, where instances are represented by various features. In our email classification example, a message might be analyzed by multiple features: its length, the type of attachments it contains, the domain of the sender, whether that sender appears in the user's address book, whether a particular word appears in the subject, and so on.*

Bayesian classifier

One general approach to this problem, which is referred to as Bayesian classifier, *is to learn a probability distribution of the features of instances of each class. In the language of probabilistic models, we use the random variables $\boldsymbol{X}$ to represent the instance, and the random variable C to represent the category of the instance. The distribution $P(\boldsymbol{X} \mid C)$ is the probability of a particular combination of features given the category. Using Bayes rule, we have that*

$$P(C \mid \boldsymbol{X}) \propto P(C)P(\boldsymbol{X} \mid C).$$

Thus, if we have a good model of how instances of each category behave (that is, of $P(\boldsymbol{X} \mid C)$), we can combine it with our prior estimate for the frequency of each category (that is, $P(C)$) to estimate the posterior probability of each of the categories (that is, $P(C \mid \boldsymbol{X})$). We can then decide either to predict the most likely category or to perform a more complex decision based on the strength of likelihood of each option. For example, to reduce the number of erroneously removed messages, a junk-mail filter might remove email messages only when the probability that it is junk mail is higher than a strict threshold.

This Bayesian classification approach is quite intuitive. Loosely speaking, it states that to classify objects successfully, we need to recognize the characteristics of objects of each category. Then, we can classify a new object by considering whether it matches the characteristic of each of the classes. More formally, we use the language of probability to describe each category, assigning higher probability to objects that are typical for the category and low probability to ones that are not.

naive Bayes

The main hurdle in constructing a Bayesian classifier is the question of representation of the multivariate distribution $p(\boldsymbol{X} \mid C)$. The naive Bayes *classifier is one where we use the simplest representation we can think of. That is, we assume that each feature X_i is independent of all the other features given the class variable C. That is,*

$$P(\boldsymbol{X} \mid C) = \prod_i P(X_i \mid C).$$

Learning the distribution $P(C)P(\boldsymbol{X} \mid C)$ is thus reduced to learning the parameters in the naive Bayes structure, with the category variable C rendering all other features as conditionally independent of each other.

As can be expected, learning this classifier is a straightforward application of the parameter estimation that we consider in this chapter. Moreover, classifying new examples requires simple computation, evaluating $P(c) \prod_i P(x_i \mid c)$ for each category c.

Although this simple classifier is often dismissed as naive, in practice it is often surprisingly effective. From a training perspective, this classifier is quite robust, since in most applications, even with relatively few training examples, we can learn the parameters of conditional distribution

$P(X_i \mid C)$. *However, one might argue that robust learning does not compensate for oversimplified independence assumption. Indeed, the strong independence assumption usually results in poor representation of the distribution of instances. However,* **errors in estimating the probability of an instance do not necessarily lead to classification errors. For classification, we are interested in the relative size of the conditional distribution of the instances given different categories. The ranking of different labels may not be that sensitive to errors in estimating the actual probability of the instance.** *Empirically, one often finds that the naive Bayes classifier correctly classifies an example to the right category, yet its posterior probability is very skewed and quite far from the correct distribution.*

In practice, the naive Bayes classifier is often a good baseline classifier to try before considering more complex solutions. It is easy to implement, it is robust, and it can handle different choices of descriptions of instances (for example, box 17.E).

17.2.4 Gaussian Bayesian Networks ⋆

Our discussion until now has focused on learning discrete-state Bayesian networks with multinomial parameters. However, the concepts we have developed in this section carry through to a wide variety of other types of Bayesian networks. In particular, the global decomposition properties we proved for a Bayesian network apply, without any change, to any other type of CPD. That is, if the data are complete, the learning problem reduces to a set of local learning problems, one for each variable. The main difference is in applying the maximum likelihood estimation process to a CPD of a different type: how we define the sufficient statistics, and how we compute the maximum likelihood estimate from them. In this section, we demonstrate how MLE principles can be applied in the setting of linear Gaussian Bayesian networks. In section 17.2.5 we provide a general procedure for CPDs in the exponential family.

Consider a variable X with parents $\boldsymbol{U} = \{U_1, \ldots, U_k\}$ with a linear Gaussian CPD:

$$P(X \mid \boldsymbol{u}) = \mathcal{N}\left(\beta_0 + \beta_1 u_1 + \ldots, \beta_k u_k; \sigma^2\right).$$

Our task is to learn the parameters $\boldsymbol{\theta}_{X|\boldsymbol{U}} = \langle \beta_0, \ldots, \beta_k, \sigma \rangle$. To find the MLE values of these parameters, we need to differentiate the likelihood and solve the equations that define a stationary point. As usual, it will be easier to work with the log-likelihood function. Using the definition of the Gaussian distribution, we have that

$$\begin{aligned}\ell_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D}) &= \log L_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D}) \\ &= \sum_m \left[-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\frac{1}{\sigma^2}\left(\beta_0 + \beta_1 u_1[m] + \ldots + \beta_k u_k[m] - x[m]\right)^2 \right].\end{aligned}$$

We start by considering the gradient of the log-likelihood with respect to β_0:

$$\begin{aligned}\frac{\partial}{\partial \beta_0}\ell_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D}) &= \sum_m -\frac{1}{\sigma^2}\left(\beta_0 + \beta_1 u_1[m] + \ldots + \beta_k u_k[m] - x[m]\right) \\ &= -\frac{1}{\sigma^2}\left(M\beta_0 + \beta_1 \sum_m u_1[m] + \ldots + \beta_k \sum_m u_k[m] - \sum_m x[m]\right).\end{aligned}$$

Equating this gradient to 0, and multiplying both sides with $\frac{\sigma^2}{M}$, we get the equation

$$\frac{1}{M}\sum_m x[m] = \beta_0 + \beta_1 \frac{1}{M}\sum_m u_1[m] + \ldots + \beta_k \frac{1}{M}\sum_m u_k[m].$$

Each of the terms is the average value of one of the variables in the data. We use the notation

$$\boldsymbol{E}_{\mathcal{D}}[X] = \frac{1}{M}\sum_m x[m]$$

to denote this expectation. Using this notation, we see that we get the following equation:

$$\boldsymbol{E}_{\mathcal{D}}[X] = \beta_0 + \beta_1 \boldsymbol{E}_{\mathcal{D}}[U_1] + \ldots + \beta_k \boldsymbol{E}_{\mathcal{D}}[U_k]. \tag{17.6}$$

Recall that theorem 7.3 specifies the mean of a linear Gaussian variable X in terms of the means of its parents $U_1, \ldots, U_k$, using an expression that has precisely this form. Thus, equation (17.6) tells us that the MLE parameters should be such that the mean of X in the data is consistent with the predicted mean of X according to the parameters.

Next, consider the gradient with respect to one of the parameters β_i. Using similar arithmetic manipulations, we see that the equation $0 = \frac{\partial}{\partial \beta_i}\ell_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D})$ can be formulated as:

$$\boldsymbol{E}_{\mathcal{D}}[X \cdot U_i] = \beta_0 \boldsymbol{E}_{\mathcal{D}}[U_i] + \beta_1 \boldsymbol{E}_{\mathcal{D}}[U_1 \cdot U_i] + \ldots + \beta_k \boldsymbol{E}_{\mathcal{D}}[U_k \cdot U_i]. \tag{17.7}$$

At this stage, we have $k+1$ linear equations with $k+1$ unknowns, and we can use standard linear algebra techniques for solving for the value of $\beta_0, \beta_1, \ldots, \beta_k$. We can get additional intuition, however, by doing additional manipulation of equation (17.7). Recall that the covariance $\boldsymbol{Cov}[X;Y] = \boldsymbol{E}[X \cdot Y] - \boldsymbol{E}[X] \cdot \boldsymbol{E}[Y]$. Thus, if we subtract $\boldsymbol{E}_{\mathcal{D}}[X] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i]$ from the left-hand side of equation (17.7), we would get the empirical covariance of X and U_i. Using equation (17.6), we have that this term can also be written as:

$$\boldsymbol{E}_{\mathcal{D}}[X] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i] = \beta_0 \boldsymbol{E}_{\mathcal{D}}[U_i] + \beta_1 \boldsymbol{E}_{\mathcal{D}}[U_1] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i] + \ldots + \beta_k \boldsymbol{E}_{\mathcal{D}}[U_k] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i].$$

Subtracting this equation from equation (17.7), we get:

$$\begin{aligned}\boldsymbol{E}_{\mathcal{D}}[X \cdot U_i] - \boldsymbol{E}_{\mathcal{D}}[X] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i] \quad &= \quad \beta_1 \left(\boldsymbol{E}_{\mathcal{D}}[U_1 \cdot U_i] - \boldsymbol{E}_{\mathcal{D}}[U_1] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i]\right) + \ldots + \\ &\qquad \beta_k \left(\boldsymbol{E}_{\mathcal{D}}[U_k \cdot U_i] - \boldsymbol{E}_{\mathcal{D}}[U_k] \cdot \boldsymbol{E}_{\mathcal{D}}[U_i]\right).\end{aligned}$$

Using $\boldsymbol{Cov}_{\mathcal{D}}[X;U_i]$ to denote the observed covariance of X and U_i in the data, we get:

$$\boldsymbol{Cov}_{\mathcal{D}}[X;U_i] = \beta_1 \boldsymbol{Cov}_{\mathcal{D}}[U_1;U_i] + \ldots + \beta_k \boldsymbol{Cov}_{\mathcal{D}}[U_k;U_i].$$

In other words, the observed covariance of X with U_i should be the one predicted by theorem 7.3 given the parameters and the observed covariances between the parents of X.

Finally, we need to find the value of the σ^2 parameter. Taking the derivative of the likelihood and equating to 0, we get an equation that, after suitable reformulation, can be written as

$$\sigma^2 = \boldsymbol{Cov}_{\mathcal{D}}[X;X] - \sum_i \sum_j \beta_i \beta_j \boldsymbol{Cov}_{\mathcal{D}}[U_i;U_j] \tag{17.8}$$

(see exercise 17.4). Again, we see that the MLE estimate has to match the constraints implied by theorem 7.3.

The global picture that emerges is as follows. To estimate $P(X \mid \boldsymbol{U})$, we estimate the means of X and $\boldsymbol{U}$ and covariance matrix of $\{X\} \cup \boldsymbol{U}$ from the data. The vector of means and covariance matrix defines a joint Gaussian distribution over $\{X\} \cup \boldsymbol{U}$. (In fact, this is the MLE estimate of the joint Gaussian; see exercise 17.5.) We then solve for the (unique) linear Gaussian that matches the joint Gaussian with these parameters. For this purpose, we can use the formulas provided by theorem 7.4. While these equations seem somewhat complex, they are merely describing the solution to a system of linear equations.

This discussion also identifies the sufficient statistics we need to collect to estimate linear Gaussians. These are the univariate terms of the form $\sum_m x[m]$ and $\sum_m u_i[m]$, and the interaction terms of the form $\sum_m x[m] \cdot u_i[m]$ and $\sum_m u_i[m] \cdot u_j[m]$. From these, we can estimate the mean and covariance matrix of the joint distribution.

Box 17.B — Concept: Nonparametric Models. *The discussion in this chapter has focused on estimating parameters for specific parametric models of CPDs: multinomials and linear Gaussians. However, a theory of maximum likelihood and Bayesian estimation exists for a wide variety of other parametric models. Moreover, in recent years, there has been a growing interest in the use of* nonparametric Bayesian estimation *methods, where a (conditional) distribution is not defined to be in some particular parametric class with a fixed number of parameters, but rather the complexity of the representation is allowed to grow as we get more data instances. In the case of discrete variables, any CPD can be described as a table, albeit perhaps a very large one; thus a nonparametric method is less essential (although see section 19.5.2.2 for a very useful example of a nonparametric method in the discrete case). In the case of continuous variables, we do not have a "universal" parametric distribution. While Gaussians are often the default, many distributions are not well fit by them, and it is often difficult to determine which parametric family (if any) will be appropriate for a given variable. In such cases, nonparametric methods offer a useful substitute. In such methods, we use the data points themselves as the basis for a probability distribution. Many nonparametric methods have been developed; we describe one simple variant that serves to illustrate this type of approach.*

nonparametric Bayesian estimation

Suppose we want to learn the distribution $P(X \mid \boldsymbol{U})$ from data. A reasonable assumption is that the CPD is smooth. Thus, if we observe $x, \boldsymbol{u}$ in a training sample, it should increase the probability of seeing similar values of X for similar values of $\boldsymbol{U}$. More precisely, we increase the density of $p(X = x + \epsilon \mid \boldsymbol{U} = \boldsymbol{u} + \delta)$ for small values of ϵ and δ.

kernel density estimation

One simple approach that captures this intuition is the use of kernel density estimation *(also known as* Parzen windows*). The idea is fairly simple: given the data $\mathcal{D}$, we estimate a "local" joint density $\tilde{p}_X(X, \boldsymbol{U})$ by spreading out density around each example $x[m], \boldsymbol{u}[m]$. Formally, we write*

$$\tilde{p}_X(x, \boldsymbol{u}) = \frac{1}{M} \sum_m K(x, \boldsymbol{u}; x[m], \boldsymbol{u}[m], \alpha),$$

where K is a kernel density function *and α is a parameter (or vector of parameters) controlling K. A common choice of kernel is a simple round Gaussian distribution with radius α around $x[m], \boldsymbol{u}[m]$:*

$$K(x, \boldsymbol{u}; x[m], \boldsymbol{u}[m], \alpha) = \mathcal{N}\left(\begin{pmatrix} x[m] \\ \boldsymbol{u}[m] \end{pmatrix}; \alpha^2 I \right),$$

where I is the identity matrix and α is the width of the window. Of course, many other choices for kernel function are possible; in fact, if K defines a probability measure (nonnegative and integrates to 1), then $\tilde{p}_X(x, \mathbf{u})$ is also a probability measure. Usually we choose kernel functions that are local, in that they put most of the mass in the vicinity of their argument. For such kernels, the resulting density $\tilde{p}_X(x, \mathbf{u})$ will have high mass in regions where we have seen many data instances $(x[m], \mathbf{u}[m])$ and low mass in regions where we have seen none.

We can now reformulate this local joint distribution to produce a conditional distribution:

$$p(x \mid \mathbf{u}) = \frac{\sum_m K(x, \mathbf{u}; x[m], \mathbf{u}[m], \alpha)}{\sum_m K(\mathbf{u}; \mathbf{u}[m], \alpha)}$$

where $K(\mathbf{u}; \mathbf{u}[m], \alpha)$ is $K(x, \mathbf{u}; x[m], \mathbf{u}[m], \alpha)$ marginalized over x.

Note that this learning procedure estimates virtually no parameters: the CPD is derived directly from the training instances. The only free parameter is α, which is the width of the window. Importantly, this parameter cannot be estimated using maximum likelihood: The α that maximizes the likelihood of the training set is $\alpha = 0$, which gives maximum density to the training instances themselves. This, of course, will simply memorize the training instances without any generalization. Thus, this parameter is generally selected using cross-validation.

The learned CPD here is essentially the list of training instances, which has both advantages and disadvantages. On the positive side, the estimates are very flexible and tailor themselves to the observations; indeed, as we get more training data, we can produce arbitrarily expressive representations of our joint density. On the negative side, there is no "compression" of the original data, which has both computational and statistical ramifications. Computationally, when there are many training samples the learned CPDs can become unwieldy. Statistically, this learning procedure makes no attempt to generalize beyond the data instances that we have seen. In high-dimensional spaces with limited data, most points in the space will be "far" from data instances, and therefore the estimated density will tend to be quite poor in most parts of the space. Thus, this approach is primarily useful in cases where we have a large number of training instances relative to the dimension of the space.

Finally, while these approaches help us avoid parametric assumptions on the learning side, we are left with the question of how to avoid them on the inference side. As we saw, most inference procedures are geared to working with parametric representations, mostly Gaussians. Thus, when performing inference with nonparametric CPDs, we must generally either use parametric approximations, or resort to sampling.

17.2.5 Maximum Likelihood Estimation as M-Projection ⋆

The MLE principle is a general one, in that it gives a recipe how to construct estimators for different statistical models (for example, multinomials and Gaussians). As we have seen, for simple examples the resulting estimators are quite intuitive. However, the same principle can be applied in a much broader range of parametric models. Indeed, as we now show, we have already discussed the framework that forms the basis for this generalization.

In section 8.5, we defined the notion of *projection*: finding the distribution, within a specified class, that is closest to a given target distribution. Parameter estimation is similar in the sense

that we select a distribution from a given class — all of those that can be described by the model — that is "closest" to our data. Indeed, we can show that maximum likelihood estimation aims to find the distribution that is "closest" to the empirical distribution $\hat{P}_{\mathcal{D}}$ (see equation (16.4)).

We start by rewriting the likelihood function in terms of the empirical distribution.

Proposition 17.2 *Let $\mathcal{D}$ be a data set, then*

$$\log L(\boldsymbol{\theta} : \mathcal{D}) = M \cdot \boldsymbol{E}_{\hat{P}_{\mathcal{D}}}[\log P(\mathcal{X} : \boldsymbol{\theta})].$$

PROOF We rewrite the likelihood by combining all identical instances in our training set and then writing the likelihood in terms of the empirical probability of each entry in our joint distribution:

$$\begin{aligned}
\log L(\boldsymbol{\theta} : \mathcal{D}) &= \sum_m \log P(\xi[m] : \boldsymbol{\theta}) \\
&= \sum_{\xi} \left[\sum_m \boldsymbol{I}\{\xi[m] = \xi\} \right] \log P(\xi : \boldsymbol{\theta}) \\
&= \sum_{\xi} M \cdot \hat{P}_{\mathcal{D}}(\xi) \log P(\xi : \boldsymbol{\theta}) \\
&= M \cdot \boldsymbol{E}_{\hat{P}_{\mathcal{D}}}[\log P(\mathcal{X} : \boldsymbol{\theta})].
\end{aligned}$$

■

We can now apply proposition 16.1 to the empirical distribution to conclude that

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = M \left(\boldsymbol{H}_{\hat{P}_{\mathcal{D}}}(\mathcal{X}) - \boldsymbol{D}(\hat{P}_{\mathcal{D}}(\mathcal{X}) \| P(\mathcal{X} : \boldsymbol{\theta})) \right). \tag{17.9}$$

From this result, we immediately derive the following relationship between MLE and M-projections.

Theorem 17.1 *The MLE $\hat{\boldsymbol{\theta}}$ in a parametric family relative to a data set $\mathcal{D}$ is the M-projection of $\hat{P}_{\mathcal{D}}$ onto the parametric family*

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \boldsymbol{D}(\hat{P}_{\mathcal{D}} \| P_{\boldsymbol{\theta}}).$$

We see that MLE finds the distribution $P(\mathcal{X} : \boldsymbol{\theta})$ that is the M-projection of $\hat{P}_{\mathcal{D}}$ onto the set of distributions representable in our parametric family.

This result allows us to call upon our detailed analysis of M-projections in order to generalize MLE to other parametric classes in the exponential family. In particular, in section 8.5.2, we discussed the general notion of sufficient statistics and showed that the M-projection of a distribution P into a class of distributions $\mathcal{Q}$ was defined by the parameters $\boldsymbol{\theta}$ such that $\boldsymbol{E}_{Q_{\boldsymbol{\theta}}}[\tau(\mathcal{X})] = \boldsymbol{E}_P[\tau(\mathcal{X})]$. In our setting, we seek the parameters $\boldsymbol{\theta}$ whose expected sufficient statistics match those in $\hat{P}_{\mathcal{D}}$, that is, the sufficient statistics in $\mathcal{D}$.

If our CPDs are in an exponential family where the mapping *ess* from parameters to sufficient statistics is invertible, we can simply take the sufficient statistic vector from $\hat{P}_{\mathcal{D}}$, and invert this mapping to produce the MLE. Indeed, this process is precisely the one that gave rise to our MLE for multinomials and for linear Gaussians, as described earlier. However, the same process can be applied to many other classes of distributions in the exponential family.

This analysis provides us with a notion of sufficient statistics $\tau(\mathcal{X})$ and a clearly defined path to deriving MLE parameters for any distribution in the exponential family. Somewhat more surprisingly, it turns out that a parametric family has a sufficient statistic *only if* it is in the exponential family.

17.3 Bayesian Parameter Estimation

17.3.1 The Thumbtack Example Revisited

Although the MLE approach seems plausible, it can be overly simplistic in many cases. Assume again that we perform the thumbtack experiment and get 3 heads out of 10. It may be quite reasonable to conclude that the parameter θ is 0.3. But what if we do the same experiment with a standard coin, and we also get 3 heads? We would be much less likely to jump to the conclusion that the parameter of the coin is 0.3. Why? Because we have a lot more experience with tossing coins, so we have a lot more *prior knowledge* about their behavior. Note that we do not want our prior knowledge to be an absolute guide, but rather a reasonable starting assumption that allows us to counterbalance our current set of 10 tosses, under the assumption that they may not be typical. However, if we observe 1,000,000 tosses of the coin, of which 300,000 came out heads, then we may be more willing to conclude that this is a trick coin, one whose parameter is closer to 0.3.

Maximum likelihood allows us to make neither of these distinctions: between a thumbtack and a coin, and between 10 tosses and 1,000,000 tosses of the coin. There is, however, another approach, the one recommended by Bayesian statistics.

17.3.1.1 Joint Probabilistic Model

In this approach, we encode our prior knowledge about θ with a probability distribution; this distribution represents how likely we are a priori to believe the different choices of parameters. Once we quantify our knowledge (or lack thereof) about possible values of θ, we can create a joint distribution over the parameter θ and the data cases that we are about to observe $X[1], \ldots, X[M]$. This joint distribution captures our assumptions about the experiment.

Let us reconsider these assumptions. Recall that we assumed that tosses are independent of each other. Note, however, that this assumption was made when θ was fixed. If we do not know θ, then the tosses are not marginally independent: Each toss tells us something about the parameter θ, and thereby about the probability of the next toss. However, once θ is known, we cannot learn about the outcome of one toss from observing the results of others. Thus, we assume that the tosses are *conditionally independent* given θ. We can describe these assumptions using the probabilistic model of figure 17.3.

Having determined the model structure, it remains to specify the local probability models in this network. We begin by considering the probability $P(X[m] \mid \theta)$. Clearly,

$$P(x[m] \mid \theta) = \begin{cases} \theta & \text{if } x[m] = x^1 \\ 1 - \theta & \text{if } x[m] = x^0. \end{cases}$$

Note that since we now treat θ as a random variable, we use the conditioning bar, instead of $P(x[m] : \theta)$.

prior parameter distribution

To finish the description of the joint distribution, we need to describe $P(\theta)$. This is our *prior distribution* over the value of θ. In our case, this is a continuous density over the interval $[0, 1]$. Before we discuss particular choices for this distribution, let us consider how we use it.

The network structure implies that the joint distribution of a particular data set and θ

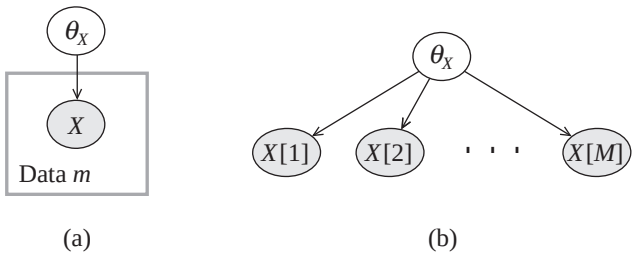


Figure 17.3 Meta-network for IID samples of a random variable X. (a) Plate model; (b) Ground Bayesian network.

factorizes as

$$\begin{aligned} P(x[1],\ldots,x[M],\theta) &= P(x[1],\ldots,x[M] \mid \theta)P(\theta) \\ &= P(\theta)\prod_{m=1}^{M} P(x[m] \mid \theta) \\ &= P(\theta)\theta^{M[1]}(1-\theta)^{M[0]}, \end{aligned}$$

where $M[1]$ is the number of heads in the data, and $M[0]$ is the number of tails. Note that the expression $P(x[1],\ldots,x[M] \mid \theta)$ is simply the likelihood function $L(\theta : \mathcal{D})$.

This network specifies a joint probability model over parameters and data. There are several ways in which we can use this network. Most obviously, we can take an observed data set $\mathcal{D}$ of M outcomes, and use it to instantiate the values of $x[1],\ldots,x[M]$; we can then compute the *posterior distribution* over θ:

posterior parameter distribution

$$P(\theta \mid x[1],\ldots,x[M]) = \frac{P(x[1],\ldots,x[M] \mid \theta)P(\theta)}{P(x[1],\ldots,x[M])}.$$

In this posterior, the first term in the numerator is the likelihood, the second is the prior over parameters, and the denominator is a normalizing factor that we will not expand on right now. We see that the posterior is (proportional to) a product of the likelihood and the prior. This product is normalized so that it will be a proper density function. In fact, if the prior is a uniform distribution (that is, $P(\theta) = 1$ for all $\theta \in [0,1]$), then the posterior is just the normalized likelihood function.

17.3.1.2 Prediction

If we do use a uniform prior, what then is the difference between the Bayesian approach and the MLE approach of the previous section? The main philosophical difference is in the use of the posterior. Instead of selecting from the posterior a single value for the parameter θ, we use it, in its entirety, for *predicting* the probability over the next toss.

To derive this prediction in a principled fashion, we introduce the value of the next coin toss $x[M+1]$ to our network. We can then compute the probability over $x[M+1]$ given the observations of the first M tosses. Note that, in this model, the parameter θ is unknown, and

we are considering all of its possible values. By reasoning over the possible values of θ and using the chain rule, we see that

$$\begin{aligned} P(x[M+1] \mid x[1], \ldots, x[M]) &= \\ &= \int P(x[M+1] \mid \theta, x[1], \ldots, x[M]) P(\theta \mid x[1], \ldots, x[M]) d\theta \\ &= \int P(x[M+1] \mid \theta) P(\theta \mid x[1], \ldots, x[M]) d\theta, \end{aligned}$$

where we use the conditional independencies implied by the meta-network to rewrite $P(x[M+1] \mid \theta, x[1], \ldots, x[M])$ as $P(x[M+1] \mid \theta)$. In other words, we are *integrating* our posterior over θ to predict the probability of heads for the next toss.

Let us go back to our thumbtack example. Assume that our prior is uniform over θ in the interval $[0, 1]$. Then $P(\theta \mid x[1], \ldots, x[M])$ is proportional to the likelihood $P(x[1], \ldots, x[M] \mid \theta) = \theta^{M[1]}(1-\theta)^{M[0]}$. Plugging this into the integral, we need to compute

$$P(X[M+1] = x^1 \mid x[1], \ldots, x[M]) = \frac{1}{P(x[1], \ldots, x[M])} \int \theta \cdot \theta^{M[1]}(1-\theta)^{M[0]} d\theta.$$

Doing all the math (see exercise 17.6), we get (for uniform priors)

$$P(X[M+1] = x^1 \mid x[1], \ldots, x[M]) = \frac{M[1]+1}{M[1]+M[0]+2}. \tag{17.10}$$

Bayesian estimator

This prediction, called the *Bayesian estimator*, is quite similar to the MLE prediction of equation (17.1), except that it adds one "imaginary" sample to each count. Clearly, as the number of samples grows, the Bayesian estimator and the MLE estimator converge to the same value.

Laplace's correction

The particular estimator that corresponds to a uniform prior is often referred to as *Laplace's correction*.

17.3.1.3 Priors

We now want to consider nonuniform priors. The challenge here is to pick a distribution over this continuous space that we can represent compactly (for example, using an analytic formula), and update efficiently as we get new data. For reasons that we will discuss, an appropriate prior in this case is the *Beta distribution*:

Beta distribution

Definition 17.3

Beta hyperparameters

A Beta distribution *is parameterized by two* hyperparameters α_1, α_0*, which are positive reals. The distribution is defined as follows:*

$$\theta \sim \text{Beta}(\alpha_1, \alpha_0) \text{ if } p(\theta) = \gamma \theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1}.$$

The constant γ is a normalizing constant, defined as follows:

$$\gamma = \frac{\Gamma(\alpha_1 + \alpha_0)}{\Gamma(\alpha_1)\Gamma(\alpha_0)},$$

Gamma function

where $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ *is the* Gamma function. ■

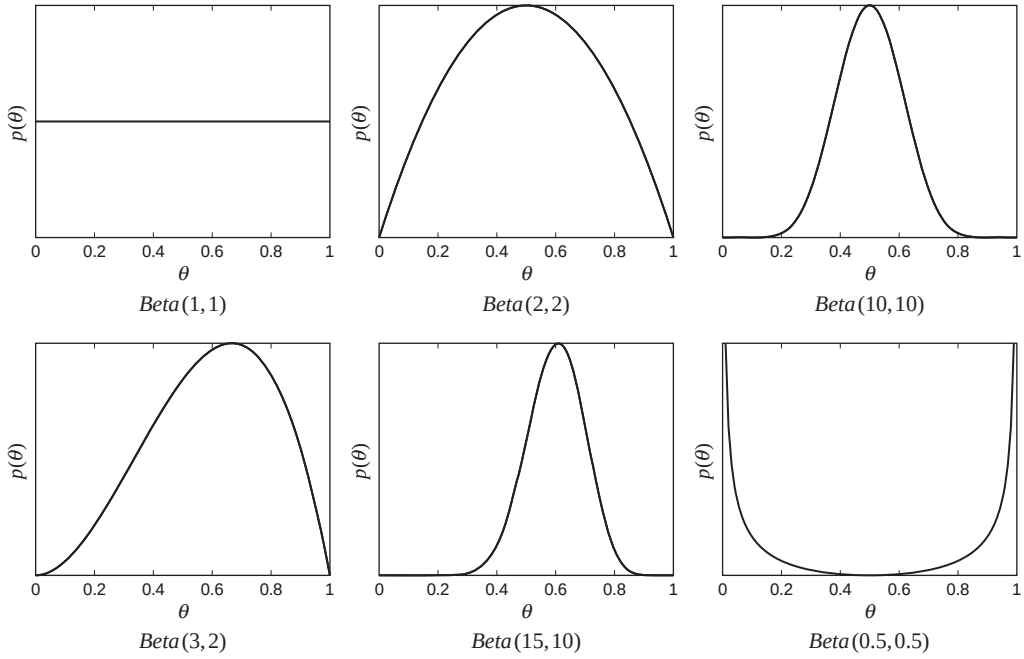


Figure 17.4 Examples of Beta distributions for different choices of hyperparameters

Intuitively, the hyperparameters α_1 and α_0 correspond to the number of imaginary heads and tails that we have "seen" before starting the experiment. Figure 17.4 shows Beta distributions for different values of α.

At first glance, the normalizing constant for the Beta distribution might seem somewhat obscure. However, the Gamma function is actually a very natural one: it is simply a continuous generalization of factorials. More precisely, it satisfies the properties $\Gamma(1) = 1$ and $\Gamma(x+1) = x\Gamma(x)$. As a consequence, we easily see that $\Gamma(n+1) = n!$ when n is an integer.

Beta distributions have properties that make them particularly useful for parameter estimation. Assume our distribution $P(\theta)$ is *Beta*(α_1, α_0), and consider a single coin toss X. Let us compute the *marginal probability* over X, based on $P(\theta)$. To compute the marginal probability, we need to integrate out θ; standard integration techniques can be used to show that:

$$\begin{aligned} P(X[1] = x^1) &= \int_0^1 P(X[1] = x^1 \mid \theta) \cdot P(\theta) d\theta \\ &= \int_0^1 \theta \cdot P(\theta) d\theta = \frac{\alpha_1}{\alpha_1 + \alpha_0}. \end{aligned}$$

This conclusion supports our intuition that the Beta prior indicates that we have seen α_1 (imaginary) heads α_0 (imaginary) tails.

Now, let us see what happens as we get more observations. Specifically, we observe $M[1]$ heads and $M[0]$ tails. It follows easily that:

$$\begin{aligned} P(\theta \mid x[1], \ldots, x[M]) &\propto P(x[1], \ldots, x[M] \mid \theta) P(\theta) \\ &\propto \theta^{M[1]}(1-\theta)^{M[0]} \cdot \theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1} \\ &= \theta^{\alpha_1 + M[1] - 1}(1-\theta)^{\alpha_0 + M[0] - 1}, \end{aligned}$$

which is precisely *Beta*$(\alpha_1 + M[1], \alpha_0 + M[0])$. This result illustrates a key property of the Beta distribution: If the prior is a Beta distribution, then the posterior distribution, that is, the prior conditioned on the evidence, is also a Beta distribution. In this case, we say that the Beta distribution is *conjugate* to the Bernoulli likelihood function (see definition 17.4).

conjugate prior

An immediate consequence is that we can compute the probabilities over the next toss:

$$P(X[M+1] = x^1 \mid x[1], \ldots, x[M]) = \frac{\alpha_1 + M[1]}{\alpha + M},$$

where $\alpha = \alpha_1 + \alpha_0$. In this case, our posterior Beta distribution tells us that we have seen $\alpha_1 + M[1]$ heads (imaginary and real) and $\alpha_0 + M[0]$ tails.

It is interesting to examine the effect of the prior on the probability over the next coin toss. For example, the prior *Beta*$(1, 1)$ is very different than *Beta*$(10, 10)$: Although both predict that the probability of heads in the first toss is 0.5, the second prior is more entrenched, and it requires more observations to deviate from the prediction 0.5. To see this, suppose we observe 3 heads in 10 tosses. Using the first prior, our estimate is $\frac{3+1}{10+2} = \frac{1}{3} \approx 0.33$. On the other hand, using the second prior, our estimate is $\frac{3+10}{10+20} = \frac{13}{30} \approx 0.43$. However, as we obtain more data, the effect of the prior diminishes. If we obtain $1,000$ tosses of which 300 are heads, the first prior gives us an estimate of $\frac{300+1}{1,000+2}$ and the second an estimate of $\frac{300+10}{1,000+20}$, both of which are very close to 0.3. Thus, **the Bayesian framework allows us to capture both of the relevant distinctions. The distinction between the thumbtack and the coin can be captured by the strength of the prior: for a coin, we might use $\alpha_1 = \alpha_0 = 100$, whereas for a thumbtack, we might use $\alpha_1 = \alpha_0 = 1$. The distinction between a few samples and many samples is captured by the peakedness of our posterior, which increases with the amount of data.**

17.3.2 Priors and Posteriors

We now turn to examine in more detail the Bayesian approach to dealing with unknown parameters. We start with a discussion of the general principle and deal with the case of Bayesian networks in the next section.

As before, we assume a general learning problem where we observe a training set $\mathcal{D}$ that contains M IID samples of a set of random variable $\mathcal{X}$ from an unknown distribution $P^*(\mathcal{X})$. We also assume that we have a parametric model $P(\xi \mid \boldsymbol{\theta})$ where we can choose parameters from a parameter space Θ.

Recall that the MLE approach attempts to find the parameters $\hat{\boldsymbol{\theta}}$ in Θ that are "best" given the data. The Bayesian approach, on the other hand, does not attempt to find such a *point estimate*. Instead, the underlying principle is that we should keep track of our *beliefs* about $\boldsymbol{\theta}$'s values, and use these beliefs for reaching conclusions. That is, we should quantify the subjective probability we assign to different values of $\boldsymbol{\theta}$ after we have seen the evidence. Note that, in representing

point estimate

such subjective probabilities, we now treat $\boldsymbol{\theta}$ as a random variable. Thus, the Bayesian approach requires that we use probabilities to describe our initial uncertainty about the parameters $\boldsymbol{\theta}$, and then use probabilistic reasoning (that is, Bayes rule) to take into account our observations.

To perform this task, we need to describe a joint distribution $P(\mathcal{D}, \boldsymbol{\theta})$ over the data and the parameters. We can easily write

$$P(\mathcal{D}, \boldsymbol{\theta}) = P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta}).$$

parameter prior

The first term is just the likelihood function we discussed earlier. The second term is the *prior distribution* over the possible values in Θ. This prior captures our initial uncertainty about the parameters. It can also capture our previous experience before starting the experiment. For example, if we study coin tossing, we might have prior experience that suggests that most coins are unbiased (or nearly unbiased).

parameter posterior

Once we have specified the likelihood function and the prior, we can use the data to derive the *posterior distribution* over the parameters. Since we have specified a joint distribution over all the quantities in question, the posterior is immediately derived by Bayes rule:

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})}.$$

marginal likelihood

The term $P(\mathcal{D})$ is the *marginal likelihood* of the data

$$P(\mathcal{D}) = \int_{\Theta} P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})d\boldsymbol{\theta},$$

that is, the integration of the likelihood over all possible parameter assignments. This is the a priori probability of seeing this particular data set given our prior beliefs.

As we saw, for some probabilistic models, the likelihood function can be compactly described by using sufficient statistics. Can we also compactly describe the posterior distribution? In general, this depends on the form of the prior. As we saw in the thumbtack example of section 17.1.1, we can sometimes find priors for which we have a description of the posterior.

As another example of the forms of priors and posteriors, let us examine the learning problem of example 17.3. Here we need to describe our uncertainty about the parameters of a multinomial distribution. The parameter space Θ is the space of all nonnegative vectors $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_K \rangle$ such that $\sum_k \theta_k = 1$. As we saw in example 17.3, the likelihood function in this model has the form:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_k \theta_k^{M[k]}.$$

Since the posterior is a product of the prior and the likelihood, it seems natural to require that the prior also have a form similar to the likelihood.

Dirichlet distribution

Dirichlet hyperparameters

One such prior is the *Dirichlet distribution*, which generalizes the Beta distribution we discussed earlier. A Dirichlet distribution is specified by a set of *hyperparameters* $\alpha_1, \ldots, \alpha_K$, so that

$$\boldsymbol{\theta} \sim Dirichlet(\alpha_1, \ldots, \alpha_K) \text{ if } P(\boldsymbol{\theta}) \propto \prod_k \theta_k^{\alpha_k - 1}.$$

Dirichlet posterior

We use α to denote $\sum_j \alpha_j$. If we use a Dirichlet prior, then the *posterior* is also Dirichlet:

Proposition 17.3 *If $P(\boldsymbol{\theta})$ is Dirichlet$(\alpha_1, \ldots, \alpha_K)$ then $P(\boldsymbol{\theta} \mid \mathcal{D})$ is Dirichlet$(\alpha_1 + M[1], \ldots, \alpha_K + M[K])$, where $M[k]$ is the number of occurrences of x^k.*

Priors such as the Dirichlet are useful, since they ensure that the posterior has a nice compact description. Moreover, this description uses the same representation as the prior. This phenomenon is a general one, and one that we strive to achieve, since it makes our computation and representation much easier.

Definition 17.4 conjugate prior

A family of priors $P(\boldsymbol{\theta} : \boldsymbol{\alpha})$ is conjugate *to a particular model $P(\xi \mid \boldsymbol{\theta})$ if for any possible data set $\mathcal{D}$ of IID samples from $P(\xi \mid \boldsymbol{\theta})$, and any choice of legal hyperparameters $\boldsymbol{\alpha}$ for the prior over $\boldsymbol{\theta}$, there are hyperparameters $\boldsymbol{\alpha}'$ that describe the posterior. That is,*

$$P(\boldsymbol{\theta} : \boldsymbol{\alpha}') \propto P(\mathcal{D} \mid \boldsymbol{\theta}) P(\boldsymbol{\theta} : \boldsymbol{\alpha}).$$

■

For example, Dirichlet priors are conjugate to the multinomial model. We note that this does not preclude the possibility of other families that are also conjugate to the same model. See exercise 17.7 for an example of such a prior for the multinomial model. We can find conjugate priors for other models as well. See exercise 17.8 and exercise 17.11 for the development of conjugate priors for the Gaussian distribution.

This discussion shows some examples where we can easily update our beliefs about $\boldsymbol{\theta}$ after observing a set of instances $\mathcal{D}$. This update process results in a posterior that combines our prior knowledge and our observations. What can we do with the posterior? We can use the posterior to determine properties of the model at hand. For example, to assess our beliefs that a coin we experimented with is biased toward heads, we might compute the posterior probability that $\theta > t$ for some threshold t, say 0.6.

Another use of the posterior is to predict the probability of future examples. Suppose that we are about to sample a new instance $\xi[M+1]$. Since we already have observations over previous instances, the *Bayesian estimator* is the posterior distribution over a new example:

Bayesian estimator

$$\begin{aligned} P(\xi[M+1] \mid \mathcal{D}) &= \int P(\xi[M+1] \mid \mathcal{D}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta} \\ &= \int P(\xi[M+1] \mid \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta} \\ &= \boldsymbol{E}_{P(\boldsymbol{\theta} \mid \mathcal{D})}[P(\xi[M+1] \mid \boldsymbol{\theta})], \end{aligned}$$

where, in the second step, we use the fact that instances are independent given $\boldsymbol{\theta}$. Thus, our prediction is the average over all parameters according to the posterior.

Let us examine prediction with the Dirichlet prior. We need to compute

$$P(x[M+1] = x^k \mid \mathcal{D}) = \boldsymbol{E}_{P(\boldsymbol{\theta} \mid \mathcal{D})}[\theta_k].$$

To compute the prediction on a new data case, we need to compute the expectation of particular parameters with respect for a Dirichlet distribution over $\boldsymbol{\theta}$.

Proposition 17.4 *Let $P(\boldsymbol{\theta})$ be a Dirichlet distribution with hyperparameters $\alpha_1, \ldots, \alpha_k$, and $\alpha = \sum_j \alpha_j$, then $E[\theta_k] = \frac{\alpha_k}{\alpha}$.*

Recall that our posterior is $Dirichlet(\alpha_1 + M[1], \ldots, \alpha_K + M[K])$ where $M[1], \ldots, M[K]$ are the sufficient statistics from the data. Hence, the prediction with Dirichlet priors is

$$P(x[M+1] = x^k \mid \mathcal{D}) = \frac{M[k] + \alpha_k}{M + \alpha}.$$

This prediction is similar to prediction with the MLE parameters. The only difference is that we added the hyperparameters to our counts when making the prediction. For this reason the Dirichlet hyperparameters are often called *pseudo-counts*. We can think of these as the number of times we have seen the different outcomes in our prior experience before conducting our current experiment.

pseudo-counts

The total α of the pseudo-counts reflects how confident we are in our prior, and is often called the *equivalent sample size*. Using α, we can rewrite the hyperparameters as $\alpha_k = \alpha\theta'_k$, where $\boldsymbol{\theta}' = \{\theta'_k : k = 1, \ldots, K\}$ is a distribution describing the *mean prediction* of our prior. We can see that the prior prediction (before observing any data) is simply $\boldsymbol{\theta}'$. Moreover, we can rewrite the prediction given the posterior as:

equivalent sample size

mean prediction

$$P(x[M+1] = x^k \mid \mathcal{D}) = \frac{\alpha}{M+\alpha}\theta'_k + \frac{M}{M+\alpha} \cdot \frac{M[k]}{M}. \tag{17.11}$$

That is, the prediction is a weighted average (convex combination) of the prior mean and the MLE estimate. The combination weights are determined by the relative magnitude of α — the confidence of the prior (or total weight of the pseudo-counts) — and M — the number of observed samples. We see that the Bayesian prediction converges to the MLE estimate when $M \to \infty$. Intuitively, when we have a very large training set the contribution of the prior is negligible, and the prediction will be dominated by the frequency of outcomes in the data. We also get convergence to the MLE estimate when $\alpha \to 0$, so that we have only a very weak prior. Note that the case where $\alpha = 0$ is not achievable: the normalization constant for the Dirichlet prior grows to infinity when the hyperparameters are close to 0. Thus, the prior with $\alpha = 0$ (that is, $\alpha_k = 0$ for all k) is not well defined. The prior with $\alpha = 0$ is often called a *improper prior*. The difference between the Bayesian estimate and the MLE estimate arises when M is not too large, and α is not close to 0. In these situations, the Bayesian estimate is "biased" toward the prior probability $\boldsymbol{\theta}'$.

improper prior

To gain some intuition for the interaction between these different factors, figure 17.5 shows the effect of the strength and means of the prior on our estimates. We can see that, as the amount of real data grows, our estimate converges to the true underlying distribution, regardless of the starting point. The convergence time grows both with the difference between the prior mean and the empirical mean, and with the strength of the prior. We also see that the Bayesian estimate is more stable than the MLE estimate, because with few instances, even single samples will change the MLE estimate dramatically.

Example 17.7

Suppose we are trying to estimate the parameter associated with a coin, and we observe one head and one tail. Our MLE estimate of θ_1 *is* $1/2 = 0.5$. *Now, if the next observation is a head, we will change our estimate to be* $2/3 \approx 0.66$. *On the other hand, if our next observation is a tail, we will change our estimate to* $1/3 \approx 0.33$. *In contrast, consider the Bayesian estimate with a Dirichlet prior with* $\alpha = 1$ *and* $\theta'_1 = 0.5$. *With this estimator, our original estimate is* $1.5/3 = 0.5$. *If we observe another head, we revise to* $2.5/4 = 0.625$, *and if observe another tail, we revise to*

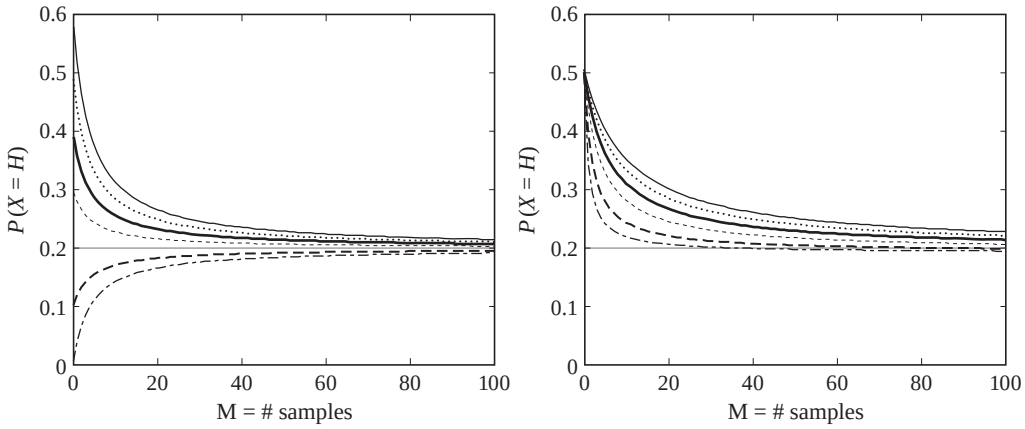


Figure 17.5 The effect of the strength and means of the Beta prior on our posterior estimates. Our data set is an idealized version of samples from a biased coin where the frequency of heads is 0.2: for a given data set size M, we assume that $\mathcal{D}$ contains $0.2M$ heads and $0.8M$ tails. The x axis represents the number of samples (M) in our data set $\mathcal{D}$, and the y axis the expected probability of heads according to the Bayesian estimate. (a) shows the effect of varying the prior means θ'_1, θ'_0, for a fixed prior strength α. (b) shows the effect of varying the prior strength for a fixed prior mean $\theta'_1 = \theta'_0 = 0.5$.

$1.5/4 = 0.375$. *We see that the estimate changes by slightly less after the update. If* α *is larger, then the smoothing is more aggressive. For example, when* $\alpha = 5$*, our estimate is* $4.5/8 = 0.5625$ *after observing a head, and* $3.5/8 = 0.4375$ *after observing a tail. We can also see this effect visually in figure 17.6, which shows our changing estimate for* $P(\theta_H)$ *as we observe a particular sequence of tosses.* ■

This smoothing effect results in more robust estimates when we do not have enough data to reach definite conclusions. If we have good prior knowledge, we revert to it. Alternatively, if we do not have prior knowledge, we can use a uniform prior that will keep our estimate from taking extreme values. In general, it is a bad idea to have extreme estimates (ones where some of the parameters are close to 0), since these might assign too small probability to new instances we later observe. In particular, as we already discussed, probability estimates that are actually 0 are dangerous, since no amount of evidence can change them. Thus, if we are unsure about our estimates, it is better to bias them away from extreme estimates. The MLE estimate, on the other hand, often assigns probability 0 to values that were not observed in the training data.

17.4 Bayesian Parameter Estimation in Bayesian Networks

We now turn to Bayesian estimation in the context of a Bayesian network. Recall that the Bayesian framework requires us to specify a joint distribution over the unknown parameters and the data instances. As in the single parameter case, we can understand the joint distribution over parameters and data as a Bayesian network.

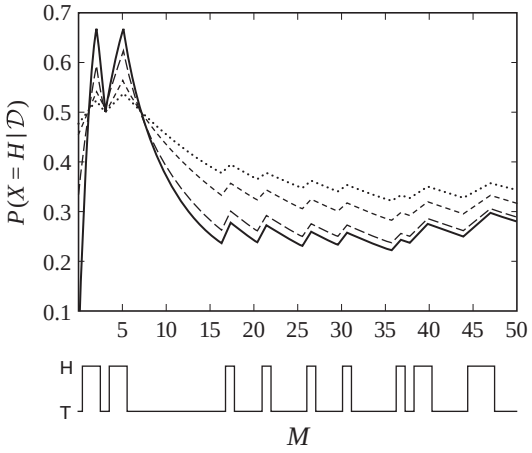


Figure 17.6 The effect of different priors on smoothing our parameter estimates. The graph shows the estimate of $P(X = H|\mathcal{D})$ (y-axis) after seeing different number of samples (x-axis). The graph below the x-axis shows the particular sequence of tosses. The solid line corresponds to the MLE estimate, and the remaining ones to Bayesian estimates with different strengths and uniform prior means. The large-dash line corresponds to *Beta*$(1, 1)$, the small-dash line to *Beta*$(5, 5)$, and the dotted line to *Beta*$(10, 10)$.

17.4.1 Parameter Independence and Global Decomposition

17.4.1.1 A Simple Example

Suppose we want to estimate parameters for a simple network with two variables X and Y so that X is the parent of Y. Our training data consist of observations $X[m], Y[m]$ for $m = 1, \ldots, M$. In addition, we have unknown parameter vectors $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_{Y|X}$. The dependencies between these variables are described in the network of figure 17.7. This is the *meta-network* that describes our learning setup.

meta-network

This Bayesian network structure immediately reveals several points. For example, as in our simple thumbtack example, the instances are independent given the unknown parameters. A simple examination of active trails shows that $X[m]$ and $Y[m]$ are d-separated from $X[m']$ and $Y[m']$ once we observe the parameter variables.

In addition, the network structure embodies the assumption that the priors for the individual parameters variables are a priori independent. That is, we believe that knowing the value of one parameter tells us nothing about another. More precisely, we define

Definition 17.5 global parameter independence

Let $\mathcal{G}$ be a Bayesian network structure with parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{X_1|\mathrm{Pa}_{X_1}}, \ldots, \boldsymbol{\theta}_{X_n|\mathrm{Pa}_{X_n}})$. A prior $P(\boldsymbol{\theta})$ is said to satisfy global parameter independence *if it has the form:*

$$P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}).$$

■

This assumption may not be suitable for all domains, and it should be considered with care.

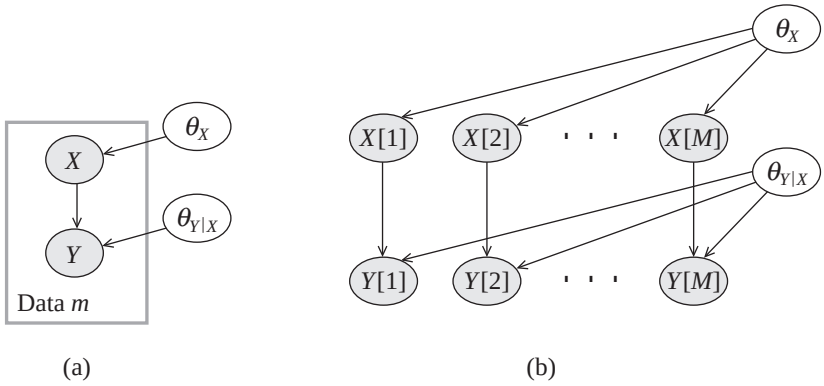


Figure 17.7 Meta-network for IID samples from a network $X \rightarrow Y$ with global parameter independence. (a) Plate model; (b) Ground Bayesian network.

Example 17.8

Consider an extension of our student *example, where our student takes multiple classes. For each class, we want to learn the distribution of Grade given the student's Intelligence and the course Difficulty. For classes taught by the same instructor, we might believe that the grade distribution is the same; for example, if two classes are both difficult, and the student is intelligent, his probability of getting an A is the same in both. However, under the global parameter independence assumption, these are two different random variables, and hence their parameters are independent.* ■

Thus, although we use the global parameter independence in much of our discussion, it is not always appropriate, and we relax it in some of our later discussion (such as section 17.5 and section 18.6.2).

If we accept global parameter independence, we can draw an important conclusion. Complete data d-separates the parameters for different CPDs. For example, if $x[m]$ and $y[m]$ are observed for all m, then $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_{Y|X}$ are d-separated. To see this, note that any path between the two has the form

$$\boldsymbol{\theta}_X \rightarrow X[m] \rightarrow Y[m] \leftarrow \boldsymbol{\theta}_{Y|X},$$

so that the observation of $x[m]$ blocks the path. Thus, if these two parameter variables are independent a priori, they are also independent a posteriori. Using the definition of conditional independence, we conclude that

$$P(\boldsymbol{\theta}_X, \boldsymbol{\theta}_{Y|X} \mid \mathcal{D}) = P(\boldsymbol{\theta}_X \mid \mathcal{D}) P(\boldsymbol{\theta}_{Y|X} \mid \mathcal{D}).$$

This decomposition has immediate practical ramifications. Given the data set $\mathcal{D}$, we can determine the posterior over $\boldsymbol{\theta}_X$ independently of the posterior over $\boldsymbol{\theta}_{Y|X}$. Once we can solve each problem separately, we can combine the results. This is the analogous result to the likelihood decomposition for MLE estimation of section 17.2.2. In the Bayesian setting this property has additional importance. It tells us the posterior can be represented in a compact factorized form.

17.4.1.2 General Networks

We can generalize this conclusion to the general case of Bayesian network learning. Suppose we are given a network structure $\mathcal{G}$ with parameters $\boldsymbol{\theta}$. In the Bayesian framework, we need to specify a prior $P(\boldsymbol{\theta})$ over all possible parameterizations of the network. The posterior distribution over parameters given the data samples $\mathcal{D}$ is simply

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})}.$$

The term $P(\boldsymbol{\theta})$ is our prior distribution, $P(\mathcal{D} \mid \boldsymbol{\theta})$ is the probability of the data given a particular parameter settings, which is simply the likelihood function. Finally, $P(\mathcal{D})$ is the normalizing constant. As we discussed, this term is called the *marginal likelihood*; it will play an important role in the next chapter. For now, however, we can ignore it, since it does not depend on $\boldsymbol{\theta}$ and only serves to normalize the posterior.

marginal likelihood

As we discussed in section 17.2, we can decompose the likelihood into local likelihoods:

$$P(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_i L_i(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} : \mathcal{D}).$$

Moreover, if we assume that we have global parameter independence, then

$$P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}).$$

Combining these two decompositions, we see that

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{1}{P(\mathcal{D})} \prod_i \left[L_i(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} : \mathcal{D}) P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}) \right].$$

Now each subset $\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}$ of $\boldsymbol{\theta}$ appears in just one term in the product. Thus, we have that the posterior can be represented as a product of local terms.

Proposition 17.5 *Let $\mathcal{D}$ be a complete data set for $\mathcal{X}$, let $\mathcal{G}$ be a network structure over these variables. If $P(\boldsymbol{\theta})$ satisfies global parameter independence, then*

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_i P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} \mid \mathcal{D}).$$

The proof of this property follows from the steps we discussed. It can also be derived directly from the structure of the meta-Bayesian network (as in the network of figure 17.7).

17.4.1.3 Prediction

This decomposition of the posterior allows us to simplify various tasks. For example, suppose that, in our simple two-variable network, we want to compute the probability of another instance $x[M+1], y[M+1]$ based on our previous observations $x[1], y[1], \ldots, x[M], y[M]$. According to the structure of our meta-network, we need to sum out (or more precisely integrate out) the unknown parameter variables

$$P(x[M+1], y[M+1] \mid \mathcal{D}) = \int P(x[M+1], y[M+1] \mid \mathcal{D}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta},$$

where the integration is over all legal parameter values. Since $\boldsymbol{\theta}$ d-separates instances from each other, we have that

$$\begin{aligned} & P(x[M+1], y[M+1] \mid \mathcal{D}, \boldsymbol{\theta}) \\ &= P(x[M+1], y[M+1] \mid \boldsymbol{\theta}) \\ &= P(x[M+1] \mid \boldsymbol{\theta}_X) P(y[M+1] \mid x[M+1], \boldsymbol{\theta}_{Y|X}). \end{aligned}$$

Moreover, as we just saw, the posterior probability also decomposes into a product. Thus,

$$\begin{aligned} & P(x[M+1], y[M+1] \mid \mathcal{D}) \\ &= \int \int P(x[M+1] \mid \boldsymbol{\theta}_X) P(y[M+1] \mid x[M+1], \boldsymbol{\theta}_{Y|X}) \\ &\qquad P(\boldsymbol{\theta}_X \mid \mathcal{D}) P(\boldsymbol{\theta}_{Y|X} \mid \mathcal{D}) d\boldsymbol{\theta}_X d\boldsymbol{\theta}_{Y|X} \\ &= \left(\int P(x[M+1] \mid \boldsymbol{\theta}_X) P(\boldsymbol{\theta}_X \mid \mathcal{D}) d\boldsymbol{\theta}_X \right) \\ &\qquad \left(\int P(y[M+1] \mid x[M+1], \boldsymbol{\theta}_{Y|X}) P(\boldsymbol{\theta}_{Y|X} \mid \mathcal{D}) d\boldsymbol{\theta}_{Y|X} \right). \end{aligned}$$

In the second step, we use the fact that the double integral of two unrelated functions is the product of the integrals. That is:

$$\int \int f(x) g(y) dx dy = \left(\int f(x) dx \right) \left(\int g(y) dy \right).$$

Thus, we can solve the prediction problem for the two variables X and Y separately.

The same line of reasoning easily applies to the general case, and thus we can see that, in the setting of proposition 17.5, we have

$$\begin{aligned} & P(X_1[M+1], \ldots, X_n[M+1] \mid \mathcal{D}) = \\ & \qquad \prod_i \int P(X_i[M+1] \mid \mathrm{Pa}_{X_i}[M+1], \boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}) P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} \mid \mathcal{D}) d\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}. \end{aligned} \tag{17.12}$$

We see that we can solve the prediction problem for each CPD independently and then combine the results.

We stress that the discussion so far was based on the assumption that the priors over parameters for different CPDs are independent. We see that, when learning from complete data, this assumption alone suffices to get a decomposition of the learning problem to several "local" problems, each one involving one CPD.

At this stage it might seem that the Bayesian framework introduces new complications that did not appear in the MLE setup. Note, however, that in deriving the MLE decomposition, we used the property that we can choose parameters for one CPD independently of the others. Thus, we implicitly made a similar assumption to get decomposition. The Bayesian treatment forces us to make such assumptions explicit, allowing us to more carefully evaluate their validity. We view this as a benefit of the Bayesian framework.

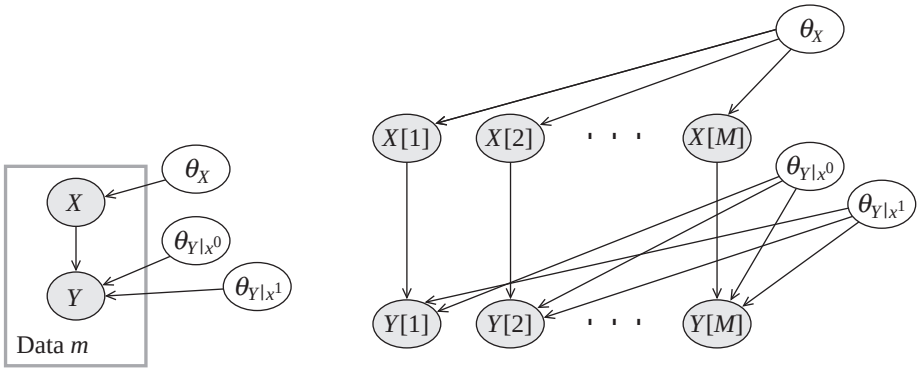


Figure 17.8 Meta-network for IID samples from a network $X \rightarrow Y$ with local parameter independence. (a) Plate model. (b) Ground Bayesian network.

17.4.2 Local Decomposition

Based on the preceding discussion, we now need to solve localized Bayesian estimation problems to get a global Bayesian solution. We now examine this localized estimation task for table-CPDs. The case for tree-CPDs is treated in section 17.5.2.

Consider, for example, the learning setting described in figure 17.7, where we take both X and Y to be binary. As we have seen, we need to represent the posterior $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_{Y|X}$ given the data. We already know how to deal with the posterior over $\boldsymbol{\theta}_X$. If we use a Dirichlet prior over $\boldsymbol{\theta}_X$, then the posterior $P(\boldsymbol{\theta}_X \mid x[1], \ldots, x[M])$ is also represented as a Dirichlet distribution.

A less obvious question is how to deal with the posterior over $\boldsymbol{\theta}_{Y|X}$. If we are learning table-CPDs, this parameter vector contains four parameters $\theta_{y^0|x^0}, \ldots, \theta_{y^1|x^1}$. In our discussion of maximum likelihood estimation, we saw how the local likelihood over these parameters can be further decomposed into two terms, one over the parameters $\boldsymbol{\theta}_{Y|x^0}$ and one over the parameters $\boldsymbol{\theta}_{Y|x^1}$. Do we have a similar phenomenon in the Bayesian setting?

We start with the prior over $\boldsymbol{\theta}_{Y|X}$. One obvious choice is a Dirichlet prior over $\boldsymbol{\theta}_{Y|x^1}$ and another over $\boldsymbol{\theta}_{Y|x^0}$. More precisely, we have

$$P(\boldsymbol{\theta}_{Y|X}) = P(\boldsymbol{\theta}_{Y|x^1})P(\boldsymbol{\theta}_{Y|x^0}),$$

where each of the terms on the right is a Dirichlet prior. Thus, in this case, we assume that the two groups of parameters are independent a priori.

This independence assumption, in effect, allows us to replace the node $\boldsymbol{\theta}_{Y|X}$ in figure 17.7 with two nodes, $\boldsymbol{\theta}_{Y|x^1}$ and $\boldsymbol{\theta}_{Y|x^0}$ that are both roots (see figure 17.8). What can we say about the posterior distribution of these parameter groups? At first, it seems that the two are dependent on each other given the data. Given an observation of $y[m]$, the path

$$\boldsymbol{\theta}_{Y|x^0} \rightarrow Y[m] \leftarrow \boldsymbol{\theta}_{Y|x^1}$$

is active (since we observe the sink of a v-structure), and thus the two parameters are not

d-separated.

This, however, is not the end of the story. We get more insight if We examine how $y[m]$ depends on the two parameters. Clearly,

$$P(y[m] = y \mid x[m], \boldsymbol{\theta}_{Y|x^0}, \boldsymbol{\theta}_{Y|x^1}) = \begin{cases} \boldsymbol{\theta}_{y|x^0} & \text{if } x[m] = x^0 \\ \boldsymbol{\theta}_{y|x^1} & \text{if } x[m] = x^1. \end{cases}$$

We see that $y[m]$ does not depends on the value of $\boldsymbol{\theta}_{Y|x^0}$ when $x[m] = x^1$. This example is an instance of the same type of context specific independence that we discussed in example 3.7. As discussed in section 5.3, we can perform a more refined form of d-separation test in such a situation by removing arcs that are ruled inactive in particular contexts. For the CPD of $y[m]$, we see that once we observe the value of $x[m]$, one of the two arcs into $y[m]$ is inactive. If $x[m] = x^0$, then the arc $\boldsymbol{\theta}_{Y|x^1} \rightarrow y[m]$ is inactive, and if $x[m] = x^1$, then $\boldsymbol{\theta}_{Y|x^0} \rightarrow y[m]$ is inactive. In either case, the v-structure $\boldsymbol{\theta}_{Y|x^0} \rightarrow y[m] \leftarrow \boldsymbol{\theta}_{Y|x^1}$ is removed. Since this removal occurs for every $m = 1, \ldots, M$, we conclude that no active path exists between $\boldsymbol{\theta}_{Y|x^0}$ and $\boldsymbol{\theta}_{Y|x^1}$ and thus, the two are independent given the observation of the data. In other words, we can write

$$P(\boldsymbol{\theta}_{Y|X} \mid \mathcal{D}) = P(\boldsymbol{\theta}_{Y|x^1} \mid \mathcal{D}) P(\boldsymbol{\theta}_{Y|x^0} \mid \mathcal{D}).$$

Suppose that $P(\boldsymbol{\theta}_{Y|x^0})$ is a Dirichlet prior with hyperparameters $\alpha_{y^0|x^0}$ and $\alpha_{y^1|x^0}$. As in our discussion of the local decomposition for the likelihood function in section 17.2.3, we have that the likelihood terms that involve $\boldsymbol{\theta}_{Y|x^0}$ are those that measure the probability of $P(y[m] \mid x[m], \boldsymbol{\theta}_{Y|X})$ when $x[m] = x^0$. Thus, we can decompose the joint distribution over parameters and data as follows:

$$\begin{aligned} P(\boldsymbol{\theta}, \mathcal{D}) \quad = \quad & P(\boldsymbol{\theta}_X) L_X(\boldsymbol{\theta}_X : \mathcal{D}) \\ & P(\boldsymbol{\theta}_{Y|x^1}) \prod_{m: x[m] = x^1} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^1}) \\ & P(\boldsymbol{\theta}_{Y|x^0}) \prod_{m: x[m] = x^0} P(y[m] \mid x[m] : \boldsymbol{\theta}_{Y|x^0}). \end{aligned}$$

Thus, this joint distribution is a product of three separate joint distributions with a Dirichlet prior for some multinomial parameter and data drawn from this multinomial. Our analysis for updating a single Dirichlet now applies, and we can conclude that the posterior $P(\boldsymbol{\theta}_{Y|x^0} \mid \mathcal{D})$ is Dirichlet with hyperparameters $\alpha_{y^0|x^0} + M[x^0, y^0]$ and $\alpha_{y^1|x^0} + M[x^0, y^1]$.

We can generalize this discussion to arbitrary networks.

Definition 17.6

local parameter independence

Let X be a variable with parents $\boldsymbol{U}$. We say that the prior $P(\boldsymbol{\theta}_{X|\boldsymbol{U}})$ satisfies local parameter independence *if*

$$P(\boldsymbol{\theta}_{X|\boldsymbol{U}}) = \prod_{\boldsymbol{u}} P(\boldsymbol{\theta}_{X|\boldsymbol{u}}).$$

■

The same pattern of reasoning also applies to the general case.

Proposition 17.6

Let $\mathcal{D}$ be a complete data set for $\mathcal{X}$, let $\mathcal{G}$ be a network structure over these variables with table-CPDs. If the prior $P(\boldsymbol{\theta})$ satisfies global and local parameter independence, then

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_i \prod_{\mathrm{pa}_{X_i}} P(\boldsymbol{\theta}_{X_i|\mathrm{pa}_{X_i}} \mid \mathcal{D}).$$

Moreover, if $P(\boldsymbol{\theta}_{X|\boldsymbol{u}})$ is a Dirichlet prior with hyperparameters $\alpha_{x^1|\boldsymbol{u}}, \ldots, \alpha_{x^K|\boldsymbol{u}}$, then the posterior $P(\boldsymbol{\theta}_{X|\boldsymbol{u}} \mid \mathcal{D})$ is a Dirichlet distribution with hyperparameters $\alpha_{x^1|\boldsymbol{u}} + M[\boldsymbol{u}, x^1], \ldots, \alpha_{x^K|\boldsymbol{u}} + M[\boldsymbol{u}, x^K]$.

As in the case of a single multinomial, this result induces a predictive model in which, for the next instance, we have that

$$P(X_i[M+1] = x_i \mid \boldsymbol{U}[M+1] = \boldsymbol{u}, \mathcal{D}) = \frac{\alpha_{x_i|\boldsymbol{u}} + M[x_i, \boldsymbol{u}]}{\sum_i \alpha_{x_i|\boldsymbol{u}} + M[x_i, \boldsymbol{u}]}. \tag{17.13}$$

Plugging this result into equation (17.12), we see that for computing the probability of a new instance, we can use a single network parameterized as usual, via a set of multinomials, but ones computed as in equation (17.13).

17.4.3 Priors for Bayesian Network Learning

Bayesian network parameter prior

It remains only to address the question of assessing the set of *parameter priors* required for a Bayesian network. In a general Bayesian network, each node X_i has a set of multinomial distributions $\boldsymbol{\theta}_{X_i|\mathrm{pa}_{X_i}}$, one for each instantiation pa_{X_i} of X_i's parents Pa_{X_i}. Each of these parameters will have a separate Dirichlet prior, governed by hyperparameters

$$\boldsymbol{\alpha}_{X_i|\mathrm{pa}_{X_i}} = (\alpha_{x_i^1|\mathrm{pa}_{X_i}}, \ldots, \alpha_{x_i^{K_i}|\mathrm{pa}_{X_i}}),$$

where K_i is the number of values of X_i.

We can, of course, ask our expert to assign values to each of these hyperparameters based on his or her knowledge. This task, however, is rather unwieldy. Another approach, called the K2 prior, is to use a fixed prior, say $\alpha_{x_i^j|\mathrm{pa}_{X_i}} = 1$, for all hyperparameters in the network. As we discuss in the next chapter, this approach has consequences that are conceptually unsatisfying; see exercise 18.10.

A common approach to addressing the specification task uses the intuitions we described in our discussion of Dirichlet priors in section 17.3.1. As we showed, we can think of the hyperparameter α_{x^k} as an imaginary count in our prior experience. This intuition suggests the following representation for a prior over a Bayesian network. Suppose we have an imaginary data set $\mathcal{D}'$ of "prior" examples. Then, we can use counts from this imaginary data set as hyperparameters. More specifically, we set

$$\alpha_{x_i|\mathrm{pa}_{X_i}} = \alpha[x_i, \mathrm{pa}_{X_i}],$$

where $\alpha[x_i, \mathrm{pa}_{X_i}]$ is the number of times $X_i = x_i$ and $\mathrm{Pa}_{X_i} = \mathrm{pa}_{X_i}$ in $\mathcal{D}'$. We can easily see that prediction with this setting of hyperparameters is equivalent to MLE prediction from the combined data set that contains instances of both $\mathcal{D}$ and $\mathcal{D}'$.

One problem with this approach is that it requires storing a possibly large data set of pseudo-instances. Instead, we can store the size of the data set α and a representation $P'(X_1, \ldots, X_n)$ of the frequencies of events in this prior data set. If $P'(X_1, \ldots, X_n)$ is the distribution of events in $\mathcal{D}'$, then we get that

$$\alpha_{x_i|\mathrm{pa}_{X_i}} = \alpha \cdot P'(x_i, \mathrm{pa}_{X_i}).$$

How do we represent P'? Clearly, one natural choice is via a Bayesian network. Then, we can use Bayesian network inference to efficiently compute the quantities $P'(x_i, \mathrm{pa}_{X_i})$. Note that P' does not have to be structured in the same way as the network we learn (although it can be). It is, in fact, quite common to define P' as a set of independent marginals over the X_i's. A prior that can be represented in this manner (using α and P') is called a *BDe prior*. Aside from being philosophically pleasing, it has some additional benefits that we will discuss in the next chapter.

BDe prior

Box 17.C — Case Study: Learning the ICU-Alarm **Network.** *To give an example of the techniques described in this chapter, we evaluate them on a synthetic example. Figure 17.C.1 shows the graph structure of the real-world* ICU-Alarm *Bayesian network, hand-constructed by an expert, for monitoring patients in an Intensive Care Unit (ICU). The network has 37 nodes and a total of 504 parameters. We want to evaluate the ability of our parameter estimation algorithms to reconstruct the network parameters from data.*

ICU-Alarm

We generated a training set from the network, by sampling from the distribution specified by the network. We then gave the algorithm only the (correct) network structure, and the generated data, and measured the ability of our algorithms to reconstruct the parameters. We tested the MLE approach, and several Bayesian approaches. All of the approaches used a uniform prior mean, but different prior strengths α.

In performing such an experiment, there are many ways of measuring the quality of the learned network. One possible measure is the difference between the original values of the model parameters and the estimated ones. A related approach measures the distance between the original CPDs and the learned ones (in the case of table-CPDs, these two approaches are the same, but not for general parameterizations). These approaches place equal weights on different parameters, regardless of the extent to which they influence the overall distribution.

The approach we often take is the one described in section 16.2.1, where we measure the relative entropy between the generating distribution P^* and the learned distribution $\tilde{P}$ (see also section 8.4.2). This approach provides a global measure of the extent to which our learned distribution resembles the true distribution. Figure 17.C.2 shows the results for different stages of learning. As we might expect, when more instances are available, the estimation is better. The improvement is drastic in early stages of learning, where additional instances lead to major improvements. When the number of instances in our data set is larger, additional instances lead to improvement, but a smaller one.

More surprisingly, we also see that the MLE achieves the poorest results, a consequence of its extreme sensitivity to the specific training data used. The lowest error is achieved with a very weak prior — $\alpha = 5$ — which is enough to provide smoothing. As the strength of the prior grows, it starts to introduce a bias, not giving the data enough importance. Thus, the error of the estimated probability increases. However, we also note that the effect of the prior, even for $\alpha = 50$, disappears reasonably soon, and all of the approaches converge to the same line. Interestingly, the different

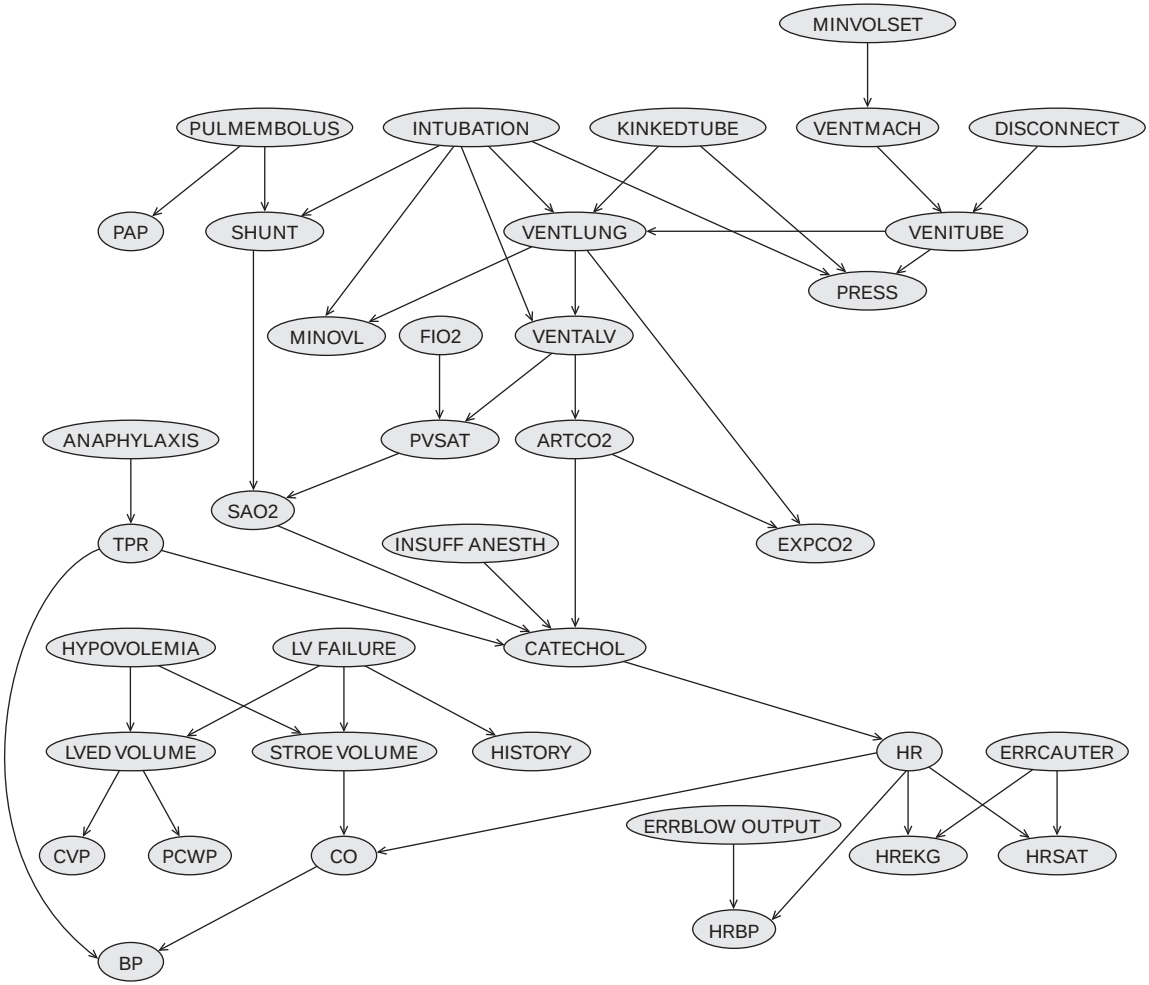


Figure 17.C.1 — The ICU-Alarm **Bayesian network.**

Bayesian approaches converge to this line long before the MLE approach. Thus, at least in this example, an overly strong bias provided by the prior is still a better compromise than the complete lack of smoothing of the MLE approach.

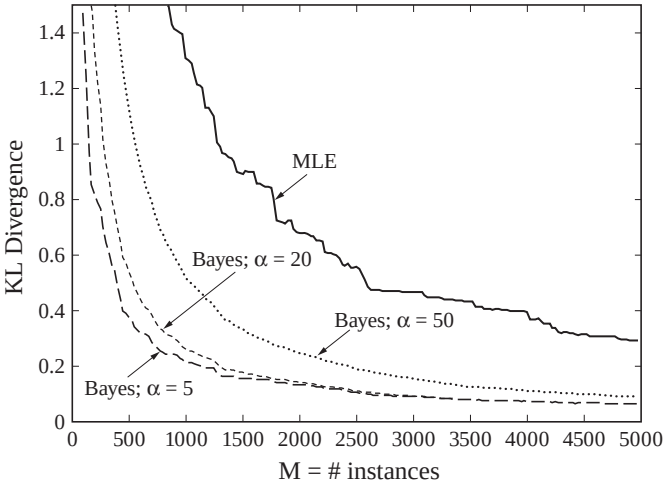


Figure 17.C.2 — Learning curve for parameter estimation for the ICU-Alarm **network** Relative entropy to true model as the amount of training data grows, for different values of the prior strength α.

17.4.4 MAP Estimation ⋆

Our discussion in this chapter has focused solely on Bayesian estimation for multinomial CPDs. Here, we have a closed form solution for the integral required for Bayesian prediction, and thus we can perform it efficiently. In many other representations, the situation is not so simple. In some cases, such as the noisy-or model or the logistic CPDs of section 5.4.2, we do not have a conjugate prior or a closed-form solution for the Bayesian integral. In those cases, Bayesian prediction requires numerical solutions for high-dimensional integrals. In other settings, such as the linear Gaussian CPD, we do have a conjugate prior (the *normal-Gamma distribution*), but we may prefer other priors that offer other desirable properties (such as the sparsity-inducing Laplacian prior described in section 20.4.1).

normal-Gamma distribution

MAP estimation

When a full Bayesian solution is impractical, we can resort to using *maximum a posteriori (MAP) estimation*. Here, we search for parameters that maximize the *posterior probability*:

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log P(\boldsymbol{\theta} \mid \mathcal{D}).$$

When we have a large amount of data, the posterior is often sharply peaked around its maximum $\tilde{\boldsymbol{\theta}}$. In this case, the integral

$$P(X[M+1] \mid \mathcal{D}) = \int P(X[M+1] \mid \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}$$

will be roughly $P(X[M+1] \mid \tilde{\boldsymbol{\theta}})$. More generally, we can view the MAP estimate as a way of using the prior to provide *regularization* over the likelihood function:

regularization

$$\begin{aligned}\arg\max_{\boldsymbol{\theta}} \log P(\boldsymbol{\theta} \mid \mathcal{D}) &= \arg\max_{\boldsymbol{\theta}} \log \left(\frac{P(\boldsymbol{\theta}) P(\mathcal{D} \mid \boldsymbol{\theta})}{P(\mathcal{D})} \right) \\ &= \arg\max_{\boldsymbol{\theta}} \left(\log P(\boldsymbol{\theta}) + \log P(\mathcal{D} \mid \boldsymbol{\theta}) \right). \end{aligned} \tag{17.14}$$

That is, $\tilde{\boldsymbol{\theta}}$ is the maximum of a function that sums together the log-likelihood function and $\log P(\boldsymbol{\theta})$. This latter term takes into account the prior on different parameters and therefore biases the parameter estimate away from undesirable parameter values (such as those involving conditional probabilities of 0) when we have few learning instances. When the number of samples is large, the effect of the prior becomes negligible, since $\ell(\boldsymbol{\theta} : \mathcal{D})$ grows linearly with the number of samples whereas the prior does not change.

Because our parameter priors are generally well behaved, MAP estimation is often no harder than maximum likelihood estimation, and is therefore often applicable in practice, even in cases where Bayesian estimation is not. Importantly, however, it does not offer all of the same benefits as a full Bayesian estimation. In particular, it does not attempt to represent the shape of the posterior and thus does not differentiate between a flat posterior and a sharply peaked one. As such, it does not give us a sense of our confidence in different aspects of the parameters, and the predictions do not average over our uncertainty. This approach also suffers from issues regarding representation independence; see box 17.D.

---

representation independence

Box 17.D — Concept: Representation Independence. *One important property we may want of an estimator is* representation independence. *To understand this concept better, suppose that in our thumbtack example, we choose to use a parameter η, so that $P'(X = H \mid \eta) = \frac{1}{1+e^{-\eta}}$. We have that $\eta = \log \frac{\theta}{1-\theta}$ where θ is the parameter we used earlier. Thus, there is a one-to-one correspondence between a choice θ and a choice η. Although one choice of parameters might seem more natural to us than another, there is no formal reason why we should prefer one over the other, since both can represent exactly the same set of distributions.*

More generally, a reparameterization of a given family is a new set of parameter values η in a space Υ and a mapping from the new parameters to the original one, that is, from η to $\boldsymbol{\theta}(\eta)$ so that $P(\cdot \mid \eta)$ in the new parameterization is equal to $P(\cdot \mid \boldsymbol{\theta}(\eta))$ in the original parameterization. In addition, we require that the reparameterization maintain the same set of distributions, that is, for each choice of $\boldsymbol{\theta}$ there is η such that $P(\cdot \mid \eta) = P(\cdot \mid \boldsymbol{\theta})$.

This concept immediately raises the question as to whether the choice of representation can impact our estimates. While we might prefer a particular way of parameterization because it is more intuitive or interpretable, we may not want this choice to bias our estimated parameters.

Fortunately, it is not difficult to see that maximum likelihood estimation is insensitive to reparameterization. If we have two different ways to represent the same family of the distribution, then the distributions *in the family that maximize the likelihood using one parameterization also maximize the likelihood with the other parameterization. More precisely, if $\hat{\eta}$ is MLE, then the matching parameter values $\boldsymbol{\theta}(\hat{\eta})$ are also MLE when we consider the likelihood function in the $\boldsymbol{\theta}$ space. This property is a direct consequence of the fact that the likelihood function is a function of the distribution induced by the parameter values, and not of the actual parameter values.*

The situation with Bayesian inference is subtler. Here, instead of identifying the maximum parameter value, we now perform integration over all possible parameter values. Naively, it seems that such an estimation is more sensitive to the parameterization than MLE, which depends only

on the maximum of the likelihood surface. However, a careful choice of prior can account for the representation change and thereby lead to representation independence. Intuitively, if we consider a reparameterization η with a function $\boldsymbol{\theta}(\eta)$ mapping to the original parameter space, then we would like the prior on η to maintain the probability of events. That is,

$$P(A) = P(\{\boldsymbol{\theta}(\eta) : \eta \in A\}), \ \forall A \subset \Upsilon. \tag{17.15}$$

This constraint implies that the prior over different regions of parameters is maintained. Under this assumption, Bayesian prediction will be identical under the two parameterizations.

We illustrate the notion of a reparameterized prior in the context of a Bernoulli distribution:

Example 17.9

Consider a Beta prior over the parameter θ of a Bernoulli distribution:

$$P(\theta : \alpha_0, \alpha_1) = c\theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1}, \tag{17.16}$$

where c is the normalizing constant described in definition 17.3. Recall (example 8.5) that the natural parameter for a Bernoulli distribution is

$$\eta = \log \frac{\theta}{1-\theta}$$

with the transformation

$$\theta = \frac{1}{1+e^{-\eta}}, \quad 1-\theta = \frac{1}{1+e^{\eta}}.$$

What is the prior distribution on η? To preserve the probability of events, we want to make sure that for every interval $[a, b]$

$$\int_a^b c\theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1} d\theta = \int_{\log \frac{a}{1-a}}^{\log \frac{b}{1-b}} P(\eta) d\eta.$$

To do so, we need to perform a change of variables. Using the relation between η and θ, we get

$$d\eta = \frac{1}{\theta(1-\theta)} d\theta.$$

Plugging this into the equation, we can verify that an appropriate prior is:

$$P(\eta) = c\left(\frac{1}{1+e^{-\eta}}\right)^{\alpha_1}\left(\frac{1}{1+e^{\eta}}\right)^{\alpha_0},$$

where c is the same constant as before. This means that the prior on η, when stated in terms of θ, is $\theta^{\alpha_1}(1-\theta)^{\alpha_0}$, in contrast to equation (17.16). At first this discrepancy seems like a contradiction. However, we have to remember that the transformation from θ to η takes the region $[0, 1]$ and stretches it to the whole real line. Thus, the matching prior cannot be uniform. ■

This example demonstrates that a uniform prior, which we consider to be unbiased or uninformative, can seem very different when we consider a different parameterization.

Thus, both MLE and Bayesian estimation (when carefully executed) are representation-independent. This property, unfortunately, does not carry through to MAP estimation. Here we are not interested in the integral over all parameters, but rather in the density of the prior at different values of the parameters. This quantity does change when we reparameterize the prior.

Example 17.10

Consider the setting of example 17.9 and develop the MAP parameters for the priors we considered there. When we use the θ parameterization, we can check that

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\theta} \log P(\theta) = \frac{\alpha_1 - 1}{\alpha_0 + \alpha_1 - 2}.$$

On the other hand,

$$\tilde{\eta} = \arg\max_{\eta} \log P(\eta) = \log \frac{\alpha_1}{\alpha_0}.$$

To compare the two, we can transform $\tilde{\eta}$ to θ representation and find

$$\boldsymbol{\theta}(\tilde{\eta}) = \frac{\alpha_1}{\alpha_0 + \alpha_1}.$$

In other words, the MAP of the η parameterization gives the same predictions as the mean parameterization if we do the full Bayesian inference. ■

☞ **Thus, MAP estimation is more sensitive to choices in formalizing the likelihood and the prior than MLE or full Bayesian inference. This suggests that the MAP parameters involve, to some extent, an arbitrary choice.** *Indeed, we can bias the MAP toward different solutions if we construct a specific reparameterization where the density is particularly large in specific regions of the parameter space. The parameterization dependency of MAP is a serious caveat we should be aware of.*

17.5 Learning Models with Shared Parameters

In the preceding discussion, we focused on parameter estimation for Bayesian networks with table-CPDs. In this discussion, we made the strong assumption that the parameters for each conditional distribution $P(X_i \mid \boldsymbol{u}_i)$ can be estimated separately from parameters of other conditional distributions. In the Bayesian case, we also assumed that the priors on these distributions are independent. This assumption is a very strong one, which often does not hold in practice. In real-life systems, we often have *shared parameters*: parameters that occur in multiple places across the network. In this section, we discuss how to perform parameter estimation in networks where the same parameters are used multiple times.

shared parameters

Analogously to our discussion of parameter estimation, we can exploit both global and local structure. Global structure occurs when the same CPD is used across multiple variables in the network. This type of sharing arises naturally from the template-based models of chapter 6. Local structure is finer-grained, allowing parameters to be shared even within a single CPD; it arises naturally in some types of structured CPDs. We discuss each of these scenarios in turn, focusing on the simple case of MLE.

We then discuss the issues arising when we want to use Bayesian estimation. Finally, we discuss the *hierarchical Bayes* framework, a "softer" version of parameter sharing, where parameters are encouraged to be similar but do not have to be identical.

17.5.1 Global Parameter Sharing

Let us begin with a motivating example.

Example 17.11

Let us return to our student, who is now taking two classes c_1, c_2, each of which is associated with a Difficulty variable, D_1, D_2. We assume that the grade G_i of our student in class c_i depends on his intelligence and the class difficulty. Thus, we model G_i as having I and D_i as parents. Moreover, we might assume that these grades share the same conditional distribution. That is, the probability that an intelligent student receives an "A" in an easy class is the same regardless of the identity of the particular class. Stated differently, we assume that the difficulty variable summarizes all the relevant information about the challenge the class presents to the student.

How do we formalize this assumption? A straightforward solution is to require that for all choices of grade g, difficulty d and intelligence i, we have that

$$P(G_1 = g \mid D_1 = d, I = i) = P(G_2 = g \mid D_2 = d, I = i).$$

Importantly, this assumption does not imply that the grades are necessarily the same, but rather that the probability of getting a particular grade is the same if the class has the same difficulty. ■

This example is simply an instance of a network induced by the simple plate model described in example 6.11 (using D_i to encode $D(c_i)$ and similarly for G_i). Thus, as expected, template models give rise to shared parameters.

17.5.1.1 Likelihood Function with Global Shared Parameters

As usual, the key to parameter estimation lies in understanding the structure of the likelihood function. To analyze this structure, we begin with some notation. Consider a network structure $\mathcal{G}$ over a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$, parameterized by a set of parameters $\boldsymbol{\theta}$. Each variable X_i is associated with a CPD $P(X_i \mid \boldsymbol{U}_i, \boldsymbol{\theta})$. Now, rather than assume that each such CPD has its own parameterization $\boldsymbol{\theta}_{X_i|\boldsymbol{U}_i}$, we assume that we have a certain set of shared parameters that are used by multiple variables in the network. Thus, the sharing of parameters is *global*, over the entire network.

global parameter sharing

More precisely, we assume that $\boldsymbol{\theta}$ is partitioned into disjoint subsets $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^K$; with each such subset, we associate a set of variables $\mathcal{V}^k \subset \mathcal{X}$, such that $\mathcal{V}^1, \ldots, \mathcal{V}^K$ is a disjoint partition of $\mathcal{X}$. For $X_i \in \mathcal{V}^k$, we assume that the CPD of X_i depends only on $\boldsymbol{\theta}^k$; that is,

$$P(X_i \mid \boldsymbol{U}_i, \boldsymbol{\theta}) = P(X_i \mid \boldsymbol{U}_i, \boldsymbol{\theta}^k). \tag{17.17}$$

Moreover, we assume that the form of the CPD is the same for all $X_i, X_j \in \mathcal{V}^k$; that is,

$$P(X_i \mid \boldsymbol{U}_i, \boldsymbol{\theta}^k) = P(X_j \mid \boldsymbol{U}_j, \boldsymbol{\theta}^k). \tag{17.18}$$

We note that this last statement makes sense only if $Val(X_i) = Val(X_j)$ and $Val(\boldsymbol{U}_i) = Val(\boldsymbol{U}_j)$. To avoid ambiguous notation, for any variable $X_i \in \mathcal{V}^k$, we use y_k^l to range over possible values of X_i and $\boldsymbol{w}_k^l$ to range over the possible values of its parents.

Consider the decomposition of the probability distribution in this case:

$$\begin{aligned} P(X_1, \ldots, X_n \mid \boldsymbol{\theta}) &= \prod_{i=1}^{n} P(X_i \mid \mathrm{Pa}_{X_i}, \boldsymbol{\theta}) \\ &= \prod_{k=1}^{K} \prod_{X_i \in \mathcal{V}^k} P(X_i \mid \mathrm{Pa}_{X_i}, \boldsymbol{\theta}) \\ &= \prod_{k=1}^{K} \prod_{X_i \in \mathcal{V}^k} P(X_i \mid \mathrm{Pa}_{X_i}, \boldsymbol{\theta}^k), \end{aligned}$$

where the second equality follows from the fact that $\mathcal{V}^1, \ldots, \mathcal{V}^K$ defines a partition of $\mathcal{X}$, and the third equality follows from equation (17.17).

Now, let $\mathcal{D}$ be some assignment of values to the variables $X_1, \ldots, \mathcal{X}_n$; our analysis can easily handle multiple IID instances, as in our earlier discussion, but this extension only clutters the notation. We can now write

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_{k=1}^{K} \prod_{X_i \in \mathcal{V}^k} P(x_i \mid \boldsymbol{u}_i, \boldsymbol{\theta}^k).$$

This expression is identical to the one we used in section 17.2.2 for the case of IID instances. There, for each set of parameters, we had multiple instances $\{(x_i[m], \boldsymbol{u}_i[m])\}_{m=1}^{M}$, all of which were generated from the same conditional distribution. Here, we have multiple instances $\{(x_i, \boldsymbol{u}_i)\}_{X_i \in \mathcal{V}^k}$, all of which are also generated from the same conditional distribution. Thus, it appears that we can use the same analysis as we did there.

To provide a formal derivation, consider first the case of table-CPDs. Here, our parameterization is a set of multinomial parameters $\theta^k_{y_k|\boldsymbol{w}_k}$, where we recall that y_k ranges over the possible values of each of the variables $X_i \in \mathcal{V}^k$ and $\boldsymbol{w}_k$ over the possible value assignments to its parents. Using the same derivation as in section 17.2.2, we can now write:

$$\begin{aligned} L(\boldsymbol{\theta} : \mathcal{D}) &= \prod_{k=1}^{K} \prod_{y_k, \boldsymbol{w}_k} \prod_{\substack{X_i \in \mathcal{V}^k : \\ x_i = y_k, \boldsymbol{u}_i = \boldsymbol{w}_k}} \theta^k_{y_k|\boldsymbol{w}_k} \\ &= \prod_{k=1}^{K} \prod_{y_k, \boldsymbol{w}_k} (\theta^k_{y_k|\boldsymbol{w}_k})^{\check{M}_k[y_k, \boldsymbol{w}_k]}, \end{aligned}$$

where we now have a new definition of our counts:

$$\check{M}_k[y_k, \boldsymbol{w}_k] = \sum_{X_i \in \mathcal{V}^k} \boldsymbol{I}\{x_i = y_k, \boldsymbol{u}_i = \boldsymbol{w}_k\}.$$

aggregate sufficient statistics

In other words, we now use *aggregate sufficient statistics*, which combine sufficient statistics from multiple variables across the same network.

Given this formulation of the likelihood, we can now obtain the maximum likelihood solution for each set of shared parameters to get the estimate

$$\hat{\theta}^k_{y_k|\boldsymbol{w}_k} = \frac{\check{M}_k[y_k, \boldsymbol{w}_k]}{\check{M}_k[\boldsymbol{w}_k]}.$$

Thus, we use the same estimate as in the case of independent parameters, using our aggregate sufficient statistics. Note that, in cases where our variables X_i have no parents, $\boldsymbol{w}_k$ is the empty tuple ε. In this case, $\check{M}_k[\varepsilon]$ is the number of variables X_i in $\mathcal{V}^k$.

linear exponential family

This aggregation of sufficient statistics applies not only to multinomial distributions. Indeed, for any distribution in the *linear exponential family*, we can perform precisely the same aggregation of sufficient statistics over the variables in $\mathcal{V}^k$. The result is a likelihood function in the same form as we had before, but written in terms of the aggregate sufficient statistics rather than the sufficient statistics for the individual variables. We can then perform precisely the same maximum likelihood estimation process and obtain the same form for the MLE, but using the aggregate sufficient statistics. (See exercise 17.14 for another simple example.)

Does this aggregation of sufficient statistics make sense? Returning to our example, if we treat the grade of the student in each class as independent sample from the same parameters, then each data instance provides us with two independent samples from this distribution. It is important to clarify that, although the grades of the student are dependent on his intelligence, the samples are independent samples from the same distribution. More precisely, if $D_1 = D_2$, then both G_1 and G_2 are governed by the same multinomial distribution, and the student's grades are two independent samples from this distribution.

Thus, when we share parameters, multiple observations from within the same network contribute to the same sufficient statistic, and thereby help estimate the same parameter. Reducing the number of parameters allows us to obtain parameter estimates that are less noisy and closer to the actual generating parameters. This benefit comes at a price, since it requires us to make an assumption about the domain. If the two distributions with shared parameters are actually different, the estimated parameters will be a (weighted) average of the estimate we would have had for each of them separately. When we have a small number of instances, that approximation may still be beneficial, since each of the separate estimates may be far from its generating parameters, owing to sample noise. When we have more data, however, the shared parameters estimate will be worse than the individual ones. We return to this issue in section 17.5.4, where we provide a solution that allows us to gradually move away from the shared parameter assumption as we get more data.

17.5.1.2 Parameter Estimation for Template-Based Models

As we mentioned, the template models of chapter 6 are specifically designed to encode global parameter sharing. Recall that these representations involve a set of template-level parameters, each of which is used multiple times when the ground network is defined. For the purpose of our discussion, we focus mostly on plate models, since they are the simplest of the (directed) template-based representations and serve to illustrate the main points.

As we discussed, it is customary in many plate models to explicitly encode the parameter sharing by including, in the model, random variables that encode the model parameters. This approach allows us to make clear the exact structure of the parameter sharing within the model.

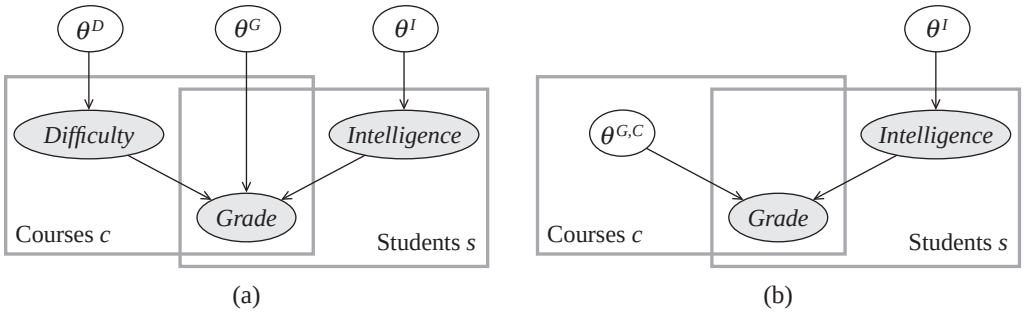


Figure 17.9 Two plate models for the University **example, with explicit parameter variables.** (a) Model where all parameters are global. (b) Model where the difficulty is a course-specific parameter rather than a discrete random variable.

As we mentioned, when parameters are global and shared across over all ground variables derived from a particular template variables, we may choose (purely as a notational convenience) not to include the parameters explicitly in the model. We begin with this simple setting, and then we extend it to the more general case.

Example 17.12

Figure 17.9a is a representation of the plate model of example 6.11, except that we now explicitly encode the parameters as variables within the model. In this representation, we have made it clear that there is only a single parameter $\theta^G_{G|I,D}$, which is the parent of the variables within the plates. Thus, as we can see, the same parameters are used in every CPD $P(G(s,c) \mid I(s), D(c))$ in the ground network. ■

When all of our parameters are global, the sharing structure is very simple. Let $A(U_1, \ldots, U_k)$ be any attribute in the set of template attributes $\aleph$. Recall from definition 6.10 that this attribute induces a ground random variable $A(\gamma)$ for any assignment $\gamma = \langle U_1 \mapsto u_1, \ldots, U_k \mapsto u_k \rangle \in \Gamma_\kappa[A]$. All of these variables share the same CPD, and hence the same parameters. Let $\boldsymbol{\theta}_A$ be the parameters for the CPD for A in the template-level model. We can now simply define $\mathcal{V}^A$ to be all of the variables of the form $A(\gamma)$ in the ground network. The analysis of section 17.5.1.1 now applies unchanged.

Example 17.13

Continuing example 17.12, the likelihood function for an assignment ξ to a ground network in the University *domain would have the form*

$$\prod_i (\theta^I_i)^{\check{M}_I[i]} \prod_d (\theta^D_d)^{\check{M}_D[d]} \prod_{g,i,d} (\theta^G_{g|i,d})^{\check{M}_G[g,i,d]}.$$

Importantly, the counts are computed as aggregate sufficient statistics, each with its own appropriate set of variables. In particular,

$$\check{M}_I[i] = \sum_{s \in \mathcal{O}^\kappa[\mathsf{Student}]} \mathbf{1}\{I(s) = i\},$$

whereas

$$\check{M}_G[g,i,d] = \sum_{\langle s,c\rangle \in \Gamma_\kappa[\mathit{Grade}]} \mathbf{I}\{I(s)=i, D(c)=d, G(s,c)=g\}.$$

For example, the counts for g^1, i^1, d^1 would be the number of all of the student,course pairs s, c such that student s has high intelligence, course c has high difficulty, and the grade of s in c is an A. The MLE for $\theta^G_{g^1|i^1,d^1}$ is the fraction of those students among all (student,course) pairs. ■

To provide a concrete formula in the more general case, we focus on table-CPDs. We can now define, for any attribute $A(\mathcal{X})$ with parents $B_1(\boldsymbol{U}_1), \ldots, B_k(\boldsymbol{U}_k)$, the following aggregate sufficient statistics:

$$\check{M}_A[a, b_1, \ldots, b_k] = \sum_{\gamma \in \Gamma_\kappa[A]} \mathbf{I}\{A(\gamma) = a, B_1(\gamma[\boldsymbol{U}_1]) = b_1, \ldots,\ B_k(\gamma[\boldsymbol{U}_k]) = b_k\}, \quad (17.19)$$

where $\gamma[\boldsymbol{U}_j]$ denotes the subtuple of the assignment to $\boldsymbol{U}$ that corresponds to the logical variables in $\boldsymbol{U}_j$. We can now define the template-model log-likelihood for a given skeleton:

$$\ell(\boldsymbol{\theta} : \kappa) = \sum_{A \in \aleph} \sum_{a \in Val(A)} \sum_{\boldsymbol{b} \in Val(\mathrm{Pa}_A)} \check{M}_A[a, \boldsymbol{b}] \log \theta_{A=a, \mathrm{Pa}_A = \boldsymbol{b}}. \quad (17.20)$$

From this formula, the maximum likelihood estimate for each of the model parameters follows easily, using precisely the same analysis as before.

In our derivation so far, we focused on the setting where all of the model parameters are global. However, as we discussed, the plate representation can also encompass more local parameterizations. Fortunately, from a learning perspective, we can reduce this case to the previous one.

Example 17.14

Figure 17.9b represents the setting where each course has its own model for how the course difficulty affects the students' grades. That is, we now have a set of parameters $\boldsymbol{\theta}^{G,c}$, which are used in the CPDs of all of the ground variables $G(c, s)$ for different values of s, but there is no sharing between $G(c, s)$ and $G(c', s)$.

As we discussed, this setting is equivalent to one where we add the course ID c as a parent to the D variable, forcing us to introduce a separate set of parameters for every assignment to c. In this case, the dependence on the specific course ID subsumes the dependence on the difficulty parameter D, which we have (for clarity) dropped from the model. From this perspective, the parameter estimation task can now be handled in exactly the same way as before for the parameters in the (much larger) CPD for G. In effect, this transformation converts global sharing to local sharing, which we will handle. ■

There is, however, one important subtlety in scenarios such as this one. Recall that, in general, different skeletons will contain different objects. Parameters that are specific to objects in the model do not transfer from one skeleton to another. Thus, we cannot simply transfer the object-specific parameters learned from one skeleton to another. This limitation is important, since a major benefit of the template-based formalisms is the ability to learn models in one instantiation of the template and use it in other instantiations. Nevertheless, the learned models are still useful

in many ways. First, the learned model itself often provides significant insight about the objects in the training data. For example, the LDA model of box 17.E tells us, for each of the documents in our training corpus, what the mix of topics in that particular document is. Second, the model parameters that are not object specific can be transferred to other skeletons. For example, the word multinomials associated with different topics that are learned from one document collection can also be used in another, leaving only the new document-specific parameters to be inferred.

Although we have focused our formal presentation on plate models, the same analysis also applies to DBNs, PRMs, and a variety of other template-based languages that share parameters. See exercise 17.16 and exercise 17.17 for two examples.

17.5.2 Local Parameter Sharing

local parameter sharing

In the first part of this section, we focused on cases where all of the parameter sharing occurred between different CPDs. However, we might also have shared parameters that are shared *locally*, within a single CPD.

Example 17.15

Consider the CPD of figure 5.4 where we model the probability of the student getting a job based on his application, recommendation letter, and SAT scores. As we discussed there, if the student did not formally apply for a position, then the recruiting company does not have access to the recommendation letter or SAT scores. Thus, for example, the conditional probability distribution $P(J \mid a^0, s^0, l^0)$ *is equal to* $P(J \mid a^0, s^1, l^1)$. ■

In fact, the representations we considered in section 5.3 can be viewed as encoding parameter sharing for different conditional distributions. That is, each of these representations (for example, tree-CPDs) is a language to specify which of the conditional distributions within a CPD are equal to each other. As we saw, these equality constraints had implications in terms of the independence statements encoded by a model and can also in some cases be exploited in inference. (We note that not all forms of local structure can be reduced to a simple set of equality constraints on conditional distributions within a CPD. For example, noisy-or CPDs or generalized linear models combine their parameters in very different ways and require very different techniques than the ones we discuss in this section.)

Here, we focus on the setting where the CPD defines a set of multinomial distributions, but some of these distributions are shared across multiple contexts. In particular, we assume that our graph $\mathcal{G}$ now defines a set of multinomial distributions that make up CPDs for $\mathcal{G}$: for each variable X_i and each $\boldsymbol{u}_i \in Val(\boldsymbol{U}_i)$ we have a multinomial distribution. We can use the tuple $\langle X_i, \boldsymbol{u}_i \rangle$ to designate this multinomial distribution, and define

$$\mathcal{D} = \cup_{i=1}^{n} \{\langle X_i, \boldsymbol{u}_i \rangle : \boldsymbol{u}_i \in Val(\boldsymbol{U}_i)\}$$

to be the set containing all the multinomial distributions in $\mathcal{G}$. We can now define a set of shared parameters $\boldsymbol{\theta}^k$, $k = 1, \ldots, K$, where each $\boldsymbol{\theta}^k$ is associated with a set $\mathcal{D}^k \subseteq \mathcal{D}$ of multinomial distributions. As before, we assume that $\mathcal{D}^1, \ldots, \mathcal{D}^K$ defines a disjoint partition of $\mathcal{D}$. We assume that all conditional distributions within $\mathcal{D}^k$ share the same parameters $\boldsymbol{\theta}^k$. Thus, we have that if $\langle X_i, \boldsymbol{u}_i \rangle \in \mathcal{D}^k$, then

$$P(x_i^j \mid \boldsymbol{u}_i) = \theta_j^k.$$

For this constraint to be coherent, we require that all the multinomial distributions within the same partition have the same set of values: for any $\langle X_i, \boldsymbol{u}_i\rangle, \langle X_j, \boldsymbol{u}_j\rangle \in \mathcal{D}^k$, we have that $Val(X_i) = Val(X_j)$.

Clearly, the case where no parameter is shared can be represented by the trivial partition into singleton sets. However, we can also define more interesting partitions.

Example 17.16

To capture the tree-CPD of figure 5.4, we would define the following partition:

$$\begin{aligned}
\mathcal{D}^{a^0} &= \{\langle J, (a^0, s^0, l^0)\rangle, \langle J, (a^0, s^0, l^1)\rangle, \langle J, (a^0, s^1, l^0)\rangle, \langle J, (a^0, s^1, l^1)\rangle\} \\
\mathcal{D}^{a^1, s^0, l^0} &= \{\langle J, (a^1, s^0, l^0)\rangle\} \\
\mathcal{D}^{a^1, s^0, l^1} &= \{\langle J, (a^1, s^0, l^1)\rangle\} \\
\mathcal{D}^{a^1, s^1} &= \{\langle J, (a^1, s^1, l^0)\rangle, \langle J, (a^1, s^1, l^1)\rangle\}.
\end{aligned}$$

■

More generally, this partition-based model can capture local structure in both tree-CPDs and in rule-based CPDs. In fact, when the network is composed of multinomial CPDs, this finer-grained sharing also generalizes the sharing structure in the global partition models of section 17.5.1.

We can now reformulate the likelihood function in terms of the shared parameters $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^K$. Recall that we can write

$$P(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_i \prod_{\boldsymbol{u}_i} \left[\prod_{x_i} P(x_i \mid \boldsymbol{u}_i, \boldsymbol{\theta})^{M[x_i, \boldsymbol{u}_i]} \right].$$

Each of the terms in square brackets is the local likelihood of a single multinomial distribution. We now rewrite each of the terms in the innermost product in terms of the shared parameters, and we aggregate them according to the partition:

$$\begin{aligned}
P(\mathcal{D} \mid \boldsymbol{\theta}) &= \prod_i \prod_{\boldsymbol{u}_i} \left[\prod_{x_i} P(x_i \mid \boldsymbol{u}_i, \boldsymbol{\theta})^{M[x_i, \boldsymbol{u}_i]} \right] \\
&= \prod_k \prod_{\langle X_i, \boldsymbol{u}_i\rangle \in \mathcal{D}^k} \prod_j (\theta_j^k)^{M[x_i^j, \boldsymbol{u}_i]} \\
&= \prod_k \left[\prod_j (\theta_j^k)^{\sum_{\langle X_i, \boldsymbol{u}_i\rangle \in \mathcal{D}^k} M[x_i^j, \boldsymbol{u}_i]} \right]. \qquad (17.21)
\end{aligned}$$

This final expression is reminiscent of the likelihood in the case of independent parameters, except that now each of the terms in the square brackets involves the shared parameters. Once again, we can define a notion of aggregate sufficient statistics:

$$\check{M}_k[j] = \sum_{\langle X_i, \boldsymbol{u}_i\rangle \in \mathcal{D}^k} M[x_i^j, \boldsymbol{u}_i],$$

and use those aggregate sufficient statistics for parameter estimation, exactly as we used the unaggregated sufficient statistics before.

17.5.3 Bayesian Inference with Shared Parameters

To perform Bayesian estimation, we need to put a prior on the parameters. In the case without parameter sharing, we had a separate (independent) prior for each parameter. This model is clearly in violation of the assumptions made by parameter sharing. If two parameters are shared, we want them to be identical, and thus it is inconsistent to assume they have independent prior. The right approach is to place a prior on the shared parameters.

Consider, in particular, the local analysis of section 17.5.2, we would place a prior on each of the multinomial parameters $\boldsymbol{\theta}^k$. As before, it is very convenient to assume that each of these set of parameters are independent from each other. This assumption corresponds to the local parameter independence we made earlier, but applied in the context where we force the given parameter-sharing strategy. We can use a similar idea in the global analysis of section 17.5.1, introducing a prior over each set of parameters θ^k. If we impose an independence assumption for the priors of the different sets, we obtain a shared-parameter version of the global parameter independence assumption.

One important subtlety relates to the choice of the prior. Given a model with shared parameters, the analysis of section 17.4.3 no longer applies directly. See exercise 17.13 for one possible extension.

As usual, if the prior decomposes as a product and the likelihood decomposes as a product along the same lines, then our posterior also decomposes. For example, returning to equation (17.21), we have that:

$$P(\boldsymbol{\theta} \mid \mathcal{D}) \propto \prod_{k=1}^{K} P(\boldsymbol{\theta}^k) \prod_j (\theta^k_j)^{\check{M}_k[j]}.$$

The actual form of the posterior depends on the prior. Specifically, if we use multinomial distributions with Dirichlet priors, then the posterior will also be a Dirichlet distribution with the appropriate hyperparameters.

This discussion seems to suggest that the line of reasoning we had in the case of independent parameters is applicable to the case of shared parameters. However, there is one subtle point that can be different. Consider the problem of predicting the probability of the next instance, which can be written as:

$$P(\xi[M+1] \mid \mathcal{D}) = \int P(\xi[M+1] \mid \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}.$$

To compute this formula, we argued that, since $P(\xi[M+1] \mid \boldsymbol{\theta})$ is a product of parameters $\prod_i \theta_{x_i[M+1]|\boldsymbol{u}_i[M+1]}$, and since the posterior of these parameters are independent, then we can write (for multinomial parameters)

$$P(\xi[M+1] \mid \mathcal{D}) = \prod_i \boldsymbol{E}\big[\theta_{x_i[M+1]|\boldsymbol{u}_i[M+1]} \mid \mathcal{D}\big],$$

where each of the expectations is based on the posterior over $\theta_{x_i[M+1]|\boldsymbol{u}_i[M+1]}$.

When we have shared parameters, we have to be more careful. If we consider the network of example 17.11, then when the $(M+1)$st instance has two courses of the same difficulty, the likelihood term $P(\xi[M+1] \mid \boldsymbol{\theta})$ involves a product of two parameters that are not independent. More explicitly, the likelihood involves $P(G_1[M+1] \mid I[M+1], D_1[M+1])$ and $P(G_2[M+1] \mid$

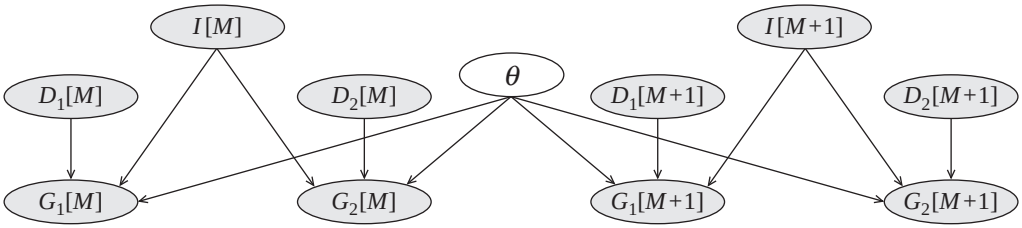


Figure 17.10 Example meta-network for a model with shared parameters, corresponding to example 17.11.

$I[M+1], D_2[M+1]$); if $D_1[M+1] = D_2[M+1]$ then the two parameters are from the same multinomial distribution. Thus, the posterior over these two parameters is not independent, and we cannot write their expectation as a product of expectations.

Another way of understanding this problem is by examining the meta-network for learning in such a situation. The meta-network for Bayesian parameter learning for the network of example 17.11 is shown in figure 17.10. As we can see, in this network $G_1[M+1]$ is *not* independent of $G_2[M+1]$ given $I[M+1]$ because of the trail through the shared parameters. Stated differently, observing $G_1[M+1]$ will cause us to update the parameters and therefore change our estimate of the probability of $G_2[M+1]$.

Note that this problem can happen only in particular forms of shared parameters. If the shared parameters are within the same CPD, as in example 17.15, then the $(M+1)$st instance can involve at most one parameter from each partitions of shared parameters. In such a situation, the problem does not arise and we can use the average of the parameters to compute the probability of the next instance. However, if we have shared parameters across different CPDs (that is, entries in two or more CPDs share parameters), this problem can occur.

How do we solve this problem? The correct Bayesian prediction solution is to compute the average for the product of two (or more) parameters from the same posterior. This is essentially identical to the question of computing the probability of two or more test instances. See exercise 17.18. This solution, however, leads to many complications if we want to use the Bayesian posterior to answer queries about the distribution of the next instance. In particular, this probability no longer factorizes in the form of the original Bayesian network, and thus, we cannot use standard inference procedures to answer queries about future instances. For this reason, a pragmatic approximation is to use the expected parameters for each CPD and ignore the dependencies induced by the shared parameters. When the number of training samples is large, this solution can be quite a good approximation to the true predictive distribution. However, when the number of training examples is small, this assumption can skew the estimate; see exercise 17.18.

17.5.4 Hierarchical Priors ⋆

In our discussion of Bayesian methods for table-CPDs, we made the strong independence assumption (global and local parameter independence) to decouple the estimation of parameters.

Our discussion of shared parameters relaxed these assumptions by moving to the other end of the spectrum and forcing parameters to be identical. There are many situations where neither solution is appropriate.

Example 17.17

Using our favorite university *domain, suppose we learn a model from records of students, classes, teachers, and so on. Now suppose that we have data from several universities. Because of various factors, the data from each university have different properties. These differences can be due to the different population of students in each one (for example, one has more engineering oriented students while the other has more liberal arts students), somewhat different grade scale, or other factors. This leads to a dilemma. We can learn one model over all the data, ignoring the university specific bias. This allows us to pool the data to get a larger and more reliable data set. Alternatively, we can learn a different model for each university, or, equivalently, add a University variable that is the parent of many of the variables in the network. This approach allows us to tailor the parameters to each university. However, this flexibility comes at a price — learning parameters in one university does not help us learn better parameters from the data in the other university. This partitions the data into smaller sets, and in each one we need to learn from scratch that intelligent students tend to get an A in easy classes.* ■

Example 17.18

bigram model

A similar problem might arise when we consider learning dependencies in text domains. As we discussed in box 6.B, a standard model for word sequences is a bigram model, *which views the words as forming a Markov chain, where we have a conditional probability over the next word given the current word. That is, we want to learn* $P(W^{(t+1)} \mid W^{(t)})$ *where* W *is a random variable taking values from the dictionary of words. Here again, the context* $W^{(t)}$ *can definitely change our distribution over* $W^{(t+1)}$*, but we still want to share some information across these different conditional distributions; for example, the probability of common words, such as "the," should be high in almost all of the conditional distributions we learn.* ■

In both of these examples, we aim to learn a conditional distribution, say $P(Y \mid X)$. Moreover, we want the different conditional distributions $P(Y \mid x)$ to be similar to each other, yet not identical. Thus, the Bayesian learning problem assuming local independence (figure 17.11a) is not appropriate.

One way to bias the different distributions to be similar to each other is to have the same prior over them. If the prior is very strong, it will bias the different estimates to the same values. In particular, in the domain of example 17.18, we want the prior to bias both distributions toward giving high probability to frequent words. Where do we get such a prior? One simple ad hoc solution is to use the data to set the prior. For example, we can use the frequency of words in training set to construct a prior where more frequent words have a larger hyperparameter. Such an approach ensures that more frequent words have higher posterior in each of the conditional distributions, even if there are few training examples for that particular conditional distribution.

shrinkage

This approach (which is a slightly more Bayesian version of the *shrinkage* of exercise 6.2) works fairly well, and is often used in practice. However, it seems to contradict the whole premise of the Bayesian framework: a prior is a distribution that we formulate over our parameters *prior* to seeing the data. This approach also leaves unaddressed some important questions, such as determining the relative strength of the prior based on the amount of data used to compute it. A more coherent and general approach is to stay within the Bayesian framework, and to

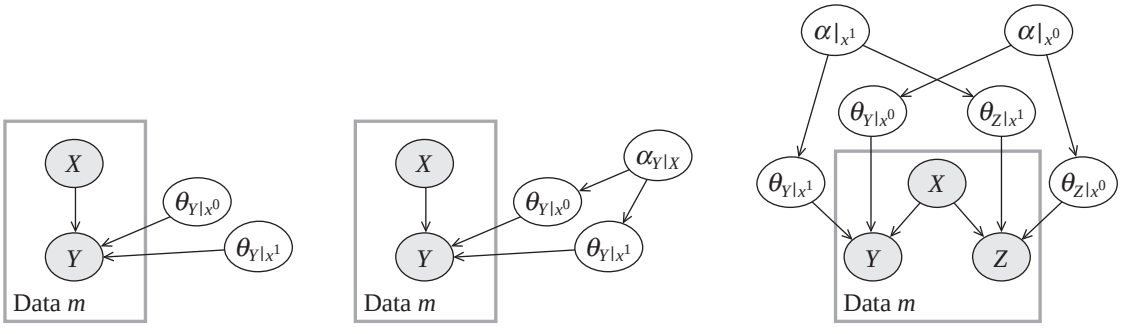


Figure 17.11 Independent and hierarchical priors. (a) A plate model for $P(Y \mid X)$ under assumption of parameter independence. (b) A plate model for a simple hierarchical prior for the same CPD. (c) A plate model for two CPDs $P(Y \mid X)$ and $P(Z \mid X)$ that respond similarly to X.

hierarchical Bayes

introduce explicitly our uncertainty about the joint prior as part of the model. Just as we introduced random variables to denote parameters of CPDS, we now take one step further and introduce a random variable to denote the hyperparameters. The resulting model is called a *hierarchical Bayesian model*. It uses a factored probabilistic model to describe our prior. This idea, of specifically defining a probabilistic model over our priors, can be used to define rich priors in a broad range of settings. Exercise 17.7 provides another example.

Figure 17.11b shows a simple example, where we have a variable that is the parent of both $\boldsymbol{\theta}_{Y|x^0}$ and $\boldsymbol{\theta}_{Y|x^1}$. As a result, these two parameters are no longer independent in the prior, and consequently in the posterior. Intuitively, the effect of the prior will be to shift both $\boldsymbol{\theta}_{Y|x^0}$ and $\boldsymbol{\theta}_{Y|x^1}$ to be closer to each other. However, as usual when using priors, the effect of the prior diminishes as we get more data. In particular, as we have more data for the different contexts x^1 and x^0, the effect of the prior will gradually decrease. Thus, the hierarchical priors (as for all priors) are particularly useful in the sparse-data regime.

How do we represent the distribution over the hyperparameter $P(\vec{\alpha})$? One option is to create a prior where each component α_y is governed by some distribution, say a Gamma distribution (recall that components are strictly positive). That is,

$$P(\vec{\alpha}) = \prod_y P(\alpha_y),$$

Gamma distribution

where $P(\alpha_y) \sim \text{Gamma}(\mu_y)$ is a *Gamma distribution* with (hyper-)hyperparameter μ_y. The other option is to write $\vec{\alpha}$ as the product of equivalent sample size N_0 with a probability distribution p_0, the first governed by a Gamma distribution and the other by a Dirichlet distribution. (These two representations are actually closely related; see box 19.E.)

Moreover, the same general idea can be used in broader ways. In this example, we used a hierarchical prior to relate two (or more) conditional distributions in the same CPD. We can similarly relax the global parameter independence assumption and introduce dependencies between the parameters for two (or more) CPDs. For example, if we believe that two variables

Y and Z depend on X in a similar (but not identical) way, then we introduce a common prior on $\boldsymbol{\theta}_{Y|x^0}$ and $\boldsymbol{\theta}_{Z|x^0}$, and similarly another common prior for $\boldsymbol{\theta}_{Y|x^1}$ and $\boldsymbol{\theta}_{Z|x^1}$; see figure 17.11c.

The idea of a hierarchical structure can also be extended to additional levels of a hierarchy. For example, in the case of similar CPDs, we might argue that there is a similarity between the distributions of Y and Z given x^0 and x^1. If we believe that this similarity is weaker than the similarity between the distributions of the two variables, we can introduce another hierarchy layer to relate the hyperparameters $\alpha_{\cdot|x^0}$ and $\alpha_{\cdot|x^1}$. To do so, we might introduce hyper-hyperparameters μ that specify a joint prior over $\alpha_{\cdot|x^0}$ and $\alpha_{\cdot|x^1}$.

The notion of hierarchical priors can be readily applied to other types of CPDs. For example, if X is a Gaussian variable without parents, then, as we saw in exercise 17.8, a conjugate prior for the mean of X is simply a Gaussian prior on the mean parameter. This observation suggests that we can easily create a hierarchical prior that relates X and another Gaussian variable Y, by having a common Gaussian over the means of the variables. Since the distribution over the hyperparameter is a Gaussian, we can easily extend the hierarchy upward. Indeed, hierarchical priors over Gaussians are often used to model the dependencies of parameters of related populations. For example, we might have a Gaussian over the SAT score, where one level of the hierarchy corresponds to different classes in the same school, the next one to different schools in the same district, the following to different districts in the same state, and so on.

The framework of hierarchical priors gives us a flexible language to introduce dependencies in the priors over parameters. Such dependencies are particularly useful when we have small amount of examples relevant to each parameter but many such parameters that we believe are reasonably similar. In such situations, hierarchical priors "spread" the effect of the observations between parameters with shared hyperparameters.

One question we did not discuss is how to perform learning with hierarchical priors. As with previous discussions of Bayesian learning, we want to compute expectations with respect to the posterior distribution. Since we relaxed the global and local independence assumptions, the posterior distribution no longer decomposes into a product of independent terms. From the perspective of the desired behavior, this lack of independence is precisely what we wanted to achieve. However, it implies that we need to deal with a much harder computational task. In fact, when introducing the hyperparameters as a variable into the model, we transformed our learning problem into one that includes a hidden variable. To address this setting, we therefore need to apply methods for Bayesian learning with hidden variables; see section 19.3 for a discussion of such methods.

Box 17.E — Concept: Bag-of-Word Models for Text Classification. *Consider the problem of* text classification*: classifying a document into one of several categories (or classes). Somewhat surprisingly, some of the most successful techniques for this problem are based on viewing the document as an unordered* bag of words*, and representing the distribution of this bag in different categories. Most simply, the distribution is encoded using a naive Bayes model. Even in this simple approach, however, there turn out to be design choices that can make important differences to the performance of the model.*

text classification

bag of words

Our first task is to represent a document by a set of random variables. This involves various processing steps. The first removes various characters such as punctuation marks as well as words that are viewed as having no content (such as "the," "and," and so on). In addition, most applica-

tions use a variety of techniques to map words in the document to canonical words in a predefined dictionary $\mathcal{D}$ (for example, replace "apples," "used," and "running" with "apple," "use," and "run" respectively). Once we finish this processing step, there are two common approaches to defining the features that describe the document.

Bernoulli naive Bayes

In the Bernoulli naive Bayes *model, we define a binary attribute (feature) X_i to denote whether the i'th dictionary word w_i appears in the document. This representation assumes that we care only about the presence of a word, not about how many times it appeared. Moreover, when applying the naive Bayes classifier with this representation we assume that the appearance of one word in the document is independent of the appearance of another (given the document's topic). When learning a naive Bayes classifier with this representation, we learn frequency (over a document) of encountering each dictionary word in documents of specific categories — for example, the probability that the word "ball" appears in a document of category "sports." We learn such a parameter for each pair (dictionary word, category).*

multinomial naive Bayes

In the multinomial naive Bayes *model, we define attribute to describe the specific sequence of words in the document. The variable X_i denotes which dictionary word appeared the ith word in the document. Thus, each X_i can take on many values, one for each possible word. Here, when we use the naive Bayes classifier, we assume that the choice of word in position i is independent of the choice of word in position j (again, given the document's topic). This model leads to a complication, since the distribution over X_i is over all words in the document, which requires a large number of parameters. Thus, we further assume that the probability that a particular word is used in position i does not depend on i; that is, the probability that $X_i = w$ (given the topic) is the same as the probability that $X_j = w$. In other words, we use parameter sharing between $P(X_i \mid C)$ and $P(X_j \mid C)$. This implies that the total number of parameters is again one for each (dictionary word, category).*

In both models, we might learn that the word "quarterback" is much more likely in documents whose topic is "sports" than in documents whose topic is "economics," the word "bank" is more associated with the latter subjects, and the word "dollar" might appear in both. Nevertheless, the two models give rise to quite different distributions. Most notably, if a word w appears in several different positions in the document, in the Bernoulli model the number of occurrences will be ignored, while in the multinomial model we will multiply the probability $P(w \mid C)$ several times. If this probability is very small in one category, the overall probability of the document given that category will decrease to reflect the number of occurrences. Another difference is in how the document length plays a role here. In the Bernoulli model, each document is described by exactly the same number of variables, while the multinomial model documents of different lengths are associated with a different number of random variables.

The plate model provides a compact and elegant way of making explicit the subtle distinctions between these two models. In both models, we have two different types of objects — documents, and individual words in the documents. Document objects d are associated with the attribute T, representing the document topic. However, the notion of "word objects" is different in the different models.

In the Bernoulli naive Bayes model, our words correspond to words in some dictionary (for example, "cat," "computer," and so on). We then have a binary-valued attribute $A(d, w)$ for each document d and dictionary word w, which takes the value true if the word w appears in the document d. We can model this case using a pair of intersecting plates, one for documents and the other for dictionary words, as shown in figure 17.E.1a.

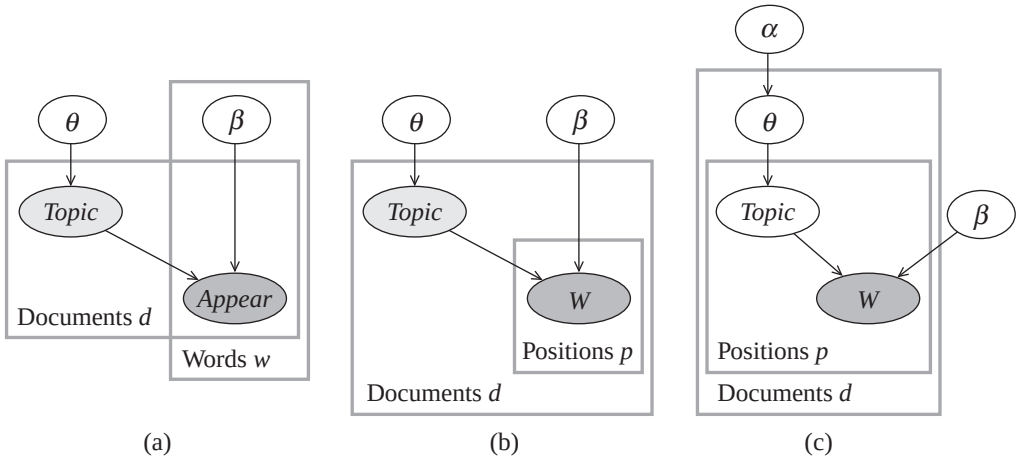


Figure 17.E.1 — Different plate models for text (a) Bernoulli naive Bayes; (b) Multinomial naive Bayes. (c) Latent Dirichlet Allocation.

In the multinomial naive Bayes model, our word objects correspond not to dictionary words, but to word positions P within the document. Thus, we have an attribute W of records representing pairs (D, P), where D is a document and P is a position within it (that is, first word, second word, and so on). This attribute takes on values in the space of dictionary words, so that $W(d, p)$ is the random variable whose value is the actual dictionary word in position p in document d. However, all of these random variables are generated from the same multinomial distribution, which depends on the document topic. The appropriate plate model is shown in figure 17.E.1b.[1]

The plate representation of the two models makes explicit the fact that the Bernoulli parameter $\beta_W[w]$ in the Bernoulli model is different for different words, whereas in the multinomial model, the parameter β_W is the same for all positions within the document. Empirical evidence suggests that the multinomial model is, in general, more successful than the Bernoulli.

In both models, the parameters are estimated from data, and the resulting model used for classifying new documents. The parameters for these models measure the probability of a word given a topic, for example, the probability of "bank" given "economics." For common words, such probabilities can be assessed reasonably well even from a small number of training documents. However, as the number of possible words is enormous, Bayesian parameter estimation is used to avoid overfitting, especially ascribing probability zero to words that do not appear in the training set. With Bayesian estimation, we can learn a naive Bayes model for text from a fairly small corpus, whereas

1. Note that, as defined in section 6.4.1.2, a skeleton for a plate model specifies a fixed set of objects for each class. In the multinomial plate model, this assumption implies that we specify a fixed set of word positions, which applies to all documents. In practice, however, documents have different lengths, and so we would want to allow a different set of word positions for each document. Thus, we want the set $\mathcal{O}^{\kappa}[p]$ to depend on the specific document d. When plates are nested, as they are in this case, we can generalize our notion of skeleton, allowing the set of objects in a nested plate Q_1 to depend on the index of an enclosing plate Q_2.

more realistic models of language are generally much harder to estimate correctly. This ability to define reasonable models with a very small number of parameters, which can be acquired from a small amount of training data, is one of the key advantages of the naive Bayes model.

latent Dirichlet allocation

We can also define much richer representations that capture more fine-grained structure in the distribution. These models are often easily viewed in the plate representation. One such model, shown in figure 17.E.1c, is the latent Dirichlet allocation *(LDA) model, which extends the multinomial naive Bayes model. As in the multinomial naive Bayes model, we have a set of topics associated with a set of multinomial distributions $\boldsymbol{\theta}_W$ over words. However, in the LDA model, we do not assume that an entire document is about a single topic. Rather, we assume that each document d is associated with a continuous mixture of topics, defined using parameters $\boldsymbol{\theta}(d)$. These parameters are selected independently from each document d, from a Dirichlet distribution parameterized by a set of hyperparameters $\boldsymbol{\alpha}$. The word in position p in the document d is then selected by first selecting a topic Topic$(d, p) = t$ from the mixture $\boldsymbol{\theta}(d)$, and then selecting a specific dictionary word from the multinomial $\boldsymbol{\beta}_t$ associated with the chosen topic t. The LDA model generally provides much better results (in measures related to log-likelihood of test data) than other unsupervised clustering approaches to text. In particular, owing to the flexibility of assigning a mixture of topics to a document, there is no problem with words that have low probability relative to a particular topic; thus, this approach largely avoids the overfitting problems with the two naive Bayes models described earlier.*

17.6 Generalization Analysis ⋆

One intuition that permeates our discussion is that more training instances give rise to more accurate parameter estimates. This intuition is supported by our empirical results. In this section, we provide some formal analysis that supports this intuition. This analysis also allows us to quantify the extent to which the error in our estimates decreases as a function of the number of training samples, and increases as a function of the number of parameters we want to learn or the number of variables in our networks.

We begin with studying the asymptotic behavior of our estimator at the large-sample limit. We then provide a more refined analysis that studies the error as a function of the number of samples.

17.6.1 Asymptotic Analysis

We start by considering the asymptotic behavior of the maximum likelihood estimator. In this case, our analysis of section 17.2.5 provides an immediate conclusion: At the large sample limit, $\hat{P}_{\mathcal{D}}$ approaches P^*; thus, as the number of samples grows, $\hat{\boldsymbol{\theta}}$ approaches $\boldsymbol{\theta}^*$ — the projection of P^* onto the parametric family.

A particular case of interest arises when $P^*(\mathcal{X}) = P(\mathcal{X} : \boldsymbol{\theta}^*)$, that is, P^* is representable in the parametric family. Then, we have that $\hat{\boldsymbol{\theta}} \to \boldsymbol{\theta}^*$ as $M \to \infty$. An estimator with this property is called *consistent estimator*. In general, maximum likelihood estimators are consistent.

consistent estimator

We can make this analysis even more precise. Using equation (17.9), we can write

$$\log \frac{P(\mathcal{D} \mid \hat{\boldsymbol{\theta}})}{P(\mathcal{D} \mid \boldsymbol{\theta})} = M \left(\boldsymbol{I}\boldsymbol{D}(\hat{P}_{\mathcal{D}} \| P_{\boldsymbol{\theta}}) - \boldsymbol{I}\boldsymbol{D}(\hat{P}_{\mathcal{D}} \| P_{\hat{\theta}}) \right).$$

This equality implies that the likelihood function is sharply peaked: the decrease in the likelihood for parameters that are not the MLE is exponential in M. Of course, when we change M the data set $\mathcal{D}$ and hence the distribution $\hat{P}_{\mathcal{D}}$ also change, and thus this result does not guarantee exponential decay in M. However, for sufficiently large M, $\hat{P}_{\mathcal{D}} \to P^*$. Thus, the difference in log-likelihood of different choices of $\boldsymbol{\theta}$ is roughly M times their distance from P^*:

$$\log \frac{P(\mathcal{D} \mid \hat{\boldsymbol{\theta}})}{P(\mathcal{D} \mid \boldsymbol{\theta})} \approx M \left(\boldsymbol{I}\boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}}) - \boldsymbol{I}\boldsymbol{D}(P^* \| P_{\hat{\boldsymbol{\theta}}}) \right).$$

The terms on the right depend only on M and not on $\hat{P}_{\mathcal{D}}$. Thus, we conclude that for large values of M, the likelihood function approaches a delta function, for which all values are virtually 0 when compared to the maximal value at $\boldsymbol{\theta}^*$, the M-projection of P^*.

To summarize, this argument basically allows us to prove the following result, asserting that the Bayesian estimator is *consistent*:

Theorem 17.2 *Let P^* be the generating distribution, let $P(\cdot \mid \theta)$ be a parametric family of distributions, and let $\theta^* = \arg\min_\theta \boldsymbol{I}\boldsymbol{D}(P^* \| P(\cdot \mid \theta))$ be the M-projection of P^* on this family. Then*

$$\lim_{M \to \infty} P(\cdot \mid \hat{\theta}) = P(\cdot \mid \theta^*)$$

almost surely.

That is, when M grows larger, the estimated parameters describe a distribution that is close to the distribution with the "optimal" parameters in our parametric family.

Is our Bayesian estimator consistent? Recall that equation (17.11) shows that the Bayesian estimate with a Dirichlet prior is an interpolation between the MLE and the prior prediction. The interpolation weight depends on the number of samples M: as M grows, the weight of the prior prediction diminishes and disappears in the limit. Thus, we can conclude that Bayesian learning with Dirichlet priors is also consistent.

17.6.2 PAC-Bounds

This consistency result guarantees that, at the large sample limit, our estimate converges to the true distribution. Though a satisfying result, its practical significance is limited, since in most cases we do not have access to an unlimited number of samples. Hence, it is also important to evaluate the quality of our learned model as a function of the number of samples M. This type of analysis allows us to know how much to trust our learned model as a function of M; or, from the other side, how many samples we need to acquire in order to obtain results of a given quality. Thus, using relative entropy to the true distribution as our notion of solution quality, we use *PAC-bound* analysis (as in box 16.B) to bound $\boldsymbol{I}\boldsymbol{D}(P^* \| \hat{P})$ as a function of the number of data samples M.

PAC-bound

17.6.2.1 Estimating a Multinomial

We start with the simplest case, which forms the basis for our more extensive analysis. Here, our task is to estimate the multinomial parameters governing the distribution of a single random variable. This task is relevant to many disciplines and has been studied extensively. A basic tool used in this analysis are the *convergence bounds* described in appendix A.2.

convergence bound

Consider a data set $\mathcal{D}$ defined as a set of M IID Bernoulli random variables $\mathcal{D} = \{X[1], \ldots, X[M]\}$, where $P^*(X[m] = x^1) = p^*$ for all m. Note that we are now considering $\mathcal{D}$ itself to be a stochastic event (a random variable), sampled from the distribution $(P^*)^M$. Let $\hat{p}_\mathcal{D} = \frac{1}{M}\sum_m X[m]$. Then an immediate consequence of the *Hoeffding bound* (theorem A.3) is

Hoeffding bound

$$P_M(|\hat{p}_\mathcal{D} - p^*| > \epsilon) \leq 2e^{-2M\epsilon^2},$$

where P_M is a shorthand for $P_{\mathcal{D}\sim(P^*)^M}$. Thus, the probability that the MLE $\hat{p}_\mathcal{D}$ deviates from the true parameter by more than ϵ is bounded from above by a function that decays exponentially in M.

As an immediate corollary, we can prove a PAC-bound on estimating p^*:

Corollary 17.1

Let $\epsilon, \delta > 0$, and let

$$M > \frac{1}{2\epsilon^2}\log\frac{2}{\delta}.$$

Then $P_M(|\hat{p} - p^*| \leq \epsilon) \geq 1 - \delta$, where P_M, as before, is a probability over data sets $\mathcal{D}$ of size M sampled IID from P^*.

PROOF

$$\begin{aligned} P_M(|\hat{p} - p^*| \leq \epsilon) &= 1 - P_M(\hat{p} - p^* > \epsilon) - P_M(\hat{p} - p^* < -\epsilon) \\ &\geq 1 - 2e^{-2M\epsilon^2} \geq 1 - \delta. \end{aligned}$$

■

The number of data instances M required to obtain a PAC-bound grows quadratically in the error $1/\epsilon$, and logarithmically in the confidence $1/\delta$. For example, setting $\epsilon = 0.05$ and $\delta = 0.01$, we get that we need $M \geq 1059.66$. That is, we need a bit more than $1,000$ samples to confidently estimate the probability of an event to within 5 percent error.

This result allows us to bound the absolute value of the error between the parameters. We, however, are interested in the *relative entropy* between the two distributions. Thus, we want to bound terms of the form $\log\frac{p^*}{\hat{p}}$.

relative entropy

Lemma 17.1

For P_M defined as before:

$$P_M(\log\frac{p^*}{\hat{p}} > \epsilon) \leq e^{-2Mp^2\epsilon^2\frac{1}{(1+\epsilon)^2}}.$$

PROOF The proof relies on the following fact:

$$\text{If } \epsilon \leq x \leq y \leq 1 \text{ then } \quad (\log y - \log x) \leq \frac{1}{\epsilon}(y - x). \tag{17.22}$$

Now, consider some ϵ'. If $p^* - \hat{p} \leq \epsilon'$ then $\hat{p} > p^* - \epsilon'$. Applying equation (17.22), we get that $\log \frac{p^*}{\hat{p}} \leq \frac{\epsilon'}{p^* - \epsilon'}$. Setting $\epsilon' = \frac{\epsilon p^*}{1+\epsilon}$, and taking the contrapositive, we conclude that, if $\log \frac{p^*}{\hat{p}} > \epsilon$ then $p^* - \hat{p} > \frac{\epsilon p^*}{1+\epsilon}$. Using the Hoeffding bound to bound the probability of the latter event, we derive the desired result. ■

This analysis applies to a binary-valued random variable. We now extend it to the case of a multivalued random variable. The result provides a bound on the relative entropy between $P^*(X)$ and the maximum likelihood estimate for $P(X)$, which is simply its empirical probability $\hat{P}_{\mathcal{D}}(X)$.

Proposition 17.7 *Let $P^*(X)$ be a discrete distribution such that $P^*(x) \geq \lambda$ for all $x \in Val(X)$. Let $\mathcal{D} = \{X[1], \ldots, X[M]\}$ consist of M IID samples of X. Then*

$$P_M(\mathbf{I}\mathbf{D}(P^*(X) \| \hat{P}_{\mathcal{D}}(X)) > \epsilon) \leq |Val(X)| e^{-2M\lambda^2\epsilon^2 \frac{1}{(1+\epsilon)^2}}.$$

Proof We want to bound the error

$$\mathbf{I}\mathbf{D}(P^*(X) \| \hat{P}_{\mathcal{D}}(X)) = \sum_x P^*(x) \log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)}.$$

This expression is a weighted average of log-ratios of the type we bounded in lemma 17.1. If we bound each of the terms in this average by ϵ, we can obtain a bound for the weighted average as a whole. That is, we say that a data set is well behaved if, for each x, $\log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} \leq \epsilon$. If the data set is well behaved, we have a bound of ϵ on the term for each x, and therefore an overall bound of ϵ for the entire relative entropy.

With what probability is our data set not well behaved? It suffices that there is one x for which $\log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} > \epsilon$. We can provide an upper bound on this probability using the *union bound*, which bounds the probability of the union of a set of events as the sum of the probabilities of the individual events:

$$P\left(\exists x, \log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} > \epsilon\right) \leq \sum_x P\left(\log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} > \epsilon\right).$$

The union bound is an overestimate of the probability, since it essentially represents the case where the different "bad" events are disjoint. However, our focus is on the situation where these events are unlikely, and the error due to such overcounting is not significant.

Formalizing this argument, we obtain:

$$\begin{aligned}
P_M(\mathbf{I}\mathbf{D}(P^*(X) \| \hat{P}_{\mathcal{D}}(X)) > \epsilon) &\leq P_M\left(\exists x \ : \ \log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} > \epsilon\right) \\
&\leq \sum_x P_M\left(\log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)} > \epsilon\right) \\
&\leq \sum_x e^{-2MP^*(x)^2\epsilon^2 \frac{1}{(1+\epsilon)^2}} \\
&\leq |Val(X)| e^{-2M\lambda^2\epsilon^2 \frac{1}{(1+\epsilon)^2}},
\end{aligned}$$

where the second inequality is derived from the union bound, the third inequality from lemma 17.1, and the final inequality from the assumption that $P^*(x) \geq \lambda$. ■

This result provides us with an error bound for estimating the distribution of a random variable. We can now easily translate this result to a PAC-bound:

Corollary 17.2

Assume that $P^*(x) \geq \lambda$ for all $x \in Val(X)$. For $\epsilon, \delta > 0$, let

$$M \geq \frac{1}{2}\frac{1}{\lambda^2}\frac{(1+\epsilon)^2}{\epsilon^2}\log\frac{|Val(X)|}{\delta}.$$

Then $P_M(\mathbf{D}(P^*(X)\|\hat{P}_{\mathcal{D}}(X)) \leq \epsilon) \geq 1-\delta$.

As with the binary-valued case, the number of samples grows quadratically with $\frac{1}{\epsilon}$ and logarithmically with $\frac{1}{\delta}$. Here, however, we also have a quadratic dependency on $\frac{1}{\lambda}$. The value λ is a measure of the "skewness" of the distribution P^*. This dependence is not that surprising; we expect that, if some values of X have small probability, then we need many more samples to get a good approximation of their probability. Moreover, underestimates of $\hat{P}_{\mathcal{D}}(x)$ for such events can lead to a big error in estimating $\log \frac{P^*(x)}{\hat{P}_{\mathcal{D}}(x)}$. Intuitively, we might suspect that when $P^*(x)$ is small, it is harder to estimate, but at the same time it also less crucial for the total error. Although it is possible to use this intuition to get a better estimation (see exercise 17.19), the asymptotic dependence of M on $\frac{1}{\lambda}$ remains quadratic.

17.6.2.2 Estimating a Bayesian Network

We now consider the problem of learning a Bayesian network. Suppose that P^* is consistent with a Bayesian network $\mathcal{G}$ and that we learned parameters $\boldsymbol{\theta}$ for $\mathcal{G}$ that define a distribution $P_{\boldsymbol{\theta}}$. Using theorem 8.5, we have that

$$\mathbf{D}(P^*\|P_{\boldsymbol{\theta}}) = \sum_i \mathbf{D}(P^*(X_i \mid \mathrm{Pa}_i^{\mathcal{G}})\|P_{\boldsymbol{\theta}}(X_i \mid \mathrm{Pa}_i^{\mathcal{G}})),$$

where (as shown in appendix A.1.3.2), we have that

$$\mathbf{D}(P^*(X \mid \boldsymbol{Y})\|P(X \mid \boldsymbol{Y})) = \sum_{\boldsymbol{y}} P^*(\boldsymbol{y})\mathbf{D}(P^*(X \mid \boldsymbol{y})\|P(X \mid \boldsymbol{y})).$$

Thus, as we might expect, the error is a sum of errors in estimating the conditional probabilities.

This error term, however, makes the strong assumption that our generating distribution P^* is consistent with our target class — those distributions representable using the graph $\mathcal{G}$. This assumption is usually not true in practice. When this assumption is false, the given network structure limits the ability of the learning procedure to generalize. For example, if we give a learning procedure a graph where X and Y are independent, then no matter how good our learning procedure, we cannot achieve low generalization error if X and Y are strongly dependent in P^*. More broadly, if the given network structure is inaccurate, then there is inherent error in the learned distribution that the learning procedure cannot overcome.

One approach to deal with this problem is to assume away the cases where P^* does not conform with the given structure $\mathcal{G}$. This solution, however, makes the analysis brittle and of

little relevance to real-life scenarios. An alternative solution is to relax our expectations from the learning procedure. Instead of aiming for the error to become very small, we might aim to show that the error is not far away from the inherent error that our procedure must incur due to the limitations in expressive power of the given network structure. In other words, rather than bounding the risk, we provide a bound on the *excess risk* (see box 16.B).

excess risk

More formally, let $\Theta[\mathcal{G}]$ be the set of all possible parameterizations for $\mathcal{G}$. We now define

$$\boldsymbol{\theta}^{\mathrm{opt}} = \arg \min_{\boldsymbol{\theta} \in \Theta[\mathcal{G}]} \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}}).$$

That is, $\boldsymbol{\theta}^{\mathrm{opt}}$ is the best result we might expect the learning procedure to return. (Using the terminology of section 8.5, $\boldsymbol{\theta}^{\mathrm{opt}}$ is the *M-projection* of P^* on the family of distributions defined by $\mathcal{G}, \Theta[\mathcal{G}]$.)

M-projection

The distance $\boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}})$ reflects the minimal error we might achieve with networks with the structure $\mathcal{G}$. Thus, instead of defining "success" for our learning procedure in terms of obtaining low values for $\boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}})$ (a goal which may not be achievable), we aim to obtain low values for $\boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}}) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}})$.

What is the form of this error term? By solving for $P_{\boldsymbol{\theta}^{\mathrm{opt}}}$ and using basic manipulations we can define it in precise terms.

Theorem 17.3

Let $\mathcal{G}$ be a network structure, let P^* be a distribution, and let $P_{\boldsymbol{\theta}} = (\mathcal{G}, \boldsymbol{\theta})$ be a distribution consistent with $\mathcal{G}$. Then:

$$\boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}}) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}}) = \sum_i \boldsymbol{D}(P^*(X_i \mid \mathrm{Pa}_i^{\mathcal{G}}) \| P_{\boldsymbol{\theta}}(X_i \mid \mathrm{Pa}_i^{\mathcal{G}})).$$

The proof is left as an exercise (exercise 17.20).

This theorem shows that **the error in our learned network decomposes into two components. The first is the error inherent in $\mathcal{G}$, and the second the error due to inaccuracies in estimating the conditional probabilities for parameterizing $\mathcal{G}$.** This theorem also shows that, in terms of error analysis, the treatment of the general case leads to exactly the same terms that we had to bound when we made the assumption that P^* was consistent with $\mathcal{G}$. Thus, in this learning task, the analysis of the inconsistent case is not more difficult than the analysis of the consistent case. As we will see in later chapters, this situation is not usually the case: we can often provide bounds for the consistent case, but not for the inconsistent case.

To continue the analysis, we need to bound the error in estimating conditional probabilities. The preceding treatment showed that we can bound the error in estimating marginal probabilities of a variable or a group of variables. How different is the estimate of conditional probabilities from that of marginal probabilities? It turns out that the two are easily related.

Lemma 17.2

Let P and Q be two distributions on X and Y. Then

$$\boldsymbol{D}(P(X \mid Y) \| Q(X \mid Y)) \leq \boldsymbol{D}(P(X, Y) \| Q(X, Y)).$$

See exercise 17.21.

As an immediate corollary, we have that

$$\boldsymbol{D}(P^* \| P) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}}) \leq \sum_i \boldsymbol{D}(P^*(X_i, \mathrm{Pa}_i^{\mathcal{G}}) \| P(X_i, \mathrm{Pa}_i^{\mathcal{G}})).$$

Thus, we can use proposition 17.7 to bound the error in estimating $P(X_i, \mathrm{Pa}_i^{\mathcal{G}})$ for each X_i (where we treat $X_i, \mathrm{Pa}_i^{\mathcal{G}}$ as a single variable) and derive a bound on the error in estimating the probability of the whole network.

Theorem 17.4

Let $\mathcal{G}$ be a network structure, and P^* a distribution consistent with some network $\mathcal{G}^*$ such that $P^*(x_i \mid \mathrm{pa}_i^{\mathcal{G}^*}) \geq \lambda$ for all i, x_i, and $\mathrm{pa}_i^{\mathcal{G}^*}$. If P is the distribution learned by maximum likelihood estimate for $\mathcal{G}$, then

$$P\left(\boldsymbol{D}(P^* \| P) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}}) > n\epsilon\right) \leq nK^{d+1} e^{-2M\lambda^{2(d+1)}\epsilon^2 \frac{1}{(1+\epsilon)^2}},$$

where K is the maximal variable cardinality and d is the maximum number of parents in $\mathcal{G}$.

Proof The proof uses the union bound

$$P\left(\boldsymbol{D}(P^* \| P) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}}) > n\epsilon\right) \leq \sum_i P\left(\boldsymbol{D}(P^*(X_i, \mathrm{Pa}_i^{\mathcal{G}}) \| P(X_i, \mathrm{Pa}_i^{\mathcal{G}})) > \epsilon\right)$$

with application of proposition 17.7 to bound the probability of each of these latter events. The only technical detail we need to consider is to show that if conditional probabilities in P^* are always larger than λ, then $P^*(x_i, \mathrm{pa}_i^{\mathcal{G}}) \geq \lambda^{|\mathrm{Pa}_i^{\mathcal{G}}|+1}$; see exercise 17.22. ■

This theorem shows that we can indeed learn parameters that converge to the optimal ones as the number of samples grows. As with previous bounds, the number of samples we need is

Corollary 17.3

Under the conditions of theorem 17.4, if

$$M \geq \frac{1}{2} \frac{1}{\lambda^{2(d+1)}} \frac{(1+\epsilon)^2}{\epsilon^2} \log \frac{nK^{d+1}}{\delta},$$

then

$$P\left(\boldsymbol{D}(P^* \| P) - \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}^{\mathrm{opt}}}) < n\epsilon\right) > 1 - \delta.$$

As before, the number of required samples grows quadratically in $\frac{1}{\epsilon}$. Conversely, we expect the error to decrease roughly with $\frac{1}{\sqrt{M}}$, which is commensurate with the behavior we observe in practice (for example, see figure 17.C.2 in box 17.C). We see also that λ and d play a major role in determining M. In practice, we often do not know λ in advance, but such analysis allows us to provides guarantees under the assumption that conditional probabilities are not too small. It also allows us to predict the improvement in error (or at least in our upper bound on it) that we would obtain if we add more samples.

Note that in this analysis we "allow" the error to grow with n as we consider $\boldsymbol{D}(P^* \| P) > n\epsilon$. The argument is that, as we add more variables, we expect to incur some prediction error on each one.

Example 17.19

Consider the network where we have n independent binary-valued variables $X_1, \ldots, X_n$. In this case, we have n independent Bernoulli estimation problems, and would expect a small number of samples to suffice. Indeed, we can obtain an ϵ-close estimate to each of them (with high-probability) using the bound of lemma 17.1. However, the overall relative entropy between P^* and $\hat{P}_{\mathcal{D}}$ over the joint space $X_1, \ldots, X_n$ will grow as the sum of the relative entropies between the individual marginal distributions $P^*(X_i)$ and $\hat{P}_{\mathcal{D}}(X_i)$. Thus, even if we perform well at predicting each variable, the total error will scale linearly with n. ■

Thus, our formulation of the bound in corollary 17.3 is designed to separate out this "inevitable" linear growth in the error from any additional errors that arise from the increase in dimensionality of the distribution to be estimated.

We provided a theoretical analysis for the generalization error of maximum likelihood estimate. A natural question is to carry similar analysis when we use Bayesian estimates. Intuitively, the asymptotic behavior (for $M \to \infty$) will be similar, since the two estimates are asymptotically identical. For small values of M we do expect to see differences, since the Bayesian estimate is smoother and cannot be arbitrarily small and thus the relative entropy is bounded. See exercise 17.23 for an analysis of the Bayesian estimate.

17.7 Summary

In this chapter we examined parameter estimation for Bayesian networks when the data are complete. This is the simplest learning task for Bayesian networks, and it provides the basis for the more challenging learning problems we examine in the next chapters. We discussed two approaches for estimation: MLE and Bayesian prediction. Our primary emphasis here was on table-CPDs, although the ideas generalize to other representations. We touched on a few of these.

As we saw, a central concept in both approaches is the likelihood function, which captures how the probability of the data depends on the choice of parameters. A key property of the likelihood function for Bayesian networks is that it decomposes as a product of local likelihood functions for the individual variables. If we use table-CPDs, the likelihood decomposes even further, as a product of the likelihoods for the individual multinomial distributions $P(X \mid \mathrm{pa}_{X_i})$. This decomposition plays a central role in both maximum likelihood and Bayesian estimation, since it allows us to decompose the estimation problem and treat each of these CPDs or even each of the individual multinomials separately.

When the local likelihood has sufficient statistics, then learning is viewed as mapping values of the statistics to parameters. For networks with discrete variables, these statistics are counts of the form $M[x_i, \mathrm{pa}_{X_i}]$. Thus, learning requires us to collect these for each combination x_i, pa_{X_i} of a value of X_i and an instantiation of values to its parents. We can collect all of these counts in one pass through the data using a data structure whose size is proportional to the representation of the network, since we need one counter for each CPD entry.

Once we collect the sufficient statistics, the estimate of both methods is similar. The MLE estimate for table-CPDs has a simple closed form:

$$\hat{\theta}_{x_i|\mathrm{pa}_{X_i}} = \frac{M[x_i, \mathrm{pa}_{X_i}]}{M[\mathrm{pa}_{X_i}]}.$$

The Bayesian estimate is based on the use of a Dirichlet distribution, which is a conjugate prior to the multinomial distribution. In a conjugate prior, the posterior — which is proportional to the prior times the likelihood — has the same form as the prior. This property allows us to maintain posteriors in closed form. In particular, for a discrete Bayesian network with table-CPDs and a Dirichlet prior, the posterior of the local likelihood has the form

$$P(x_i \mid \mathrm{pa}_{X_i}, D) = \frac{M[x_i, \mathrm{pa}_{X_i}] + \alpha_{x_i|\mathrm{pa}_{X_i}}}{M[\mathrm{pa}_{X_i}] + \alpha_{\mathrm{pa}_{X_i}}}.$$

Since all we need in order to learn are the sufficient statistics, then we can easily adapt them to learn in an online setting, where additional training examples arrive. We simply store a vector of sufficient statistics, and update it as new instances are obtained.

In the more advanced sections, we saw that the same type of structure applies to other parameterizations in the exponential family. Each family defines the sufficient statistics we need to accumulate and the rules for finding the MLE parameters. We developed these rules for Gaussian CPDs, where again learning can be done using a closed-form analytical formula.

We also discussed networks where some of the parameters are shared, whether between CPDs or within a single CPD. We saw that the same properties described earlier — decomposition and sufficient statistics — allow us to provide an easy analysis for this setting. The likelihood function is now defined in terms of sufficient statistics that aggregate statistics from different parts of the network. Once the sufficient statistics are defined, the estimation procedures, whether MLE or Bayesian, are exactly the same as in the case without shared parameters.

Finally, we examined the theoretical foundations of learning. We saw that parameter estimates are asymptotically correct in the following sense. If the data are actually generated from the given network structure, then, as the number of samples increases, both methods converge to the correct parameter setting. If not, then they converge to the distribution with the given structure that is "closest" to the distribution from which the data were generated. We further analyzed the rate at which the estimates converge. As M grows, we see a *concentration phenomenon*; for most samples, the empirical distribution is in a close neighborhood of the true distribution. Thus, the chances of sampling a data set in which the MLE estimates are far from the true parameters decays exponentially with M. This analysis allowed us to provide a PAC-bound on the number of samples needed to obtain a distribution that is "close" to optimal.

concentration phenomenon

17.8 Relevant Literature

The foundations of maximum likelihood estimation and Bayesian estimation have a long history; see DeGroot (1989); Schervish (1995); Hastie et al. (2001); Bishop (2006); Bernardo and Smith (1994) for some background material. The thumbtack example is adapted from Howard (1970).

Heckerman (1998) and Buntine (1994, 1996) provide excellent tutorials and reviews on the basic principles of learning Bayesian networks from data, as well as a review of some of the key references.

The most common early application of Bayesian network learning, and perhaps even now, is learning a naive Bayes model for the purpose of classification (see, for example, Duda and Hart 1973). Spiegelhalter and Lauritzen (1990) laid the foundation for the problem of learning general Bayesian networks from data, including the introduction of the global parameter independence assumption, which underlies the decomposability of the likelihood function. This development led to a stream of significant extensions, most notably by Buntine (1991); Spiegelhalter et al. (1993); Cooper and Herskovits (1992). Heckerman et al. (1995) defined the BDe prior and showed its equivalence to a combination of assumptions about the prior.

Many papers (for example, Spiegelhalter and Lauritzen 1990; Neal 1992; Buntine 1991; Diez 1993) have proposed the use of structured CPDs as an approach for reducing the number of parameters that one needs to learn from data. In many cases, the specific learning algorithms are derived from algorithms for learning conditional probability models for probabilistic

classification. The probabilistic derivation of tree-CPDs was performed by Buntine (1993) and introduced to Bayesian networks by Friedman and Goldszmidt (1998). The analysis of Bayesian learning of Bayesian networks with linear Gaussian dependencies was performed by Heckerman and Geiger (1995); Geiger and Heckerman (Geiger and Heckerman). Buntine (1994) emphasizes the important connection between the exponential family and the task of learning Bayesian networks. Bernardo and Smith (1994) describe conjugate priors for many distributions in the exponential family. Some material on nonparametric density estimation can be found in Hastie et al. (2001); Bishop (2006). Hofmann and Tresp (1995) use Parzen window to capture conditional distributions in continuous Bayesian networks. Imoto et al. (2003) learn semiparametric spline-regression models as CPDs in Bayesian networks. Rasmussen and Williams (2006) describe *Gaussian processes*, a state-of-the-art method for nonparametric estimation, which has also been used for estimating CPDs in Bayesian networks (Friedman and Nachman 2000).

Gaussian processes

Plate models as a representation for parameter sharing in learning were introduced by Gilks et al. (1994) and Buntine (1994). Hierarchical Bayesian models have a long history in statistics; see, for example, Gelman et al. (1995).

The generalization bounds for parameter estimation in Bayesian networks were first analyzed by Höffgen (1993), and subsequently improved and refined by Friedman and Yakhini (1996) and Dasgupta (1997).

Beinlich et al. (1989) introduced the ICU-Alarm network, which has formed the benchmark for numerous studies of Bayesian network learning.

17.9 Exercises

Exercise 17.1

Show that the estimate of equation (17.1) is the maximum likelihood estimate. Hint: differentiate the log-likelihood with respect to θ.

Exercise 17.2⋆

Derive the MLE for the multinomial distribution (example 17.5). Hint, maximize the log-likelihood function using a Lagrange coefficient to enforce the constraint $\sum_k \theta_k = 1$.

Exercise 17.3

Derive the MLE for Gaussian distribution (example 17.6). Solve the equations

$$\begin{aligned} \frac{\partial \log L(\mathcal{D} : \mu, \sigma)}{\partial \mu} &= 0 \\ \frac{\partial \log L(\mathcal{D} : \mu, \sigma)}{\partial \sigma^2} &= 0. \end{aligned}$$

Exercise 17.4

Derive equation (17.8) by differentiating the log-likelihood function and using equation (17.6) and equation (17.7).

Exercise 17.5⋆

In this exercise, we examine how to estimate a joint multivariate Gaussian. Consider two continuous variables X and Y, and assume we have a data set consisting of M samples $\mathcal{D} = \{\langle x[m], y[m]\rangle : m =$

$1, \ldots, M\}$. Show that the MLE estimate for a joint Gaussian distribution over X, Y is the Gaussian with mean vector $\langle \boldsymbol{E}_{\mathcal{D}}[X], \boldsymbol{E}_{\mathcal{D}}[Y] \rangle$, and covariance matrix

$$\Sigma_{X,Y} = \begin{pmatrix} \boldsymbol{Cov}_{\mathcal{D}}[X;X] & \boldsymbol{Cov}_{\mathcal{D}}[X;Y] \\ \boldsymbol{Cov}_{\mathcal{D}}[X;Y] & \boldsymbol{Cov}_{\mathcal{D}}[Y;Y] \end{pmatrix}.$$

Exercise 17.6⋆

Derive equation (17.10) by solving the integral $\int_0^1 \theta^k (1-\theta)^{M-k} d\theta$ for different values of k. (Hint: use integration by parts.)

Exercise 17.7⋆

mixture of Dirichlets

In this problem we consider the class of parameter priors defined as a *mixture of Dirichlets*. These comprise a richer class of priors than the single Dirichlet that we discussed in this chapter. A mixture of Dirichlets represents another level of uncertainty, where we are unsure about which Dirichlet distribution is a more appropriate prior for our domain. For example, in a simple coin-tossing situation, we might be uncertain whether the coin whose parameter we are trying to learn is a fair coin, or whether it is a biased one. In this case, our prior might be a mixture of two Dirichlet distributions, representing those two cases.

In this problem, our goal is to show that the family of mixture of Dirichlet priors is conjugate to the multinomial distribution; in other words, if our prior is a mixture of Dirichlets, and our likelihood function is multinomial, then our posterior is also a mixture of Dirichlets.

a. Consider the simple possibly biased coin setting described earlier. Assume that we use a prior that is a mixture of two Dirichlet (Beta) distributions: $P(\theta) = 0.95 \cdot \mathit{Beta}(5000, 5000) + 0.05 \cdot \mathit{Beta}(1, 1)$; the first component represents a fair coin (for which we have seen many imaginary samples), and the second represents a possibly-biased coin, whose parameter we know nothing about. Show that the expected probability of heads given this prior (the probability of heads averaged over the prior) is $1/2$. Suppose that we observe the data sequence $(H, H, T, H, H, H, H, H, H, H)$. Calculate the posterior over θ, $P(\theta \mid \mathcal{D})$. Show that it is also a 2-component mixture of Beta distributions, by writing the posterior in the form $\lambda^1 Beta(\alpha_1^1, \alpha_2^1) + \lambda^2 Beta(\alpha_1^2, \alpha_2^2)$. Provide actual numeric values for the different parameters $\lambda^1, \lambda^2, \alpha_1^1, \alpha_2^1, \alpha_1^2, \alpha_2^2$.

b. Now generalize your calculations from part (l) to the case of a mixture of d Dirichlet priors over a k-valued multinomial parameters. More precisely, assume that the prior has the form

$$P(\boldsymbol{\theta}) = \sum_{i=1}^{d} \lambda^i \mathit{Dirichlet}(\alpha_1^i, \ldots, \alpha_k^i),$$

and prove that the posterior has the same form.

Exercise 17.8⋆

We now consider a Bayesian approach for learning the mean of a Gaussian distribution. It turns out that in doing Bayesian inference with Gaussians, it is mathematically easier to use the precision $\tau = \frac{1}{\sigma^2}$ rather than the variance. Note that larger the precision, the narrower the distribution around the mean.

Suppose that we have M IID samples $x[1], \ldots, x[M]$ from $X \sim \mathcal{N}\left(\theta; \tau_X^{-1}\right)$. Moreover, assume that we know the value of τ_X. Thus, the unknown parameter θ is the mean. Show that if the prior $P(\theta)$ is $\mathcal{N}\left(\mu; \tau_\theta^{-1}\right)$, then the posterior $P(\theta \mid \mathcal{D})$ is $\mathcal{N}\left(\mu'; (\tau'_\theta)^{-1}\right)$ where

$$\begin{aligned} \tau'_\theta &= M\tau_X + \tau_\theta \\ \mu' &= \frac{M\tau_X}{\tau'_\theta} \boldsymbol{E}_{\mathcal{D}}[X] + \frac{\tau_\theta}{\tau'_\theta} \mu_0. \end{aligned}$$

Hint: Start by proving $\sum_m (x[m] - \theta)^2 = M(\theta - \boldsymbol{E}_{\mathcal{D}}[X]) + c$, where c is a constant that does not depend on θ.

Exercise 17.9

We now consider making predictions with the posterior of exercise 17.8. Suppose we now want to compute the probability

$$P(x[M+1] \mid \mathcal{D}) = \int P(x[M+1] \mid \theta) P(\theta \mid \mathcal{D}) d\theta.$$

Show that this distribution is Gaussian. What is the mean and precision of this distribution?

Exercise 17.10⋆

We now consider the complementary case to exercise 17.8, where we know the mean of X, but do not know the precision. Suppose that $X \sim \mathcal{N}\left(\mu; \theta^{-1}\right)$, where θ is the unknown precision.

We start with a definition. We say that $Y \sim \text{Gamma}(\alpha; \beta)$ (for $\alpha, \beta > 0$) if

$$P(y : \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^\alpha e^{-\beta y}.$$

Gamma distribution

This distribution is called a *Gamma distribution*. Here, we have that $\mathbf{E}[Y] = \frac{\alpha}{\beta}$ and $\mathbf{Var}[Y] = \frac{\alpha}{\beta^2}$.

a. Show that Gamma distributions are a conjugate family for this learning task. More precisely, show that if $P(\theta) \sim \text{Gamma}(\alpha; \beta)$, then $P(\theta \mid \mathcal{D}) \sim \text{Gamma}(\alpha'; \beta')$ where

$$\begin{aligned} \alpha' &= \alpha + \frac{1}{2} M \\ \beta' &= \beta + \frac{1}{2} \sum_m (x[m] - \mu)^2. \end{aligned}$$

Hint: do not worry about the normalization constant, instead focus on the terms in the posterior that involve θ.

b. Derive the mean and variance of θ in the posterior. What can we say about beliefs given the data? How do they differ from the MLE estimate?

Exercise 17.11⋆⋆

Now consider the case where we know neither the mean nor the precision of X. We examine a family of distributions that are conjugate in this case.

normal-Gamma distribution

A *normal-Gamma distribution* over μ, τ is of the form:

$$P(\tau)\mathcal{P}(\mu \mid \tau)$$

where $P(\tau)$ is $\text{Gamma}(\alpha; \beta)$ and $P(\mu \mid \tau)$ is $\mathcal{N}(\mu_0; \lambda\tau)$. That is, the distribution over precisions is a Gamma distribution (as in exercise 17.10), and the distribution over the mean is a Gaussian (as in exercise 17.8), except that the precision of this distribution depends on τ.

Show that if $P(\mu, \tau)$ is normal-Gamma with parameters $\alpha, \beta, \mu_0, \lambda$, then the posterior $P(\mu, \tau \mid \mathcal{D})$ is also a Normal-Gamma distribution.

Exercise 17.12

Suppose that a prior on a parameter vector is $p(\boldsymbol{\theta}) \sim Dirichlet(\alpha_1, \ldots, \alpha_k)$. What is the MAP value of the parameters, that is, $\arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta})$?

Exercise 17.13⋆

In this exercise, we will define a general-purpose prior for models with shared parameters, along the lines of the BDe prior of section 17.4.3.

a. Following the lines of the derivation of the BDe prior, construct a parameter prior for a network with shared parameters, using the uniform distribution P' as the basis.
b. Now, extend your analysis to construct a BDe-like parameter prior for a plate model, using, as the basis, a sample size of $\alpha(\mathsf{Q})$ for each Q and a uniform distribution.

Exercise 17.14

Perform the analysis of section 17.5.1.1 in the case where the network is a Gaussian Bayesian network. Derive the form of the likelihood function in terms of the appropriate aggregate sufficient statistics, and show how the MLE is computed from these statistics.

Exercise 17.15

In section 17.5, we discussed sharing at both the global and the local level. We now consider an example where we have both. Consider the following elaboration of our University domain: Each course has an additional attribute *Level*, whose values are *undergrad* (l^0) or *grad* (l^1). The grading curve (distribution for *Grade*) now depends on *Level*: The curve is the same for all undergraduate courses, and depends on *Difficulty* and *Intelligence*, as before. For graduate courses, the distribution is different for each course and, moreover, does not depend on the student's intelligence.

Specify the set of multinomial parameters in this model, and the partition of the multinomial distributions that correctly captures the structure of the parameter sharing.

Exercise 17.16

a. Using the techniques and notation of section 17.5.1.2, describe the likelihood function for the DBN model of figure 6.2. Your answer should define the set of shared parameters, the partition of the variables in the ground network, the aggregate sufficient statistics, and the MLE.
b. Now, assume that we want to use Bayesian inference for the parameter estimation in this case. Assuming local parameter independence and a Dirichlet prior, write down the form of the prior and the posterior. How would you use your learned model to compute the probability of a new trajectory?

Exercise 17.17

Consider the application of the global sharing techniques of section 17.5.1.2 to the task of parameter estimation in a PRM. A key difference between PRMs and plate models is that the different instantiations of the same attribute in the ground network may not have the same in-degree. For instance, returning to example 6.16, let *Job*(S) be an attribute such that S is a logical variable ranging over Student. Assume that *Job*(S) depends on the average grade of all courses that the student has taken (where we round the grade-point-average to produce a discrete set of values). Show how we can parameterize such a model, and how we can aggregate statistics to learn its parameters.

Exercise 17.18

Suppose we have a single multinomial variable X with K values and we have a prior on the parameters governing X so that $\boldsymbol{\theta} \sim \mathit{Dirichlet}(\alpha_1, \ldots, \alpha_K)$. Assume we have some data set $\mathcal{D} = \{x[1], \ldots, x[M]\}$.

a. Show how to compute

$$P(X[M+1] = x^i, X[M+2] = x^j \mid \mathcal{D}).$$

(Hint: use the chain rule for probabilities.)

b. Suppose we decide to use the approximation

$$P(X[M+1] = x^i, X[M+2] = x^j \mid \mathcal{D}) \approx P(X[M+1] = x^i, \mid \mathcal{D}) P(X[M+2] = x^j \mid \mathcal{D}).$$

That is, we ignore the dependencies between $X[M+1]$ and $X[M+2]$. Analyze the error in this approximation (the ratio between the approximation and the correct probability). What is the quality of this approximation for small M? What is the asymptotic behavior of the approximation when $M \to \infty$. (Hint: deal separately with the case where $i = j$ and the case where $i \neq j$.)

Exercise 17.19★

We now prove a variant on proposition 17.7. Show that in the setting of that proposition 17.7

$$P(\boldsymbol{D}(P\|\hat{P}) > \epsilon) \leq Ke^{-2M\epsilon^2 \frac{1}{K^2} \frac{2}{(1+\frac{\epsilon}{K\chi})^2}}$$

where $K = |Val(X)|$.

a. Show that

$$P(\boldsymbol{D}(P\|\hat{P}) > \epsilon) \leq \sum_x P(\log \frac{P^*(x)}{\hat{P}(x)} > \frac{\epsilon}{KP(x)}).$$

b. Use this result and lemma 17.1 to prove the stated bound.

c. Show that the stated bound is tighter than the original bound of proposition 17.7. (Hint: examine the case when $\lambda = \frac{1}{K}$.)

Exercise 17.20

Prove theorem 17.3. Specifically, first prove that $P_{\boldsymbol{\theta}^{\text{opt}}} = \prod_i P^*(X_i \mid \text{Pa}^{\mathcal{G}}_{X_i})$ and then use theorem 8.5.

Exercise 17.21

Prove lemma 17.2. Hint: Show that $\boldsymbol{D}(P(X,Y)\|Q(X,Y)) = \boldsymbol{D}(P(X \mid Y)\|Q(X \mid Y)) + \boldsymbol{D}(P(X)\|Q(X))$, and then show that the inequality follows.

Exercise 17.22

Suppose P is a Bayesian network with $P(x_i \mid \text{pa}_i) \geq \lambda$ for all i, x_i and pa_i. Consider a family X_i, Pa_i, show that

$$P(x_i, \text{pa}_i) \geq \lambda^{|\text{Pa}_i|+1}.$$

Exercise 17.23★

We now prove a bound on the error when using Bayesian estimates. Let $\mathcal{D} = \{X[1], \ldots, X[M]\}$ consist of M IID samples of a discrete variable X. Let α and P_0 be the equivalent sample size and prior distribution for a Dirichlet prior. The Bayesian estimator will return the distribution

$$\tilde{P}(x) = \frac{M}{M+\alpha}\hat{P}(x) + \frac{\alpha}{M+\alpha}P_0(x).$$

We now want to analyze the error of such an estimate.

a. Prove the analogue of lemma 17.1. Show that

$$P(\log \frac{P^*(x)}{\tilde{P}(x)} > \epsilon) \leq e^{-\frac{2M(\frac{M}{M+\alpha}P^*(x)+\frac{\alpha}{M+\alpha}P_0(x))^2\epsilon_x^2}{(1+\epsilon_x)^2}}.$$

b. Use the union bound to show that if $P^*(x) \geq \lambda$ and $P_0(x) \geq \lambda_0$ for all $x \in Val(X)$, then

$$P(\boldsymbol{D}(P^*(X)\|\tilde{P}(X)) > \epsilon) \leq |Val(X)|e^{-2M(\frac{M}{M+\alpha}\lambda+\frac{\alpha}{M+\alpha}\lambda_0)^2\epsilon^2\frac{1}{(1+\epsilon)^2}}.$$

c. Show that

$$\frac{M}{M+\alpha}\lambda + \frac{\alpha}{M+\alpha}\lambda_0 \geq \max(\lambda, \frac{\alpha}{M+\alpha}\lambda_0).$$

d. Suppose that $\lambda_0 > \lambda$. That is, our prior is less extreme than the real distribution, which is definitely the case if we take P_0 to be the uniform distribution. What can you conclude about a PAC result for the Bayesian estimate?

18 Structure Learning in Bayesian Networks

18.1 Introduction

18.1.1 Problem Definition

In the previous chapter, we examined how to learn the parameters of Bayesian networks. We made a strong assumption that we know in advance the network structure, or at least we decide on one regardless of whether it is correct or not. In this chapter, we consider the task of learning in situations where do not know the structure of the Bayesian network in advance. Throughout this chapter, we continue with the (very) strong assumption that our data set is fully observed, deferring the discussion of learning with partially observed data to the next chapter.

As in our discussion so far, we assume that the data $\mathcal{D}$ are generated IID from an underlying distribution $P^*(\mathcal{X})$. Here, we also assume that P^* is induced by some Bayesian network $\mathcal{G}^*$ over $\mathcal{X}$. We begin by considering the extent to which independencies in $\mathcal{G}^*$ manifest in $\mathcal{D}$.

Example 18.1

Consider an experiment where we toss two standard coins X and Y independently. We are given a data set with 100 instances of this experiment. We would like to learn a model for this scenario. A "typical" data set may have 27 head/head, 22 head/tail, 25 tail/head, and 26 tail/tail entries. In the empirical distribution, the two coins are not independent. This may seem reasonable, since the probability of tossing 100 pairs of fair coins and getting exactly *25 outcomes in each category is quite small (approximately $1/1,000$). Thus, even if the two coins are independent, we do not expect the observed empirical distribution to satisfy independence.*

Now suppose we get the same results in a very different situation. Say we scan the sports section of our local newspaper for 100 days and choose an article at random each day. We mark $X = x^1$ if the word "rain" appears in the article and $X = x^0$ otherwise. Similarly, Y denotes whether the word "football" appears in the article. Here our intuitions as to whether the two random variables are independent are unclear. If we get the same empirical counts as in the coins described before, we might suspect that there is some weak connection. In other words, it is hard to be sure whether the true underlying model has an edge between X and Y or not. ■

knowledge discovery

independence test

The importance of correctly reconstructing the network structure depends on our learning goal. As we discussed in chapter 16, there are different reasons for learning the model structure. One is for *knowledge discovery*: by examining the dependencies in the learned network, we can learn the dependency structure relating variables in our domain. Of course, there are other methods that reveal correlations between variables, for example, simple statistical *independence*

tests. A Bayesian network structure, however, reveals much finer structure. For instance, it can potentially distinguish between direct and indirect dependencies, both of which lead to correlations in the resulting distribution.

If our goal is to understand the domain structure, then, clearly, the best answer we can aspire to is recovering $\mathcal{G}^*$. Even here, must be careful. Recall that there can be many perfect maps for a distribution P^*: all of the networks in the same I-equivalence class as $\mathcal{G}^*$. All of these are equally good structures for P^*, and therefore we cannot distinguish between them based only on the data D. In other words, $\mathcal{G}^*$ **is not *identifiable* from the data. Thus, the best we can hope for is an algorithm that, asymptotically, recovers $\mathcal{G}^*$'s equivalence class.**

I-equivalence

Unfortunately, as our example indicates, the goal of learning $\mathcal{G}^*$ (or an equivalent network) is hard to achieve. The data sampled from P^* are noisy and do not reconstruct this distribution perfectly. We cannot detect with complete reliability which independencies are present in the underlying distribution. Therefore, **we must generally make a decision about our willingness to include in our learned model edges about which we are less sure. If we include more of these edges, we will often learn a model that contains spurious edges. If we include fewer edges, we may miss dependencies. Both compromises lead to inaccurate structures that do not reveal the correct underlying structure. The decision of whether it is better to have spurious correlations or spurious independencies depends on the application.**

identifiability

The second and more common reason to learn a network structure is in an attempt to perform *density estimation* — that is, to estimate a statistical model of the underlying distribution. As we discussed, our goal is to use this model for reasoning about instances that were not in our training data. In other words, we want our network model to *generalize* to new instances. It seems intuitively reasonable that because $\mathcal{G}^*$ captures the true dependencies and independencies in the domain, the best generalization will be obtained if we recover the the structure $\mathcal{G}^*$. Moreover, it seems that if we do make mistakes in the structure, it is better to have too many rather than too few edges. With an overly complex structure, we can still capture P^*, and thereby represent the true distribution.

density estimation

generalization

Unfortunately, the situation is somewhat more complex. Let us go back to our coin example and assume that we had 20 data cases with the following frequencies: 3 head/head, 6 head/tail, 5 tail/head, and 6 tail/tail. We can introduce a spurious correlation between X and Y, which would give us, using maximum likelihood estimation, the parameters $P(X = H) = 0.45$, $P(Y = H \mid X = H) = 1/3$, and $P(Y = H \mid X = T) = 5/11$. On the other hand, in the independent structure (with no edge between X and Y), the parameter of Y would be $P(Y = H) = 0.4$. All of these parameter estimates are imperfect, of course, but the ones in the more complex model are significantly more likely to be skewed, because each is estimated from a much smaller data set. In particular, $P(Y = H \mid X = H)$ is estimated from a data set of 9 instances, as opposed to 20 for the estimation of $P(Y = H)$. Recall that the standard deviation of the maximum likelihood estimate behaves as $1/\sqrt{M}$. Thus, if the coins are fair, the standard deviation of the MLE estimate from 20 samples is approximately 0.11, while the standard deviation from 9 samples is approximately 0.17. This example is simply an instance of the *data fragmentation* issue that we discussed in section 17.2.3 in the previous chapter. As we discussed, when we add more parents to the variable Y, the data used to estimate the CPD fragment into more bins, leaving fewer instances in each bin to estimate the parameters and reducing the quality of the estimated parameters. In a table-CPD, the number of bins grows exponentially with the number of parents, so the (statistical) cost of adding a parent can be very

data fragmentation

large; moreover, because of the exponential growth, the incremental cost of adding a parent grows with the number of parents already there.

Thus, when doing density estimation from limited data, it is often better to prefer a sparser structure. The surprising fact is that this observation applies not only to networks that include spurious edges relative to $\mathcal{G}^*$, but also to edges in $\mathcal{G}^*$. That is, we can sometimes learn a better model in term of generalization by learning a structure with fewer edges, even if this structure is incapable of representing the true underlying distribution.

18.1.2 Overview of Methods

Roughly speaking, there are three approaches to learning without a prespecified structure.

constraint-based structure learning

One approach utilizes *constraint-based structure learning*. These approaches view a Bayesian network as a representation of independencies. They try to test for conditional dependence and independence in the data and then to find a network (or more precisely an equivalence class of networks) that best explains these dependencies and independencies. Constraint-based methods are quite intuitive: they decouple the problem of finding structure from the notion of independence, and they follow more closely the definition of Bayesian network: we have a distribution that satisfies a set of independencies, and our goal is to find an I-map for this distribution. Unfortunately, these methods can be sensitive to failures in individual independence tests. It suffices that one of these tests return a wrong answer to mislead the network construction procedure.

score-based structure learning

model selection

hypothesis space

The second approach is *score-based structure learning*. Score-based methods view a Bayesian network as specifying a statistical model and then address learning as a *model selection* problem. These all operate on the same principle: We define a *hypothesis space* of potential models — the set of possible network structures we are willing to consider — and a scoring function that measures how well the model fits the observed data. Our computational task is then to find the highest-scoring network structure. The space of Bayesian networks is a combinatorial space, consisting of a superexponential number of structures — $2^{O(n^2)}$. Therefore, even with a scoring function, it is not clear how one can find the highest-scoring network. As we will see, there are very special cases where we can find the optimal network. In general, however, the problem is (as usual) $\mathcal{NP}$-hard, and we resort to heuristic search techniques. Score-based methods consider the whole structure at once; they are therefore less sensitive to individual failures and better at making compromises between the extent to which variables are dependent in the data and the "cost" of adding the edge. The disadvantage of the score-based approaches is that they pose a search problem that may not have an elegant and efficient solution.

Bayesian model averaging

Finally, the third approach does not attempt to learn a single structure; instead, it generates an ensemble of possible structures. These *Bayesian model averaging* methods extend the Bayesian reasoning we encountered in the previous chapter and try to average the prediction of all possible structures. Since the number of structures is immense, performing this task seems impossible. For some classes of models this can be done efficiently, and for others we need to resort to approximations.

18.2 Constraint-Based Approaches

18.2.1 General Framework

In constraint-based approaches, we attempt to reconstruct a network structure that best captures the independencies in the domain. In other words, we attempt to find the best minimal I-map for the domain.

Recall that in chapter 3 we discussed algorithms for building I-maps and P-maps that assume that we can test for independence statements in the distribution. The algorithms for constraint-based learning are essentially variants of these algorithms. The main technical question is how to answer independence queries. For now, assume that we have some procedure that can answer such queries. That is, for a given distribution P, the learning algorithm can pose a question, such as "Does P satisfy $(X_1 \perp X_2, X_3 \mid X_4)$?" and receive a yes/no answer. The task of the algorithm is to carry out some algorithm that interacts with this procedure and results in a network structure that is the minimal I-map of P.

minimal I-map

We have already seen such an algorithm in chapter 3: Build-Minimal-I-Map constructs a *minimal I-map* given a fixed ordering. For each variable X_i, it then searches for the minimal subset of $X_1, \ldots, X_{i-1}$ that render X_i independent of the others. This algorithm was useful in illustrating the definition of an I-map, but it suffers from several drawbacks in the context of learning. First, the input order over variables can have a serious impact on the complexity of the network we find. Second, in learning the parents of X_i, this algorithm poses independence queries of the form $(X_i \perp \{X_1, \ldots, X_{i-1}\} - \boldsymbol{U} \mid \boldsymbol{U})$. These conditional independence statements involve a large number of variables. Although we do not assume much about the independence testing procedure, we do realize that independence statements with many variables are much more problematic to resolve from empirical data. Finally, Build-Minimal-I-Map performs a large number of queries. For determining the parents of X_i, it must, in principle, examine all the 2^{i-1} possible subsets of $X_1, \ldots, X_{i-1}$.

class PDAG

To avoid these problems, we learn an I-equivalence class rather than a single network, and we use a *class PDAG* to represent this class. The algorithm that we use is a variant of the Build-PDAG procedure of algorithm 3.5. As we discuss, this algorithm reconstructs the network that best matches the domain without a prespecified order and uses only a polynomial number of *independence tests* that involve a bounded number of variables.

independence test

To achieve these performance guarantees, we must make some assumptions:

- The network $\mathcal{G}^*$ has bounded indegree, that is, for all i, $|\mathrm{Pa}_{X_i}^{\mathcal{G}^*}| \leq d$ for some constant d.
- The independence procedure can perfectly answer any independence query that involves up to $2d + 2$ variables.
- The underlying distribution P^* is faithful to $\mathcal{G}^*$, as in definition 3.8.

The first assumption states the boundaries of when we expect the algorithm to work. If the network is simple in this sense, the algorithm will be able to learn it from the data. If the network is more complex, then we cannot hope to learn it with "small" independence queries that involve only a few variables.

The second assumption is stronger, since it requires that the oracle can deal with queries up to a certain size. The learning algorithm does not depend on how the these queries are answered. They might be answered by performing a statistical test for conditional dependence

on a training data, or by an active mechanism that gathers more samples until it can reach a significant conclusion about this relations. We discuss how to construct such an oracle in more detail later in this chapter. Note that the oracle can also be a human expert who helps in constructing a model of the network.

The third assumption is the strongest. It is required to ensure that the algorithm is not misled by spurious independencies that are not an artifact of the oracle but rather exist in the domain. By requiring that $\mathcal{G}^*$ is a perfect map of P^*, we rule out quite a few situations, for example, the (noisy) XOR example of example 3.6, and various cases where additional independencies arise from structure in the CPDs.

Once we make these assumptions, the setting is precisely the one we tackled in section 3.4.3. Thus, given an oracle that can answer independence statements perfectly, we can now simply apply Build-PMap-Skeleton. Of course, determining independencies from the data is not a trivial problem, and the answers are rarely guaranteed to be perfect in practice. We will return to these important questions. For the moment, we focus on analyzing the number of independence queries that we need to answer, and thereby the complexity of the algorithm.

Recall that, in the construction of perfect maps, we perform independence queries only in the Build-PMAP-Skeleton procedure, when we search for a witness to the separation between every pair of variables. These witnesses are also used within Mark-Immoralities to determine whether the two parents in a v-structure are conditionally independent. According to lemma 3.2, if X and Y are not adjacent in $\mathcal{G}^*$, then either $\mathrm{Pa}_X^{\mathcal{G}^*}$ or $\mathrm{Pa}_Y^{\mathcal{G}^*}$ is a witness set. If we assume that $\mathcal{G}^*$ has indegree of at most d, we can therefore limit our attention to witness sets of size at most d. Thus, the number of independence queries in this step is polynomial in n, the number of variables. Of course, this number is exponential in d, but we assume that d is a fixed constant throughout the analysis.

Thus, given our assumptions, we can perform a variant of Build-PDAG that performs a polynomial number of independence tests. We can also check all other operations; that is, applying the edge orientation rules, we can also require a polynomial number of steps. Thus, the procedure is polynomial in the number of variables.

18.2.2 Independence Tests

The only remaining question is how to answer queries about conditional independencies between variables in the data. As one might expect, this question has been extensively studied in the statistics literature. We briefly touch on some of the issues and outline one commonly-used methodology to answer this question.

The basic query of this type is to determine whether two variables are independent. As in the example in the introduction to this chapter, we are given joint samples of two variables X and Y, and we want to determine whether X and Y are independent. This basic question is often referred to as *hypothesis testing*.

hypothesis testing

18.2.2.1 Single-Sided Hypothesis Tests

In hypothesis testing, we have a base hypothesis that is usually denoted by H_0 and is referred to as the *null hypothesis*. In the particular case of the independence test, the null hypothesis is "the data were sampled from a distribution $P^*(X, Y) = P^*(X)P^*(Y)$." Note that this

null hypothesis

assumption states that the data were sampled from a *particular* distribution in which X and Y are independent. In real life, we do not have access to $P^*(X)$ and $P^*(Y)$. As a substitute, we use $\hat{P}(X)$ and $\hat{P}(Y)$ as our best approximation for this distribution. Thus, we usually form H_0 as the assumption that $P^*(X, Y) = \hat{P}(X)\hat{P}(Y)$.

decision rule

We want to test whether the data conform to this hypothesis. More precisely, we want to find a procedure that we will call a *decision rule* that will take as input a data set $\mathcal{D}$, and return a verdict, either Accept or Reject. We will denote the function the procedure computes to be $R(\mathcal{D})$. If $R(\mathcal{D}) =$ Accept, then we consider that the data satisfy the hypothesis. In our case, that would mean that we believe that the data were sampled from P^* and that the two variables are independent. Otherwise, we decide to reject the hypothesis, which in our case would imply that the variables are dependent.

The question is then, of course, how to choose a "good" decision rule. A liberal decision rule that accepts many data sets runs the risk of accepting ones that do not satisfy the hypothesis. A conservative rule that rejects many data sets runs the risk of rejecting many that satisfy the hypothesis. The common approach to evaluating a decision rule is analyze the probability of false rejection. Suppose we have access to the distribution $P(\mathcal{D} : H_0, M)$ of data sets of M instances given the null hypothesis. That is, we can evaluate the probability of seeing each particular data set if the hypothesis happens to be correct. In our case, since the hypothesis specifies the distribution P^*, this distribution is just the probability of sampling the particular instances in the data set (we assume that the size of the data set is known in advance).

If we have access to this distribution, we can compute the probability of false rejection:

$$P(\{\mathcal{D} : R(\mathcal{D}) = \text{Reject}\} \mid H_0, M).$$

Then we can say that a decision rule R has a probability of false rejection p. We often refer to $1 - p$ as the confidence in the decision to reject an hypothesis.[1]

At this point we cannot evaluate the probability of false acceptances. Since we are not willing to assume a concrete distribution on data sets that violate H_0, we cannot quantify this probability. For this reason, the decision is not symmetric. That is, rejecting the hypothesis "X and Y are independent" is not the same as accepting the hypothesis "X and Y are dependent." In particular, to define the latter hypothesis we need to specify a distribution over data sets.

18.2.2.2 Deviance Measures

deviance

The preceding discussion suggests how to evaluate decision rules. Yet, it leaves open the question of how to design such a rule. A standard framework for this question is to define a measure of *deviance* from the null hypothesis. Such a measure d is a function from possible data sets to the real line. Intuitively, large value of $d(\mathcal{D})$ implies that $\mathcal{D}$ is far away from the null hypothesis.

χ^2 statistic

To consider a concrete example, suppose we have discrete-valued, independent random variables X and Y. Typically, we expect that the counts $M[x, y]$ in the data are close to $M \cdot \hat{P}(x) \cdot \hat{P}(y)$ (where M is the number of samples). This is the expected value of the count, and, as we know, deviances from this value are improbable for large M. Based on this intuition, we can measure the deviance of the data from H_0 in terms of these distances. A common measure of this type is the χ^2 *statistic*:

1. This leads statisticians to state, "We reject the null hypothesis with confidence 95 percent" as a precise statement that can be intuitively interpreted as, "We are quite confident that the variables are correlated."

$$d_{\chi^2}(\mathcal{D}) = \sum_{x,y} \frac{(M[x,y] - M \cdot \hat{P}(x) \cdot \hat{P}(y))^2}{M \cdot \hat{P}(x) \cdot \hat{P}(y)}. \tag{18.1}$$

A data set that perfectly fits the independence assumption has $d_{\chi^2}(\mathcal{D}) = 0$, and a data set where the empirical and expected counts diverge significantly has a larger value.

mutual information

Another potential deviance measure for the same hypothesis is the *mutual information* $\boldsymbol{I}_{\hat{P}_D}(X;Y)$ in the empirical distribution defined by the data set $\mathcal{D}$. In terms of counts, this can be written as

$$d_{\boldsymbol{I}}(\mathcal{D}) = \boldsymbol{I}_{\hat{P}_D}(X;Y) = \sum_{x,y} \frac{M[x,y]}{M} \log \frac{M[x,y]/M}{M[x]/M \cdot M[y]/M}. \tag{18.2}$$

In fact, these two deviance measures are closely related to each other; see exercise 18.1.

Once we agree on a deviance measure d (say the χ^2 statistic or the empirical mutual information), we can devise a rule for testing whether we want to accept the hypothesis

$$R_{d,t}(\mathcal{D}) = \begin{cases} \text{Accept} & d(\mathcal{D}) \leq t \\ \text{Reject} & d(\mathcal{D}) > t. \end{cases}$$

This rule *accepts* the hypothesis if the deviance is small (less than the predetermined threshold t) and *rejects* the hypothesis if the deviance is large.

The choice of threshold t determines the false rejection probability of the decision rule. The computational problem is to compute the false rejection probability for different values of t.

p-value

This value is called the *p-value* of t:

$$\text{p-value}(t) = P(\{\mathcal{D} : d(\mathcal{D}) > t\} \mid H_0, M).$$

18.2.2.3 Testing for Independence

Using the tools we developed so far, we can reexamine the independence test. The basic tool we use is a test to reject the null hypothesis that distribution of X and Y is the one we would estimate if we assume that they are independent. The typical significance level we use is 95 percent. That is, we reject the null hypothesis if the deviance in the observed data has p-value of 0.05 or less.

If we want to test the independence of discrete categorical variables, we usually use the χ^2 statistic or the mutual information. The null hypothesis is that $P^*(X,Y) = \hat{P}(X)\hat{P}(Y)$.

We start by considering how to perform an *exact test*. The definition of p-value requires summing over all possible data sets. In fact, since we care only about the sufficient statistics of X and Y in the data set, we can sum over the smaller space of different sufficient statistics vectors. Suppose we have M samples; we define the space $\mathcal{C}_{X,Y}^M$ to be the set of all empirical counts over X and Y, we might observe in a data set with M samples. Then we write

$$\text{p-value}(t) = \sum_{C[X,Y] \in \mathcal{C}_{X,Y}^M} \boldsymbol{I}\{d(C[X,Y]) > t\} P(C[X,Y] \mid H_0, M),$$

where $d(C[X,Y])$ is the deviance measure (that is, χ^2 or mutual information) computed with the counts $C[X,Y]$, and

$$P(C[X,Y] \mid H_0, M) = M! \prod_{x,y} \frac{1}{C[x,y]!} P(x,y \mid H_0)^{C[x,y]} \tag{18.3}$$

is the probability of seeing a data set with these counts given H_0; see exercise 18.2.

This exact approach enumerates through all data sets. This is clearly infeasible except for small values of M. A more common approach is to examine the asymptotic distribution of $M[x, y]$ under the null hypothesis. Since this count is a sum of binary indicator variables, its distribution is approximately normal when M is large enough. Statistical theory develops the asymptotic distribution of the deviance measure under the null hypothesis. For the χ^2 statistic, this distribution is called the χ^2 *distribution*. We can use the tail probability of this distribution to approximate p-values for independence tests. Numerical procedures for such computations are part of most standard statistical packages.

χ^2 distribution

A natural extension of this test exists for testing conditional independence. Suppose we want to test whether X and Y are independent given Z. Then, H_0 is that $P^*(X, Y, Z) = \hat{P}(Z)\hat{P}(X \mid Z)\hat{P}(Y \mid Z)$, and the χ^2 statistic is

$$d_{\chi^2}(\mathcal{D}) = \sum_{x,y,z} \frac{(M[x, y, z] - M \cdot \hat{P}(z)\hat{P}(x \mid z)\hat{P}(y \mid z))^2}{M \cdot \hat{P}(z)\hat{P}(x \mid z)\hat{P}(y \mid z)}.$$

This formula extends easily to conditioning on a set of variables $\boldsymbol{Z}$.

18.2.2.4 Building Networks

We now return to the problem of learning network structure. With the methods we just discussed, we can evaluate independence queries in the Build-PDAG procedure, so that whenever the test rejects the null hypothesis we treat the variables as dependent. One must realize, however, that these tests are not perfect. Thus, we run the risk of making wrong decisions on some of the queries. In particular, if we use significance level of 95 percent, then we expect that on average 1 in 20 rejections is wrong. When testing a large number of hypotheses, a scenario called *multiple hypothesis testing*, the number of incorrect conclusions can grow large, reducing our ability to reconstruct the correct network. We can try to reduce this number by taking stricter significance levels (see exercise 18.3). This, however, runs the risk of making more errors of the opposite type.

multiple hypothesis testing

In conclusion, we have to be aware that some of the independence tests results can be wrong. The procedure Build-PDAG can be sensitive to such errors. In particular, one misleading independence test result can produce multiple errors in the resulting PDAG (see exercise 18.4). When we have relatively few variables and large sample size (and "strong" dependencies among variables), the reconstruction algorithm we described here is efficient and often manages to find a structure that is quite close to the correct structure. When the independence test results are less pronounced, the constraint-based approach can run into trouble.

18.3 Structure Scores

As discussed earlier, score-based methods approach the problem of structure learning as an optimization problem. We define a score function that can score each candidate structure with respect to the training data, and then search for a high-scoring structure. As can be expected, one of the most important decisions we must make in this framework is the choice of scoring function. In this section, we discuss two of the most obvious choices.

18.3.1 Likelihood Scores

18.3.1.1 Maximum Likelihood Parameters

A natural choice for scoring function is the likelihood function, which we used for parameter estimation. Recall that this function measures the probability of the data given a model. Thus, it seems intuitive to find a model that would make the data as probable as possible.

Assume that we want to maximize the likelihood of the model. In this case, our model is a pair $\langle \mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \rangle$. Our goal is to find both a graph $\mathcal{G}$ and parameters $\boldsymbol{\theta}_{\mathcal{G}}$ that maximize the likelihood.

In the previous chapter, we determined how to maximize the likelihood for a given structure $\mathcal{G}$. We simply use the maximum likelihood parameters $\hat{\boldsymbol{\theta}}_{\mathcal{G}}$ for that graph. A simple analysis now shows that:

$$\begin{aligned} \max_{\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}} L(\langle \mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \rangle : \mathcal{D}) &= \max_{\mathcal{G}} [\max_{\boldsymbol{\theta}_{\mathcal{G}}} L(\langle \mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \rangle : \mathcal{D})] \\ &= \max_{\mathcal{G}} [L(\langle \mathcal{G}, \hat{\boldsymbol{\theta}}_{\mathcal{G}} \rangle : \mathcal{D})]. \end{aligned}$$

In other words, to find the maximum likelihood $(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ pair, we should find the graph structure $\mathcal{G}$ that achieves the highest likelihood *when we use the MLE parameters for* $\mathcal{G}$. We define:

$$\text{score}_L(\mathcal{G} \ : \ \mathcal{D}) = \ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}),$$

where $\ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D})$ is the logarithm of the likelihood function and $\hat{\boldsymbol{\theta}}_{\mathcal{G}}$ are the maximum likelihood parameters for $\mathcal{G}$. (As usual, it will be easier to deal with the logarithm of the likelihood.)

18.3.1.2 Information-Theoretic Interpretation

To get a better intuition of the likelihood score, let us consider the scenario of example 18.1. Consider the model $\mathcal{G}_0$ where X and Y are independent. In this case, we get

$$\text{score}_L(\mathcal{G}_0 \ : \ \mathcal{D}) = \sum_m \log \hat{\theta}_{x[m]} + \log \hat{\theta}_{y[m]}.$$

On the other hand, we can consider the model G_1 where there is an arc $X \rightarrow Y$. The log-likelihood for this model is

$$\text{score}_L(\mathcal{G}_1 \ : \ \mathcal{D}) = \sum_m \log \hat{\theta}_{x[m]} + \log \hat{\theta}_{y[m]|x[m]},$$

where $\hat{\theta}_x$ is again the maximum likelihood estimate for $P(x)$, and $\hat{\theta}_{y|x}$ is the maximum likelihood estimate for $P(y \mid x)$.

We see that the score of two models share a common component (the terms of the form $\log \hat{\theta}_x$). Thus, we can write the difference between the two scores as

$$\text{score}_L(\mathcal{G}_1 \ : \ \mathcal{D}) - \text{score}_L(\mathcal{G}_0 \ : \ \mathcal{D}) = \sum_m \log \hat{\theta}_{y[m]|x[m]} - \log \hat{\theta}_{y[m]}.$$

By counting how many times each conditional probability parameter appears in this term, we can write this sum as:

$$\text{score}_L(\mathcal{G}_1 \ : \ \mathcal{D}) - \text{score}_L(\mathcal{G}_0 \ : \ \mathcal{D}) = \sum_{x,y} M[x, y] \log \hat{\theta}_{y|x} - \sum_y M[y] \log \hat{\theta}_y.$$

Let $\hat{P}$ be the empirical distribution observed in the data; that is, $\hat{P}(x, y)$ is simply the empirical frequency of x, y in D. Then, we can write $M[x, y] = M \cdot \hat{P}(x, y)$, and $M[y] = M\hat{P}(y)$. Moreover, it is easy to check that $\hat{\theta}_{y|x} = \hat{P}(y \mid x)$, and that $\hat{\theta}_y = \hat{P}(y)$. We get:

$$\text{score}_L(\mathcal{G}_1 \ : \ \mathcal{D}) - \text{score}_L(\mathcal{G}_0 \ : \ \mathcal{D}) = M \sum_{x,y} \hat{P}(x, y) \log \frac{\hat{P}(y \mid x)}{\hat{P}(y)} = M \cdot \boldsymbol{I}_{\hat{P}}(X; Y),$$

mutual information

where $\boldsymbol{I}_{\hat{P}}(X; Y)$ is the *mutual information* between X and Y in the distribution $\hat{P}$.

We see that the likelihood of the model $\mathcal{G}_1$ depends on the mutual information between X and Y. Recall that higher mutual information implies stronger dependency. Thus, stronger dependency implies stronger preference for the model where X and Y depend on each other.

Can we generalize this information-theoretic formulation of the maximum likelihood score to general network structures? Going through a similar arithmetic transformations, we can prove the following result.

Proposition 18.1 *The likelihood score* decomposes *as follows:*

decomposable score

$$\text{score}_L(\mathcal{G} \ : \ \mathcal{D}) = M \sum_{i=1}^{n} \boldsymbol{I}_{\hat{P}}(X_i; \text{Pa}^{\mathcal{G}}_{X_i}) - M \sum_{i=1}^{n} \boldsymbol{H}_{\hat{P}}(X_i). \tag{18.4}$$

Proof We have already seen that by combining all the occurrences of each parameter $\theta_{x_i|\boldsymbol{u}}$, we can rewrite the log-likelihood function as

$$\ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) = \sum_{i=1}^{n} \left[\sum_{\boldsymbol{u}_i \in Val(\text{Pa}^{\mathcal{G}}_{X_i})} \sum_{x_i} M[x_i, \boldsymbol{u}_i] \log \hat{\theta}_{x_i|\boldsymbol{u}_i} \right].$$

Consider one of the terms in the square brackets, and let $\boldsymbol{U}_i = \text{Pa}_{X_i}$.

$$\begin{aligned}
&\frac{1}{M} \sum_{\boldsymbol{u}_i} \sum_{x_i} M[x_i, \boldsymbol{u}_i] \log \hat{\theta}_{x_i|\boldsymbol{u}_i} \\
&= \sum_{\boldsymbol{u}_i} \sum_{x_i} \hat{P}(x_i, \boldsymbol{u}_i) \log \hat{P}(x_i \mid \boldsymbol{u}_i) \\
&= \sum_{\boldsymbol{u}_i} \sum_{x_i} \hat{P}(x_i, \boldsymbol{u}_i) \log \left(\frac{\hat{P}(x_i, \boldsymbol{u}_i)}{\hat{P}(\boldsymbol{u}_i)} \frac{\hat{P}(x_i)}{\hat{P}(x_i)} \right) \\
&= \sum_{\boldsymbol{u}_i} \sum_{x_i} \hat{P}(x_i, \boldsymbol{u}_i) \log \frac{\hat{P}(x_i, \boldsymbol{u}_i)}{\hat{P}(\boldsymbol{u}_i)\hat{P}(x_i)} + \sum_{x_i} \left(\sum_{\boldsymbol{u}_i} \hat{P}(x_i, \boldsymbol{u}_i) \right) \log \hat{P}(x_i) \\
&= \boldsymbol{I}_{\hat{P}}(X_i; \boldsymbol{U}_i) - \sum_{x_i} \hat{P}(x_i) \log \frac{1}{\hat{P}(x_i)} \\
&= \boldsymbol{I}_{\hat{P}}(X_i; \boldsymbol{U}_i) - \boldsymbol{H}_{\hat{P}}(X_i),
\end{aligned}$$

where (as implied by the definition) the mutual information $\boldsymbol{I}_{\hat{P}}(X_i; \text{Pa}_{X_i})$ is 0 if $\text{Pa}_{X_i} = \emptyset$. ■

Note that the second sum in equation (18.4) does not depend on the network structure, and thus we can ignore it when we compare two structures with respect to the same data set.

Recall that we can interpret $\boldsymbol{I}_P(X;Y)$ as the strength of the dependence between X and Y in P. **Thus, the likelihood of a network measures the strength of the dependencies between variables and their parents. In other words, we prefer networks where the parents of each variable are informative about it.**

This result can also be interpreted in a complementary manner.

Corollary 18.1 *Let $X_1, \ldots, X_n$ be an ordering of the variables that is consistent with edges in $\mathcal{G}$. Then,*

$$\frac{1}{M}\text{score}_L(\mathcal{G} : \mathcal{D}) = \boldsymbol{H}_{\hat{P}}(X_1, \ldots, X_n) - \sum_{i=1}^{n} \boldsymbol{I}_{\hat{P}}(X_i; \{X_1, \ldots X_{i-1}\} - \text{Pa}_{X_i}^{\mathcal{G}} \mid \text{Pa}_{X_i}^{\mathcal{G}}). \quad (18.5)$$

For proof, see exercise 18.5.

Again, this second reformulation of the likelihood has a term that does not depend on the structure, and one that does. This latter term involves conditional mutual-information expressions of the form $\boldsymbol{I}_{\hat{P}}(X_i; \{X_1, \ldots X_{i-1}\} - \text{Pa}_{X_i}^{\mathcal{G}} \mid \text{Pa}_{X_i}^{\mathcal{G}})$. That is, the information between X_i and the preceding variables in the order given X_i's parents. Smaller conditional mutual-information terms imply higher scores. Recall that conditional independence is equivalent to having zero conditional mutual information. Thus, we can interpret this formulation as measuring to what extent the Markov properties implied by $\mathcal{G}$ are violated in the data. The smaller the violations of the Markov property, the larger the score.

These two interpretations are complementary, one measuring the strength of dependence between X_i and its parents $\text{Pa}_{X_i}^{\mathcal{G}}$, and the other measuring the extent of the independence of X_i from its predecessors given $\text{Pa}_{X_i}^{\mathcal{G}}$.

The process of choosing a network structure is often subject to constraints. Some constraints are a consequence of the acyclicity requirement, others may be due to a preference for simpler structures. Our previous analysis shows that the likelihood score provides valuable guidance in selecting between different candidate networks.

18.3.1.3 Limitations of the Maximum Likelihood Score

Based on the developments in the previous chapter and the preceding analysis, we see that the likelihood score is a good measure of the fit of the estimated Bayesian network and the training data. In learning structure, however, we are also concerned about the performance of the learned network on new instances sampled from the same underlying distribution P^*. Unfortunately, in this respect, the likelihood score can run into problems.

To see this, consider example 18.1. Let $\mathcal{G}_\emptyset$ be the network where X and Y are independent, and $\mathcal{G}_{X\to Y}$ the one where X is the parent of Y. As we have seen, $\text{score}_L(\mathcal{G}_{X\to Y} : \mathcal{D}) - \text{score}_L(\mathcal{G}_\emptyset : \mathcal{D}) = M \cdot \boldsymbol{I}_{\hat{P}}(X;Y)$. Recall that the mutual information between two variables is nonnegative. Thus, $\text{score}_L(\mathcal{G}_{X\to Y} : \mathcal{D}) \geq \text{score}_L(\mathcal{G}_\emptyset : \mathcal{D})$ for any data set D. This implies that the maximum likelihood score *never* prefers the simpler network over the more complex one. And it assigns both networks the same score only in these rare situations when X and Y are truly independent in the training data.

As explained in the introduction to this chapter, there are situations where we should prefer to

learn the simpler network (for example, when X and Y are nearly independent in the training data). We see that the maximum likelihood score would never lead us to make that choice.

This observation applies to more complex networks as well. It is easy to show that adding an edge to a network structure can never decrease the maximum likelihood score. Furthermore, the more complex network will have a higher score in all but a vanishingly small fraction of cases. One approach to proving this follows directly from the notion of likelihood; see exercise 18.6. Another uses the fact that, for any $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ and any distribution P, we have that:

$$\boldsymbol{I}_P(\boldsymbol{X}; \boldsymbol{Y} \cup \boldsymbol{Z}) \geq \boldsymbol{I}_P(\boldsymbol{X}; \boldsymbol{Y}),$$

with equality holding only if $\boldsymbol{Z}$ is conditionally independent of $\boldsymbol{X}$ given $\boldsymbol{Y}$, see exercise 2.20. This inequality is fairly intuitive: if $\boldsymbol{Y}$ gives us a certain amount of information about $\boldsymbol{X}$, adding $\boldsymbol{Z}$ can only give us more information. Thus, the mutual information between a variable and its parents can only go up if we add another parent, and it will go up except in those few cases where we get a conditional independence assertion holding exactly *in the empirical distribution*. It follows that **the maximum likelihood network will exhibit a conditional independence only when that independence happens to hold exactly in the empirical distribution. Due to statistical noise, exact independence almost never occurs, and therefore, in almost all cases, the maximum likelihood network will be a fully connected one. In other words, the likelihood score *overfits* the training data (see section 16.3.1), learning a model that precisely fits the specifics of the empirical distribution in our training set. This model therefore fails to generalize well to new data cases: these are sampled from the underlying distribution, which is not identical to the empirical distribution in our training set.**

overfitting

We note that the discussion of the maximum likelihood score was in the context of networks with table-CPDs. However, the same observations also apply to learning networks with other forms of CPDs (for example, tree-CPDs, noisy-ors, or Gaussians). In these cases, the information-theoretic analysis is somewhat more elaborate, but the general conclusions about the trade-offs between models and about overfitting apply.

Since the likelihood score does not provide us with tools to avoid overfitting, we have to be careful when using it. It is reasonable to use the maximum likelihood score when there are additional mechanisms that disallow overly complicated structures. For example, we will discuss learning networks with a fixed indegree. Such a limitation can constrain the tendency to overfit when using the maximum likelihood score.

18.3.2 Bayesian Score

We now examine an alternative scoring function that is based on a Bayesian perspective; this approach extends ideas that we described in the context of parameter estimation in the previous chapter. We will start by deriving the score from the Bayesian perspective, and then we will try to understand how it avoids overfitting.

Recall that the main principle of the Bayesian approach was that whenever we have uncertainty over anything, we should place a distribution over it. In this case, we have uncertainty both over structure and over parameters. We therefore define a structure prior $P(\mathcal{G})$ that puts a prior probability on different graph structures, and a parameter prior $P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G})$, that puts a

probability on different choice of parameters once the graph is given. By Bayes rule, we have

$$P(\mathcal{G} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathcal{G})P(\mathcal{G})}{P(\mathcal{D})},$$

where, as usual, the denominator is simply a normalizing factor that does not help distinguish between different structures. Thus, we define the *Bayesian score* as:

Bayesian score

$$\text{score}_B(\mathcal{G} : \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}). \tag{18.6}$$

The ability to ascribe a prior over structures gives us a way of preferring some structures over others. For example, we can penalize dense structures more than sparse ones. It turns out, however, that the structure-prior term in the score is almost irrelevant compared to the first term. This term, $P(\mathcal{D} \mid \mathcal{G})$, takes into consideration our uncertainty over the parameters:

$$P(\mathcal{D} \mid \mathcal{G}) = \int_{\Theta_\mathcal{G}} P(\mathcal{D} \mid \boldsymbol{\theta}_\mathcal{G}, \mathcal{G}) P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G}) d\boldsymbol{\theta}_\mathcal{G}, \tag{18.7}$$

where $P(\mathcal{D} \mid \boldsymbol{\theta}_\mathcal{G}, \mathcal{G})$ is the likelihood of the data given the network $\langle \mathcal{G}, \boldsymbol{\theta}_\mathcal{G} \rangle$ and $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G})$ is our prior distribution over different parameter values for the network $\mathcal{G}$. Recall from section 17.4 that $P(\mathcal{D} \mid \mathcal{G})$ is called the *marginal likelihood* of the data given the structure, since we marginalize out the unknown parameters.

marginal likelihood

It is important to realize that the marginal likelihood is quite different from the maximum likelihood score. Both terms examine the likelihood of the data given the structure. The maximum likelihood score returns the maximum of this function. In contrast, the marginal likelihood is the average value of this function, where we average based on the prior measure $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G})$. This difference will become apparent when we analyze the marginal likelihood term.

One explanation of why the Bayesian score avoids overfitting examines the sensitivity of the likelihood to the particular choice of parameters. As we discussed, the maximal likelihood is overly "optimistic" in its evaluation of the score: It evaluates the likelihood of the training data using the best parameter values for the given data. This estimate is realistic only if these parameters are also reflective of the data in general, a situation that never occurs.

The Bayesian approach tells us that, although the choice of parameter $\hat{\boldsymbol{\theta}}$ is the most likely given the training set D, it is not the only choice. The posterior over parameters provides us with a range of choices, along with a measure of how likely each of them is. By integrating $P(\mathcal{D} \mid \boldsymbol{\theta}_\mathcal{G}, \mathcal{G})$ over the different choices of parameters $\boldsymbol{\theta}_\mathcal{G}$, we are measuring the *expected* likelihood, averaged over different possible choices of $\boldsymbol{\theta}_\mathcal{G}$. Thus, we are being more conservative in our estimate of the "goodness" of the model.

holdout testing

Another motivation can be derived from the *holdout testing* methods discussed in box 16.A. Here, we consider different network structures, parameterized by the training set, and test their predictiveness (likelihood) on the validation set. When we find a network that best generalizes to the validation set (that is, has the best likelihood on this set), we have some reason to hope that it will also generalize to other unseen instances. As we discussed, the holdout method is sensitive to the particular split into training and test sets, both in terms of the relative sizes of the sets and in terms of which instances fall into which set. Moreover, it does not use all the available data in learning the structure, a potentially serious problem when we have limited amounts of data to learn from.

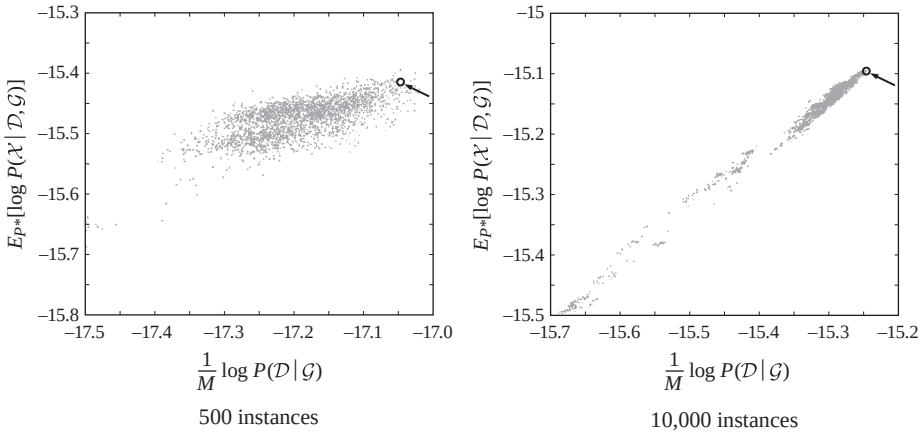


Figure 18.1 Marginal likelihood in training data as predictor of expected likelihood on underlying distribution. Comparison of the average log-marginal-likelihood per sample in training data (x-axis) to the expected log-likelihood of new samples from the underlying distribution (y-axis) for two data sets sampled from the ICU-Alarm network. Each point corresponds to a network structure; the true network structure is marked by a circle.

It turns out that the Bayesian approach can be viewed as performing a similar evaluation without explicitly splitting the data into two parts. Using the chain rule for probabilities, we can rewrite the marginal likelihood as

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_{m=1}^{M} P(\xi[m] \mid \xi[1], \ldots, \xi[m-1], \mathcal{G}).$$

Each of the terms in this product — $P(\xi[m] \mid \xi[1], \ldots, \xi[m-1], \mathcal{G})$ — is the probability of the m'th instance using the parameters learned from the first $m-1$ instances (using Bayesian estimation). We see that in this term we are using the m'th instance as a test case, since we are computing its probability using what we learned from previous instances. Thus, it provides us with one data point for testing the ability of our model to predict a new data instance, based on the model learned from the previous ones. This type of analysis is called a *prequential analysis*.

prequential analysis

However, unlike the holdout approach, we are not holding out any data. Each instance is evaluated in incremental order, and contributes both to our evaluation of the model and to our final model score. Moreover, the Bayesian score does not depend on the order of instances. Using the chain law of probabilities, we can generate a similar expansion for any ordering of the instances. Each one of these will give the same result (since these are different ways of expanding the term $P(\mathcal{D} \mid \mathcal{G})$).

This intuition suggests that

$$\frac{1}{M} \log P(\mathcal{D} \mid \mathcal{G}) \approx \boldsymbol{E}_{P^*}[\log P(\mathcal{X} \mid \mathcal{G}, \mathcal{D})] \tag{18.8}$$

is an estimator for the average log-likelihood of a new sample from the distribution P^*. In

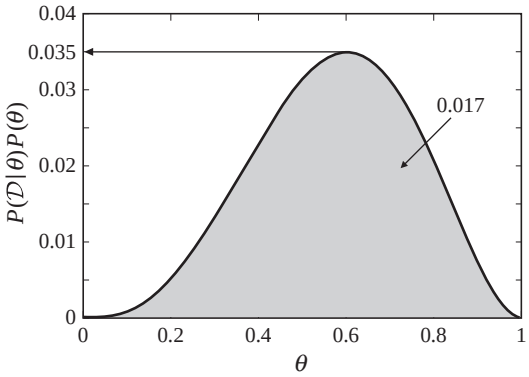


Figure 18.2 Maximal likelihood score versus marginal likelihood for the data $\langle H, T, T, H, H \rangle$.

practice, it turns out that for reasonable sample sizes this is indeed a fairly good estimator of the ability of a model to generalize to unseen data. Figure 18.1 demonstrates this property empirically for data sets sampled from the ICU-Alarm network. We generated a collection of network structures by sampling from the posterior distribution over structures given different data sets (see section 18.5). For each structure we evaluated the two sides of the preceding approximation: the average log-likelihood per sample, and the expected likelihood of new samples from the underlying distribution. As we can see, there is a general agreement between the estimate using the training data and the actual generalization error of each network structure. In particular, the difference in scores of two structures correlates with the differences in generalization error. This phenomenon is particularly noticeable in the larger training set.

We note that the Bayesian score is not the only way of providing "test set" performance using each instance. See exercise 18.12 for an alternative score with similar properties.

18.3.3 Marginal Likelihood for a Single Variable

We now examine how to compute the marginal likelihood for simple cases, and then in the next section treat the case of Bayesian networks.

Consider a single binary random variable X, and assume that we have a prior distribution $Dirichlet(\alpha_1, \alpha_0)$ over X. Consider a data set $\mathcal{D}$ that has $M[1]$ heads and $M[0]$ tails. Then, the maximum likelihood value given D is

$$P(\mathcal{D} \mid \hat{\theta}) = \left(\frac{M[1]}{M}\right)^{M[1]} \cdot \left(\frac{M[0]}{M}\right)^{M[0]}.$$

Now, consider the marginal likelihood. Here, we are not conditioning on the parameter. Instead, we need to compute the probability $P(X[1], \ldots, X[M])$ of the data given our prior. One approach to computing this term is to evaluate the integral equation (18.7). An alternative approach uses the chain rule

$$P(x[1], \ldots, x[M]) = P(x[1]) \cdot P(x[2] \mid x[1]) \cdot \ldots \cdot P(x[M] \mid x[1], \ldots, x[M-1]).$$

Recall that if we use a Beta prior, then

$$P(x[m+1] = H \mid x[1], \ldots, x[m]) = \frac{M^m[1] + \alpha_1}{m + \alpha},$$

where $M^m[1]$ is the number of heads in the first m examples. For example, if $\mathcal{D} = \langle H, T, T, H, H \rangle$,

$$\begin{aligned} P(x[1], \ldots, x[5]) &= \frac{\alpha_1}{\alpha} \cdot \frac{\alpha_0}{\alpha+1} \cdot \frac{\alpha_0+1}{\alpha+2} \cdot \frac{\alpha_1+1}{\alpha+3} \cdot \frac{\alpha_1+2}{\alpha+4} \\ &= \frac{[\alpha_1(\alpha_1+1)(\alpha_1+2)][\alpha_0(\alpha_0+1)]}{\alpha \cdots (\alpha+4)}. \end{aligned}$$

Picking $\alpha_1 = \alpha_0 = 1$, so that $\alpha = \alpha_1 + \alpha_0 = 2$, we get

$$\frac{[1 \cdot 2 \cdot 3] \cdot [1 \cdot 2]}{2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} = \frac{12}{720} = 0.017$$

(see figure 18.2), which is significantly lower than the likelihood

$$\left(\frac{3}{5}\right)^3 \cdot \left(\frac{2}{5}\right)^2 = \frac{108}{3125} \approx 0.035.$$

Thus, a model using maximum-likelihood parameters ascribes a much higher probability to this sequence than does the marginal likelihood. The reason is that the log-likelihood is making an overly optimistic assessment, based on a parameter that was designed with full retrospective knowledge to be an optimal fit to the entire sequence.

In general, for a binomial distribution with a Beta prior, we have

$$P(x[1], \ldots, x[M]) = \frac{[\alpha_1 \cdots (\alpha_1 + M[1] - 1)][\alpha_0 \cdots (\alpha_0 + M[0] - 1)]}{\alpha \cdots (\alpha + M - 1)}.$$

Each of the terms in square brackets is a product of a sequence of numbers such as $\alpha \cdot (\alpha + 1) \cdots (\alpha + M - 1)$. If α is an integer, we can write this product as $\frac{(\alpha+M-1)!}{(\alpha-1)!}$. However, we do not necessarily know that α is an integer. It turns out that we can use a generalization of the factorial function for this purpose. Recall that the *Gamma function* is such that $\Gamma(m) = (m-1)!$ and $\Gamma(x+1) = x \cdot \Gamma(x)$. Using the latter property, we can rewrite

Gamma function

$$\alpha(\alpha+1) \cdots (\alpha+M-1) = \frac{\Gamma(\alpha+M)}{\Gamma(\alpha)}.$$

Hence,

$$P(x[1], \ldots, x[M]) = \frac{\Gamma(\alpha)}{\Gamma(\alpha+M)} \cdot \frac{\Gamma(\alpha_1 + M[1])}{\Gamma(\alpha_1)} \cdot \frac{\Gamma(\alpha_0 + M[0])}{\Gamma(\alpha_0)}.$$

A similar formula holds for a multinomial distribution over the space $x^1, \ldots, x^k$, with a Dirichlet prior with hyperparameters $\alpha_1, \ldots, \alpha_k$:

$$P(x[1], \ldots, x[M]) = \frac{\Gamma(\alpha)}{\Gamma(\alpha+M)} \cdot \prod_{i=1}^{k} \frac{\Gamma(\alpha_i + M[x^i])}{\Gamma(\alpha_i)}. \tag{18.9}$$

Note that the final expression for the marginal likelihood is invariant to the order we selected in the expansion via the chain rule. In particular, any other order results in exactly the same final expression. This property is reassuring, because the IID assumption tells us that the specific order in which we get data cases is insignificant. Also note that the marginal likelihood can be computed directly from the same sufficient statistics used in the computation of the likelihood function — the counts of the different values of the variable in the data. This observation will continue to hold in the general case of Bayesian networks.

18.3.4 Bayesian Score for Bayesian Networks

We now generalize the discussion of the Bayesian score to more general Bayesian networks. Consider two possible structures over two binary random variables X and Y. $\mathcal{G}_\emptyset$ is the graph with no edges. Here, we have:

$$P(\mathcal{D} \mid \mathcal{G}_\emptyset) = \int_{\Theta_X \times \Theta_Y} P(\theta_X, \theta_Y \mid \mathcal{G}_\emptyset) P(\mathcal{D} \mid \theta_X, \theta_Y, \mathcal{G}_\emptyset) d[\theta_X, \theta_Y].$$

We know that the likelihood term $P(\mathcal{D} \mid \theta_X, \theta_Y, \mathcal{G}_\emptyset)$ can be written as a product of terms, one involving θ_X and the observations of X in the data, and the other involving θ_Y and the observations of Y in the data. If we also assume *parameter independence*, that is, that $P(\theta_X, \theta_Y \mid \mathcal{G}_\emptyset)$ decomposes as a product $P(\theta_X \mid \mathcal{G}_\emptyset) P(\theta_Y \mid \mathcal{G}_\emptyset)$, then we can simplify the integral

parameter independence

$$\begin{aligned} P(\mathcal{D} \mid \mathcal{G}_\emptyset) &= \left(\int_{\Theta_X} P(\theta_X \mid \mathcal{G}_\emptyset) \prod_m P(x[m] \mid \theta_X, \mathcal{G}_\emptyset) d\theta_X \right) \\ &\quad \left(\int_{\Theta_Y} P(\theta_Y \mid \mathcal{G}_\emptyset) \prod_m P(y[m] \mid \theta_Y, \mathcal{G}_\emptyset) d\theta_Y \right), \end{aligned}$$

where we used the fact that the integral of a product of independent functions is the product of integrals. Now notice that each of the two integrals is the marginal likelihood of a single variable. Thus, if X and Y are multinomials, and each has a Dirichlet prior, then we can write each integral using the closed form of equation (18.9).

Now consider the network $\mathcal{G}_{X \to Y} = (X \to Y)$. Once again, if we assume parameter independence, we can decompose this integral into a product of three integrals, each over a single parameter family.

$$\begin{aligned} P(\mathcal{D} \mid \mathcal{G}_{X \to Y}) &= \left(\int_{\Theta_X} P(\theta_X \mid \mathcal{G}_{X \to Y}) \prod_m P(x[m] \mid \theta_X, \mathcal{G}_{X \to Y}) d\theta_X \right) \\ &\quad \left(\int_{\Theta_{Y|x^0}} P(\theta_{Y|x^0} \mid \mathcal{G}_{X \to Y}) \prod_{m: x[m] = x^0} P(y[m] \mid \theta_{Y|x^0}, \mathcal{G}_{X \to Y}) d\theta_{Y|x^0} \right) \\ &\quad \left(\int_{\Theta_{Y|x^1}} P(\theta_{Y|x^1} \mid \mathcal{G}_{X \to Y}) \prod_{m: x[m] = x^1} P(y[m] \mid \theta_{Y|x^1}, \mathcal{G}_{X \to Y}) d\theta_{Y|x^1} \right). \end{aligned}$$

Again, each of these can be written using the closed form solution of equation (18.9).

Comparing the marginal likelihood of the two structures, we see that the term that corresponds to X is similar in both. In fact, the terms $P(x[m] \mid \theta_X, \mathcal{G}_\emptyset)$ and $P(x[m] \mid \theta_X, \mathcal{G}_{X\to Y})$ are identical (both make the same predictions given the parameter values). Thus, if we choose the prior $P(\Theta_X \mid \mathcal{G}_\emptyset)$ to be the same as $P(\Theta_X \mid \mathcal{G}_{X\to Y})$, we have that the first term in the marginal likelihood of both structures is identical.

Thus, given this assumption about the prior, the difference between the marginal likelihood of $\mathcal{G}_\emptyset$ and $\mathcal{G}_{X\to Y}$ is due to the difference between the marginal likelihood of all the observations of Y and the marginal likelihoods of the observations of Y when we partition our examples based on the observed value of X. Intuitively, if Y has a different distribution in these two cases, then the latter term will have better marginal likelihood. On the other hand, if Y is distributed in roughly the same manner in both subsets, then the simpler network will have better marginal likelihood.

To see this behavior, we consider an idealized experiment where the empirical distribution is such that $P(x^1) = 0.5$, and $P(y^1 \mid x^1) = 0.5 + p$ and $P(y^1 \mid x^0) = 0.5 - p$, where p is a free parameter. Larger values of p imply stronger dependence between X and Y. Note, however, that the marginal distributions of X and Y are the same regardless of the value of p. Thus, the score of the empty structure $\mathcal{G}_\emptyset$ does not depend on p. On the other hand, the score of the structure $\mathcal{G}_{X\to Y}$ depends on p. Figure 18.3 illustrates how these scores change as functions of the number of training samples. The graph compares the average score per instance (of equation (18.8)) for both structures for different values of p.

We can see that, as we get more data, the Bayesian score prefers the structure $\mathcal{G}_{X\to Y}$ where X and Y are dependent. When the dependency between them is strong, this preference arises very quickly. But as the dependency becomes weaker, more data are required in order to justify this selection. Thus, if the two variables are independent, small fluctuations in the data, due to sampling noise, are unlikely to cause a preference for the more complex structure. By contrast, any fluctuation from pure independence in the empirical distribution will cause the likelihood score to select the more complex structure.

We now return to consider the general case. As we can expect, the same arguments we applied to the two-variable networks apply to any network structure.

Proposition 18.2 *Let $\mathcal{G}$ be a network structure, and let $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G})$ be a parameter prior satisfying global parameter independence. Then,*

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_i \int_{\Theta_{X_i|\mathrm{Pa}_{X_i}}} \prod_m P(x_i[m] \mid \mathrm{pa}_{X_i}[m], \boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}, \mathcal{G}) P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} \mid \mathcal{G}) d\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}}.$$

Moreover, if $P(\boldsymbol{\theta}_\mathcal{G})$ also satisfies local parameter independence, then

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_i \prod_{\boldsymbol{u}_i \in Val(\mathrm{Pa}^\mathcal{G}_{X_i})} \int_{\Theta_{X_i|\boldsymbol{u}_i}} \prod_{m, \boldsymbol{u}_i[m]=\boldsymbol{u}_i} P(X_i[m] \mid \boldsymbol{u}_i, \boldsymbol{\theta}_{X_i|\boldsymbol{u}_i}, \mathcal{G}) P(\boldsymbol{\theta}_{X_i|\boldsymbol{u}_i} \mid \mathcal{G}) d\boldsymbol{\theta}_{X_i|\boldsymbol{u}_i}.$$

Using this proposition and the results about the marginal likelihood of Dirichlet priors, we conclude the following result: If we consider a network with Dirichlet priors where $P(\boldsymbol{\theta}_{X_i|\mathrm{pa}_{X_i}} \mid$

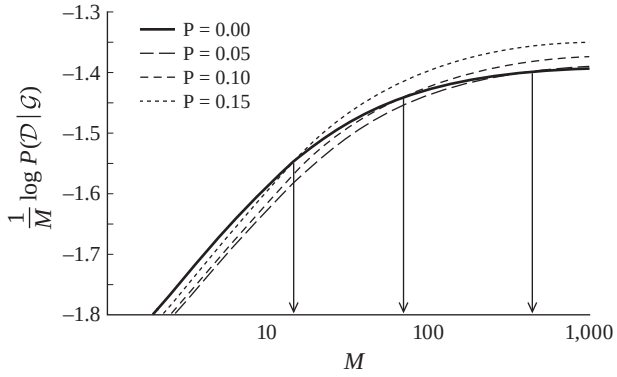


Figure 18.3 The effect of correlation on the Bayesian score. The solid line indicates the score of the independent model $\mathcal{G}_\emptyset$. The remaining lines indicate the score of the more complex structure $\mathcal{G}_{X\rightarrow Y}$, for different sampling distributions parameterized by p.

$\mathcal{G}$) has hyperparameters $\{\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i} : j = 1,\ldots,|X_i|\}$ then

$$P(\mathcal{D}\mid\mathcal{G}) = \prod_i \prod_{\boldsymbol{u}_i\in Val(\mathrm{Pa}^{\mathcal{G}}_{X_i})} \frac{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i})}{\Gamma(\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i}+M[\boldsymbol{u}_i])} \prod_{x_i^j\in Val(X_i)} \left[\frac{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i}+M[x_i^j,\boldsymbol{u}_i])}{\Gamma(\alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i})}\right],$$

where $\alpha^{\mathcal{G}}_{X_i|\boldsymbol{u}_i} = \sum_j \alpha^{\mathcal{G}}_{x_i^j|\boldsymbol{u}_i}$. In practice, we use the logarithm of this formula, which is more manageable to compute numerically.[2]

18.3.5 Understanding the Bayesian Score

As we have just seen, **the Bayesian score seems to be biased toward simpler structures, but as it gets more data, it is willing to recognize that a more complex structure is necessary. In other words, it appears to trade off fit to data with model complexity, thereby reducing the extent of *overfitting*.** To understand this behavior, it is useful to consider an approximation to the Bayesian score that better exposes its fundamental properties.

overfitting

Theorem 18.1

If we use a Dirichlet parameter prior for all parameters in our network, then, when $M\rightarrow\infty$, we have that:

$$\log P(\mathcal{D}\mid\mathcal{G}) = \ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}}:\mathcal{D}) - \frac{\log M}{2}\mathrm{Dim}[\mathcal{G}] + O(1),$$

model dimension

where $\mathrm{Dim}[\mathcal{G}]$ *is the* model dimension*, or the number of* independent parameters *in* $\mathcal{G}$.

independent parameters

See exercise 18.7 for the proof.

2. Most scientific computation libraries have efficient numerical implementation of the function $\log\Gamma(x)$, which enables us to compute this score efficiently.

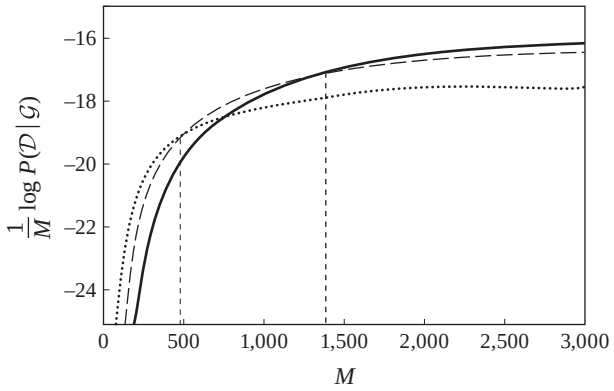


Figure 18.4 The Bayesian score of three structures, evaluated on synthetic data generated from the ICU-Alarm **network.** The solid line is the original structure, which has 509 parameters. The dashed line is a simplification that has 359 parameters. The dotted line is a tree-structure and has 214 parameters.

Thus, we see that the Bayesian score tends to trade off the likelihood — fit to data — on one hand and some notion of model complexity on the other hand. This approximation is called the *BIC score* (for Bayesian information criterion):

BIC score

$$\text{score}_{BIC}(\mathcal{G} \;:\; \mathcal{D}) = \ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) - \frac{\log M}{2}\text{Dim}[\mathcal{G}].$$

We note that the negation of this quantity can be viewed as the number of bits required to encode both the model ($\log M/2$ bits per model parameter, a derivation whose details we omit) and the data given the model (as per our discussion in section A.1.3). Thus, this objective is also known as *minimum description length*.

minimum description length

We can decompose this score even further using our analysis from equation (18.4):

$$\text{score}_{BIC}(\mathcal{G} \;:\; \mathcal{D}) = M\sum_{i=1}^{n} \boldsymbol{I}_{\hat{P}}(X_i;\text{Pa}_{X_i}) - M\sum_{i=1}^{n} \boldsymbol{H}_{\hat{P}}(X_i) - \frac{\log M}{2}\text{Dim}[\mathcal{G}].$$

We can observe several things about the behavior of this score function. First, the entropy terms do not depend on the graph, so they do not influence the choice of structure and can be ignored. The score exhibits a trade-off between fit to data and model complexity: the stronger the dependence of a variable on its parents, the higher the score; the more complex the network, the lower the score. However, the mutual information term grows linearly in M, whereas the complexity term grows logarithmically. Therefore, the larger M is, the more emphasis will be given to the fit to data.

Figure 18.4 illustrates this theorem empirically. It shows the Bayesian score of three structures on a data set generated by the ICU-Alarm network. One of these structures is the correct one, and the other two are simplifications of it. We can see that, for small M, the simpler structures have the highest scores. This is compatible with our analysis: for small data sets, the penalty term outweighs the likelihood term. But as M grows, the score begins to exhibit an increasing preference for the more complex structures. With enough data, the true model is preferred.

This last statement is a general observation about the BIC and Bayesian scores: Asymptotically, these scores will prefer a structure that exactly fits the dependencies in the data. To make this statement precise, we introduce the following definition:

Definition 18.1 consistent score

Assume that our data are generated by some distribution P^* for which the network $\mathcal{G}^*$ is a perfect map. We say that a scoring function is consistent *if the following properties hold as the amount of data $M \to \infty$, with probability that approaches 1 (over possible choices of data set D):*

- *The structure $\mathcal{G}^*$ will maximize the score.*
- *All structures $\mathcal{G}$ that are not I-equivalent to $\mathcal{G}^*$ will have strictly lower score.* ■

Theorem 18.2

The BIC score is consistent.

PROOF Our goal is to prove that for sufficiently large M, if the graph that maximizes the BIC score is $\mathcal{G}$, then $\mathcal{G}$ is I-equivalent to $\mathcal{G}^*$. We briefly sketch this proof.

Consider some graph $\mathcal{G}$ that implies an independence assumption that $\mathcal{G}^*$ does not support. Then $\mathcal{G}$ cannot be an I-map of the true underlying distribution P. Hence, $\mathcal{G}$ cannot be a maximum likelihood model with respect to the true distribution P^*, so that we must have:

$$\sum_i \boldsymbol{I}_{P^*}(X_i; \mathrm{Pa}_{X_i}^{\mathcal{G}^*}) > \sum_i \boldsymbol{I}_{P^*}(X_i; \mathrm{Pa}_{X_i}^{\mathcal{G}}).$$

As $M \to \infty$, our empirical distribution $\hat{P}$ will converge to P^* with probability 1. Therefore, for large M,

$$\mathrm{score}_L(\mathcal{G}^* : \mathcal{D}) - \mathrm{score}_L(\mathcal{G} : \mathcal{D}) \approx \Delta \cdot M,$$

where $\Delta = \sum_i \boldsymbol{I}_{P^*}(X_i; \mathrm{Pa}_{X_i}^{\mathcal{G}^*}) - \sum_i \boldsymbol{I}_{P^*}(X_i; \mathrm{Pa}_{X_i}^{\mathcal{G}})$. Therefore, asymptotically we have that

$$\mathrm{score}_{BIC}(\mathcal{G}^* : \mathcal{D}) - \mathrm{score}_{BIC}(\mathcal{G} : \mathcal{D}) \approx \Delta M + \frac{1}{2}(\mathrm{Dim}[\mathcal{G}] - \mathrm{Dim}[\mathcal{G}^*]) \log M.$$

The first term grows much faster than the second, so that eventually its effect will dominate, and the score of $\mathcal{G}^*$ will be better.

Now, assume that $\mathcal{G}$ implies all the independence assumptions in $\mathcal{G}^*$, but that $\mathcal{G}^*$ implies an independence assumption that $\mathcal{G}$ does not. (In other words, $\mathcal{G}$ is a superset of $\mathcal{G}^*$.) In this case, $\mathcal{G}$ can represent any distribution that $\mathcal{G}^*$ can. In particular, it can represent P^*. As $\hat{P}$ converges to P^*, we will have that:

$$\mathrm{score}_L(\mathcal{G}^* : \mathcal{D}) - \mathrm{score}_L(\mathcal{G} : \mathcal{D}) \to 0.$$

Therefore, asymptotically we have that for

$$\mathrm{score}_{BIC}(\mathcal{G}^* : \mathcal{D}) - \mathrm{score}_{BIC}(\mathcal{G} : \mathcal{D}) \approx \frac{1}{2}(\mathrm{Dim}[\mathcal{G}] - \mathrm{Dim}[\mathcal{G}^*]) \log M.$$

Now, since $\mathcal{G}$ makes fewer independence assumptions than $\mathcal{G}^*$, it must be parameterized by a larger set of parameters. Thus, $\mathrm{Dim}[\mathcal{G}] > \mathrm{Dim}[\mathcal{G}^*]$, so that $\mathcal{G}^*$ will be preferred to $\mathcal{G}$. ■

As the Bayesian score is asymptotically identical to BIC (the remaining $O(1)$ terms do not grow with M), we get:

Corollary 18.2 *The Bayesian score is consistent.*

Note that consistency is an asymptotic property, and thus it does not imply much about the properties of networks learned with limited amounts of data. Nonetheless, the proof illustrates the trade-offs that are playing a role in the definition of score.

18.3.6 Priors

Until now we did not specify the actual choice of priors we use. We now discuss possible choices of priors and their effect on the score.

18.3.6.1 Structure Priors

structure prior

We begin with the *prior* over network structures, $P(\mathcal{G})$. Note that although this term seems to describe our bias for certain structure, in fact, it plays a relatively minor role. As we can see in theorem 18.1, the logarithm of the marginal likelihood grows linearly with the number of examples, while the prior over structures remains constant. Thus, the structure prior does not play an important role in asymptotic analysis as long as it does not rule out (that is, assign probability 0) any structure.

For this reason, we often use a uniform prior over structures. Nonetheless, the structure prior can make some difference when we consider small samples. Thus, we might want to encode some of our preferences in this prior. For example, we might penalize edges in the graph, and use a prior $P(\mathcal{G}) \propto c^{|\mathcal{G}|}$, where c is some constant smaller than 1, and $|\mathcal{G}|$ is the number of edges in the graph.

Note that in both these choices (the uniform and the penalty per edge) it suffices to use a value that is proportional to the prior, since the normalizing constant is the same for all choice of $\mathcal{G}$ and hence can be ignored. For this reason, we do not need to worry about the exact number of possible network structures in order to use these priors.

structure modularity

As we will immediately see, it will be mathematically convenient to assume that the structure prior satisfies *structure modularity*. This condition requires that the prior $P(\mathcal{G})$ be proportional to a product of terms, where each term relates to one family. Formally,

$$P(\mathcal{G}) \propto \prod_i P(\mathrm{Pa}_{X_i} = \mathrm{Pa}^{\mathcal{G}}_{X_i}),$$

where $P(\mathrm{Pa}_{X_i} = \mathrm{Pa}^{\mathcal{G}}_{X_i})$ denotes the prior probability we assign to choosing the specific set of parents for X_i. Structure priors that satisfy this property do not penalize for global properties of the graph (such as its depth) but only for local properties (such as the indegrees of variables). This is clearly the case for both priors we discuss here.

In addition, it also seems reasonable to require that I-equivalent network structures are assigned the same prior. Again, this means that when two networks are equivalent, we do not distinguish between them by subjective preferences.

18.3.6.2 Parameter Priors and Score Decomposability

parameter prior

In order to use Bayesian scores, we also need to have *parameter priors* for the parameterization corresponding to every possible structure. Before we discuss how to represent such priors, we consider the desired properties from these priors.

decomposable score

global parameter independence

Proposition 18.2 shows that the Bayesian score of a network structure $\mathcal{G}$ *decomposes* into a product of terms, one for each family. This is a consequence of the *global parameter independence* assumption. In the case of parameter learning, this assumption was crucial for decomposing the learning problem into independent subproblems. Can we exploit a similar phenomenon in the case of structure learning?

In the simple example we considered in the previous section, we compared the score of two networks $\mathcal{G}_\emptyset$ and $\mathcal{G}_{X\to Y}$. We saw that if we choose the priors $P(\Theta_X \mid \mathcal{G}_\emptyset)$ and $P(\Theta_X \mid \mathcal{G}_{X\to Y})$ to be identical, the score associated with X is the same in both graphs. Thus, not only does the score of both structures have a product form, but in the case where the same variable has the same parents in both structures, the term associated with it also has the same value in both scores.

Considering more general structures, if $\mathrm{Pa}_{X_i}^{\mathcal{G}} = \mathrm{Pa}_{X_i}^{\mathcal{G}'}$ then it would seem natural that the term that measures the score of X_i given its parents in $\mathcal{G}$ would be identical to the one in $\mathcal{G}'$. This seems reasonable. Recall that the score associated with X_i measures how well it can be predicted given its parents. Thus, if X_i has the same set of parents in both structures, this term should have the same value.

Definition 18.2

decomposable score

A structure score $\mathrm{score}(\mathcal{G} : \mathcal{D})$ *is* decomposable *if the score of a structure* $\mathcal{G}$ *can be written as*

$$\mathrm{score}(\mathcal{G} : \mathcal{D}) = \sum_i \mathrm{FamScore}(X_i \mid \mathrm{Pa}_{X_i}^{\mathcal{G}} : \mathcal{D}),$$

family score

where the family score $\mathrm{FamScore}(X \mid \boldsymbol{U} : \mathcal{D})$ *is a score measuring how well a set of variables* $\boldsymbol{U}$ *serves as parents of* X *in the data set* $\mathcal{D}$. ■

As an example, the likelihood score is decomposable. Using proposition 18.1, we see that in this decomposition

$$\mathrm{FamScore}_L(X \mid \boldsymbol{U} : \mathcal{D}) = M \cdot \left[\boldsymbol{I}_{\hat{P}}(X;\boldsymbol{U}) - \boldsymbol{H}_{\hat{P}}(X)\right].$$

Score decomposability has important ramifications when we search for structures that maximize the scores. The high-level intuition is that if we have a decomposable score, then a local change in the structure (such as adding an edge) does not change the score of other parts of the structure that remained the same. As we will see, the search algorithms we consider can exploit decomposability to reduce dramatically the computational overhead of evaluating different structures during search.

Under what conditions is the Bayesian score decomposable? It turns out that a natural restriction on the prior suffices.

Definition 18.3

parameter modularity

Let $\{P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G}) : \mathcal{G} \in \boldsymbol{\mathcal{G}}\}$ *be a set of parameter priors that satisfy global parameter independence. The prior satisfies* parameter modularity *if for each* $\mathcal{G}, \mathcal{G}'$ *such that* $\mathrm{Pa}_{X_i}^{\mathcal{G}} = \mathrm{Pa}_{X_i}^{\mathcal{G}'} = \boldsymbol{U}$, *then* $P(\boldsymbol{\theta}_{X_i|\boldsymbol{U}} \mid \mathcal{G}) = P(\boldsymbol{\theta}_{X_i|\boldsymbol{U}} \mid \mathcal{G}')$. ■

Parameter modularity states that the prior over the CPD of X_i depends only on the *local* structure of the network (that is, the set of parents of X_i), and not on other parts of the network. It is straightforward to see that parameter modularity implies that the score is decomposable.

Proposition 18.3 *Let $\mathcal{G}$ be a network structure, let $P(\mathcal{G})$ be a structure prior satisfying structure modularity, and let $P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G})$ be a parameter prior satisfying global parameter independence and parameter modularity. Then, the Bayesian score over network structures is decomposable.*

18.3.6.3 Representing Parameter Priors

How do we represent our parameter priors? The number of possible structures is superexponential, which makes it difficult to elicit separate parameters for each one. How do we elicit priors for all these networks? If we require parameter modularity, the number of different priors we need is somewhat smaller, since we need a prior for each choice of parents for each variable. This number, however, is still exponential.

A simpleminded approach is simply to take some fixed Dirichlet distribution, for example, $Dirichlet(\alpha, \ldots, \alpha)$, for every parameter, where α is a predetermined constant. A typical choice is $\alpha = 1$. This prior is often referred to as the *K2 prior*, referring to the name of the software system where it was first used.

K2 prior

The K2 prior is simple to represent and efficient to use. However, it is somewhat inconsistent. Consider a structure where the binary variable Y has no parents. If we take $Dirichlet(1, 1)$ for θ_Y, we are in effect stating that our imaginary sample size is two. But now, consider a different structure where Y has the parent X, which has 4 values. If we take $Dirichlet(1, 1)$ as our prior for all parameters $\theta_{Y|x^i}$, we are effectively stating that we have seen two imaginary samples in each context x^i, for a total of eight. It seems that the number of imaginary samples we have seen for different events is a basic concept that should not vary with different candidate structures.

A more elegant approach is one we already saw in the context of parameter estimation: the *BDe prior*. We elicit a prior distribution P' over the entire probability space and an equivalent sample size α for the set of imaginary samples. We then set the parameters as follows:

BDe prior

$$\alpha_{x_i|\mathrm{pa}_{X_i}} = \alpha \cdot P'(x_i, \mathrm{pa}_{X_i}).$$

This choice will avoid the inconsistencies we just discussed. If we consider the prior over $\theta_{Y|x^i}$ in our example, then

$$\alpha_y = \alpha \cdot P'(y) = \sum_{x^i} \alpha \cdot P'(y, x^i) = \sum_{x^i} \alpha_{y|x^i}.$$

Thus, the number of imaginary samples for the different choices of parents for Y will be identical.

As we discussed, we can represent P' as a Bayesian network whose structure can represent our prior about the domain structure. Most simply, when we have no prior knowledge, we set P' to be the uniform distribution, that is, the empty Bayesian network with a uniform marginal distribution for each variable. In any case, it is important to note that the network structure is used *only* to provide parameter priors. It is not used to guide the structure search directly.

18.3.7 Score Equivalence ⋆

The BDe score turns out to satisfy an important property. Recall that two networks are I-equivalent if they encode the same set of independence statements. Hence, based on observed independencies, we cannot distinguish between I-equivalent networks. This suggests that based on observing data cases, we do not expect to distinguish between equivalent networks.

Definition 18.4 score equivalence

Let $\text{score}(\mathcal{G} : \mathcal{D})$ *be some scoring rule. We say that it satisfies* score equivalence *if for all I-equivalent networks* $\mathcal{G}$ *and* $\mathcal{G}'$ *we have* $\text{score}(\mathcal{G} : \mathcal{D}) = \text{score}(\mathcal{G}' : \mathcal{D})$ *for all data sets* $\mathcal{D}$. ■

In other words, score equivalence implies that all networks in the same equivalence class have the same score. In general, if we view I-equivalent networks as equally good at describing the same probability distributions, then we want to have score equivalence. We do not want the score to introduce artificial distinctions when we choose networks.

Do the scores discussed so far satisfy this condition?

Theorem 18.3

The likelihood score and the BIC score satisfy score equivalence.

For a proof, see exercise 18.8 and exercise 18.9

What about the Bayesian score? It turns out that the simpleminded K2 prior we discussed is *not* score-equivalent; see exercise 18.10. The BDe score, on the other hand, is score-equivalent. In fact, something stronger can be said.

Theorem 18.4

Let $P(\mathcal{G})$ *be a structure prior that assigns I-equivalent networks identical prior. Let* $P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G})$ *be a prior over parameters for networks with table-CPDs that satisfies global and local parameter independence and where for each* X_i *and* $\boldsymbol{u}_i \in Val(\text{Pa}_{X_i}^{\mathcal{G}})$*, we have that* $P(\boldsymbol{\theta}_{X_i|\boldsymbol{u}_i} \mid \mathcal{G})$ *is a Dirichlet prior. The Bayesian score with this prior satisfies score equivalence if and only if the prior is a BDe prior for some choice of* α *and* P'.

We do not prove this theorem here. See exercise 18.11 for a proof that the BDe score in this case satisfies score equivalence.

In other words, if we insist on using Dirichlet priors and also want the decomposition property, then to satisfy score equivalence, we must use a BDe prior.

18.4 Structure Search

In the previous section, we discussed scores for evaluating the quality of different candidate Bayesian network structures. These included the likelihood score, the Bayesian score, and the BIC score (which is an asymptotic approximation of the Bayesian score). We now examine how to find a structure with a high score.

We now have a well-defined optimization problem. Our input is:

- training set $\mathcal{D}$;
- scoring function (including priors, if needed);
- a set $\boldsymbol{\mathcal{G}}$ of possible network structures (incorporating any prior knowledge).

Our desired output is a network structure (from the set of possible structures) that maximizes the score.

It turns out that, for this discussion, we can ignore the specific choice of score. Our search algorithms will apply unchanged to all three of these scores.

score decomposability

As we will discuss, the main property of the scores that affect the search is their *decomposability*. That is, we assume we can write the score of a network structure $\mathcal{G}$:

$$\text{score}(\mathcal{G} : \mathcal{D}) = \sum_i \text{FamScore}(X_i \mid \text{Pa}_{X_i}^{\mathcal{G}} : \mathcal{D}).$$

score equivalence

Another property that is shared by all these scores is *score equivalence*: if $\mathcal{G}$ is I-equivalent to $\mathcal{G}'$ then $\text{score}(\mathcal{G} : \mathcal{D}) = \text{score}(\mathcal{G}' : \mathcal{D})$. This property is less crucial for search, but, as we will see, it can simplify several points.

18.4.1 Learning Tree-Structured Networks

We begin with the simplest variant of the structure learning task — the task of learning a *tree-structured network*. More precisely:

Definition 18.5 tree network

A network structure $\mathcal{G}$ is called tree-structured *if each variable X has at most one parent in $\mathcal{G}$, that is,* $|\text{Pa}_X^{\mathcal{G}}| \leq 1$. ■

Strictly speaking, the notion of tree-structured networks covers a broader class of graphs than those comprising a single tree; it also covers graphs composed of a set of disconnected trees, that is, a forest. In particular, the network of independent variables (no edges) also satisfies this definition. However, as the basic structure of these networks is still a collection of trees, we continue to use the term tree-structure.

Note that the class of trees is narrower than the class of polytrees that we discussed in chapter 9. A polytree can have variables with multiple parents, whereas a tree cannot. In other words, a tree-structured network cannot have v-structures. In fact, the problem of learning polytree-structured networks has very different computational properties than that of learning trees (see section 18.8).

Why do we care about learning trees? Most importantly, because unlike richer classes of structures, they can be learned efficiently — in polynomial time. But learning trees can also be useful in themselves. They are sparse, and therefore they avoid most of the overfitting problems associated with more complex structures. They also capture the most important dependencies in the distribution, and they can therefore provide some insight into the domain. They can also provide a better baseline for approximating the distribution than the set of independent marginals of the different variables (another commonly used simple approximation). They are thus often used as a starting point for learning a more complex structure, or even on their own in cases where we cannot afford significant computational resources.

The key properties we are going to use for learning trees are the decomposability of the score on one hand and the restriction on the number of parents on the other hand. We start by examining the score of a network and performing slight manipulations. Instead of maximizing the score of a tree structure $\mathcal{G}$, we will try to maximize the difference between its score and the score of the empty structure $\mathcal{G}_\emptyset$. We define

$$\Delta(\mathcal{G}) = \text{score}(\mathcal{G} : \mathcal{D}) - \text{score}(\mathcal{G}_\emptyset : \mathcal{D}).$$

We know that $\text{score}(\mathcal{G}_\emptyset : \mathcal{D})$ is simply a sum of terms $\text{FamScore}(X_i : \mathcal{D})$ for each X_i. That is the score of X_i if it does not have any parents. The score $\text{score}(\mathcal{G} : \mathcal{D})$ consists of terms $\text{FamScore}(X_i \mid \text{Pa}_{X_i}^{\mathcal{G}} : \mathcal{D})$. Now, there are two cases. If $\text{Pa}_{X_i}^{\mathcal{G}} = \emptyset$, then the term for X_i in both scores cancel out. If $\text{Pa}_{X_i}^{\mathcal{G}} = X_j$, then we are left with the difference between the two terms. Thus, we conclude that

$$\Delta(\mathcal{G}) = \sum_{i, \text{Pa}_{X_i}^{\mathcal{G}} \neq \emptyset} \left(\text{FamScore}(X_i \mid \text{Pa}_{X_i}^{\mathcal{G}} : \mathcal{D}) - \text{FamScore}(X_i : \mathcal{D})\right).$$

If we define the weight

$$w_{j \to i} = \text{FamScore}(X_i \mid X_j : \mathcal{D}) - \text{FamScore}(X_i : \mathcal{D}),$$

then we see that $\Delta(\mathcal{G})$ is the sum of weights on pairs X_i, X_j such that $X_j \to X_i$ in $\mathcal{G}$

$$\Delta(\mathcal{G}) = \sum_{X_j \to X_i \in \mathcal{G}} w_{j \to i}.$$

maximum weight spanning forest

We have transformed our problem to one of finding a *maximum weight spanning forest* in a directed weighted graph. Define a fully connected directed graph, where each vertex is labeled by a random variable in $\mathcal{X}$, and the weight of the edge from vertex X_j to vertex X_i is $w_{j \to i}$, and then search for a maximum-weight-spanning forest. Clearly, the sum of edge weights in a forest is exactly $\Delta(\mathcal{G})$ of the structure $\mathcal{G}$ with the corresponding set of edges. The graph structure that corresponds to that maximum-weight forest maximizes $\Delta(\mathcal{G})$.

How hard is the problem of finding a maximal-weighted directed spanning tree? It turns out that this problem has a polynomial-time algorithm. This algorithm is efficient but not simple.

The task becomes simpler if the score satisfies score equivalence. In this case, we can show (see exercise 18.13) that $w_{i \to j} = w_{j \to i}$. Thus, we can examine an undirected spanning tree (forest) problem, where we choose which edges participate in the forest, and only afterward determine their direction. (This can be done by choosing an arbitrary root and directing all edges away from it.) Finding a maximum spanning tree in undirected graph is an easy problem. One algorithm for solving it is shown in algorithm A.2; an efficient implementation of this algorithm requires time complexity of $O(n^2 \log n)$, where n is the number of vertices in the graph.

Using this reduction, we end up with an algorithm whose complexity is $O(n^2 \cdot M + n^2 \log n)$ where n is the number of variables in $\mathcal{X}$ and M is the number of data cases. This complexity is a result of two stages. In the first stage we perform a pass over the data to collect the sufficient statistics of each of the $O(n^2)$ edges. This step takes $O(n^2 \cdot M)$ time. The spanning tree computation requires $O(n^2 \log n)$ using standard data structures, but it can be reduced to $O(n^2 + n \log n) = O(n^2)$ using more sophisticated approaches. We see that the first stage dominates the complexity of the algorithm.

18.4.2 Known Order

We now consider a special case that also turns out to be easier than the general case. Suppose we restrict attention to structures that are consistent with some predetermined *variable ordering* $\prec$ over $\mathcal{X}$. In other words, we restrict attention to structures $\mathcal{G}$ where, if $X_i \in \text{Pa}_{X_j}^{\mathcal{G}}$, then $X_i \prec X_j$.

variable ordering

This assumption was a standard one in the early work on learning Bayesian networks from data. In some domains the ordering is indeed known in advance. For example, if there is a clear temporal order by which the variables are assigned values, then it is natural to try to learn a network that is consistent with the temporal flow.

Before we proceed, we stress that choosing an ordering in advance may be problematic. As we have seen in the discussion of minimal I-maps in section 3.4, a wrong choice of order can result in unnecessarily complicated I-map. Although learning does not recover an exact I-map, the same reasoning applies. Thus, a bad choice of order can result in poor learning result.

With this caveat in mind, assume that we select an ordering $\prec$; without loss of generality, assume that our ordering is $X_1 \prec X_2 \prec \ldots \prec X_n$. We want to learn a structure that maximizes the score, but so that $\mathrm{Pa}_{X_i} \subseteq \{X_1, \ldots, X_{i-1}\}$.

The first observation we make is the following. We need to find the network that maximizes the score. This score is a sum of local scores, one per variable. Note that the choice of parents for one variable, say X_i, does not restrict the choice of parents of another variable, say X_j. Since we obey the ordering, none of our choices can create a cycle. Thus, in this scenario, learning the parents of each variable is independent of the other variables. Stated more formally:

Proposition 18.4 *Let $\prec$ be an ordering over $\mathcal{X}$, and let $\mathrm{score}(\mathcal{G} : \mathcal{D})$ be a decomposable score. If we choose $\mathcal{G}$ to be the network where*

$$\mathrm{Pa}_{X_i}^{\mathcal{G}} = \arg \max_{\boldsymbol{U}_i \subseteq \{X_j : X_j \prec X_i\}} \mathrm{FamScore}(X_i \mid \boldsymbol{U}_i : \mathcal{D})$$

for each i, then $\mathcal{G}$ maximizes the score among the structures consistent with $\prec$.

Based on this observation, we can learn the parents for each variable independently from the parents of other variables. In other words, we now face n small learning problems.

Let us consider these learning problems. Clearly, we are forced to make X_1 a root. In the case of X_2 we have a choice. We can either have the edge $X_1 \rightarrow X_2$ or not. In this case, we can evaluate the difference in score between these two options, and choose the best one. Note that this difference is exactly the weight $w_{1\rightarrow 2}$ we defined when learned tree networks. If $w_{1\rightarrow 2} > 0$, we add the edge $X_1 \rightarrow X_2$; otherwise we do not.

Now consider X_3. Now we have four options, corresponding to whether we add the edge $X_1 \rightarrow X_3$, and whether we add the edge $X_2 \rightarrow X_3$. A naive approach to making these choices is to decouple the decision whether to add the edge $X_1 \rightarrow X_3$ from the decision about the edge $X_2 \rightarrow X_3$. Thus, we might evaluate $w_{1\rightarrow 3}$ and $w_{2\rightarrow 3}$ and based on these two numbers try to decide what is the best choice of parents.

Unfortunately, this approach is flawed. In general, the score $\mathrm{FamScore}(X_3 \mid X_1, X_2 : \mathcal{D})$ is *not* a function of $\mathrm{FamScore}(X_3 \mid X_1 : \mathcal{D})$ and $\mathrm{FamScore}(X_3 \mid X_2 : \mathcal{D})$. An extreme example is an XOR-like CPD where X_3 is a probabilistic function of the XOR of X_1 and X_2. In this case, $\mathrm{FamScore}(X_3 \mid X_1 : \mathcal{D})$ will be small (and potentially smaller than $\mathrm{FamScore}(X_3 \mid : \mathcal{D})$ since the two variables are independent), yet $\mathrm{FamScore}(X_3 \mid X_1, X_2 : \mathcal{D})$ will be large. By choosing the particular dependence of X_3 on the XOR of X_1 and X_2, we can change the magnitude of the latter term.

We conclude that we need to consider all four possible parent sets before we choose the parents of X_3. This does not seem that bad. However, when we examine X_4 we need to

consider eight parent sets, and so on. For learning the parents of X_n, we need to consider 2^{n-1} parent sets, which is clearly too expensive for any realistic number of variables.

In practice, we do not want to learn networks with a large number of parents. Such networks are expensive to represent, most often are inefficient to perform inference with, and, most important, are prone to overfitting. So, we may find it reasonable to restrict our attention to networks the indegree of each variable is at most d.

If we make this restriction, our situation is somewhat more reasonable. The number of possible parent sets for X_n is $1 + \binom{n-1}{1} + \ldots + \binom{n-1}{d} = O(d\binom{n-1}{d})$ (when $d < n/2$). Since the number of choices for all other variables is less than the number of choices for X_n, the procedure has to evaluate $O(dn\binom{n-1}{d}) = O(d\binom{n}{d})$ candidate parent sets. This number is polynomial in n (for a fixed d).

We conclude that learning given a fixed order and a bound on the indegree is computationally tractable. However, the computational cost is exponential in d. Hence, the exhaustive algorithm that checks all parent sets of size $\leq d$ is impractical for values of d larger than 3 or 4. When a larger d is required, we can use heuristic methods such as those described in the next section.

18.4.3 General Graphs

What happens when we consider the most general problem, where we do not have an ordering over the variables? Even if we restrict our attention to networks with small indegree, other problems arise. Suppose that adding the edge $X_1 \rightarrow X_2$ is beneficial, for example, if the score of X_1 as a parent of X_2 is higher than all other alternatives. If we decide to add this edge, we cannot add other edges — for example, $X_2 \rightarrow X_1$ — since this would introduce a cycle. The restriction on the immediate reverse of the edge we add might not seem so problematic. However, adding this edge also forbids us from adding together pairs of edges, such as $X_2 \rightarrow X_3$ and $X_3 \rightarrow X_1$. Thus, the decision on whether to add $X_1 \rightarrow X_2$ is not simple, since it has ramifications for other choices we make for parents of all the other variables.

This discussion suggests that the problem of finding the maximum-score network might be more complex than in the two cases we examined. If fact, we can make this statement more precise. Let d be an integer, we define $\boldsymbol{\mathcal{G}}_d = \{\mathcal{G} : \forall i, |\mathrm{Pa}^{\mathcal{G}}_{X_i}| \leq d\}$.

Theorem 18.5 *The following problem is $\mathcal{NP}$-hard for any $d \geq 2$:*

Given a data set $\mathcal{D}$ and a decomposable score function score, *find*

$$\mathcal{G}^* = \arg\max_{\mathcal{G} \in \boldsymbol{\mathcal{G}}_d} \mathrm{score}(\mathcal{G} \ : \ \mathcal{D}).$$

The proof of this theorem is quite elaborate, and so we do not provide it here.

Given this result, we realize that it is unlikely that there is an efficient algorithm that constructs the highest-scoring network structure for all input data sets. Unlike the situation in inference, for example, the known intermediate situations where the problem is easier are not the ones we usually encounter in practice; see exercise 18.14.

As with many intractable problems, this is not the end of the story. Instead of aiming for an algorithm that will always find the highest-scoring network, we resort to heuristic algorithms that attempt to find the best network but are not guaranteed to do so. In our case, we are

local search

faced with a combinatorial optimization problem; we need to search the space of graphs (with bounded indegree) and return a high-scoring one. We solve this problem using a *local search* approach. To do so, we define three components: a search space, which defines the set of candidate network structures; a scoring function that we aim to maximize (for example, the BDe score given the data and priors); and the search procedure that explores the search space without necessarily seeing all of it (since it is superexponential in size).

18.4.3.1 The Search Space

search space

We start by considering the *search space*. As discussed in appendix A.4.2, we can think of a search space as a graph over candidate solutions, connected by possible operators that the search procedure can perform to move between different solutions. In the simplest setting, we consider the search space where each search state denotes a complete network structure $\mathcal{G}$ over $\mathcal{X}$. This is the search space we discuss for most of this chapter. However, we will see other formulations for search spaces.

A crucial design choice that has large impact on the success of heuristic search is how the space is interconnected. If each state has few neighbors, then the search procedure has to consider only a few options at each point of the search. Thus, it can afford to evaluate each of these options. However, this comes at a price. Paths from the initial solution to a good one might be long and complex. On the other hand, if each state has many neighbors, we may be able to move quickly from the initial state to a good state, but it may be difficult to determine which step to take at each point in the search. A good trade-off for this problem chooses reasonably few neighbors for each state but ensures that the "diameter" of the search space remains small. A natural choice for the neighbors of a state representing a network structure is a set of structures that are identical to it except for small "local" modifications. Thus, we define the connectivity of our search space in terms of *operators* such as:

search operators

edge addition

edge deletion

edge reversal

- *edge addition*;
- *edge deletion*;
- *edge reversal*.

In other words, the states adjacent to a state $\mathcal{G}$ are those where we change one edge, either by adding one, deleting one, or reversing the orientation of one. Note that we only consider operations that result in legal networks. That is, acyclic networks that satisfy the constraints we put in advance (such as indegree constraints).

This definition of search space is quite natural and has several desirable properties. First, notice that the diameter of the search space is at most n^2. That is, there is a relatively short path between any two networks we choose. To see this, note that if we consider traversing a path from $\mathcal{G}_1$ to $\mathcal{G}_2$, we can start by deleting all edges in $\mathcal{G}_1$ that do not appear in $\mathcal{G}_2$, and then we can add the edges that are in $\mathcal{G}_2$ and not in $\mathcal{G}_1$. Clearly, the number of steps we take is bounded by the total number of edges we can have, n^2.

Second, recall that the score of a network $\mathcal{G}$ is a sum of local scores. The operations we consider result in changing only one local score term (in the case of addition or deletion of an edge) or two (in the case of edge reversal). Thus, they result in a local change in the score; most components in the score remain the same. This implies that there is some sense of "continuity" in the score of neighboring networks.

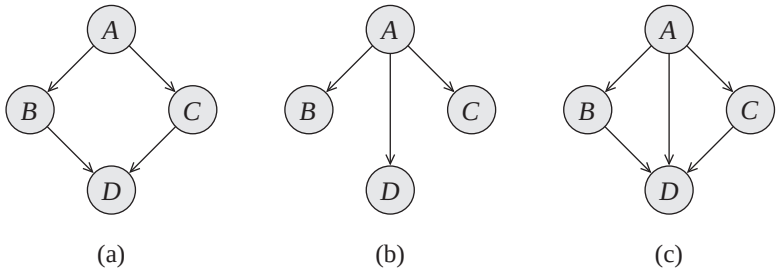


Figure 18.5 Example of a search problem requiring edge deletion. (a) original network that generated the data. (b) and (c) intermediate networks encountered during the search.

The choice of the three particular operations we consider also needs some justification. For example, if we always start the search from the empty graph $\mathcal{G}_\emptyset$, we may wonder why we include the option to delete an edge. We can reach every network by adding the appropriate arcs to the empty network. In general, however, we want the search space to allow us to reverse our choices. As we will see, this is an important property in escaping local maxima (see appendix A.4.2).

However, the ability to delete edges is important even if we perform only "greedy" operations that lead to improvement. To see this, consider the following example. Suppose the original network is the one shown in figure 18.5a, and that A is highly informative about both C and B. Starting from an empty network and adding edges greedily, we might add the edges $A \rightarrow B$ and $A \rightarrow C$. However, in some data sets, we might add the edge $A \rightarrow D$. To see why, we need to realize that A is informative about both B and C, and since these are the two parents of D, also about D. Now B and C are also informative about D. However, each of them provides part of the information, and thus neither B nor C by itself is the best parent of D. At this stage, we thus end up with the network figure 18.5b. Continuing the search, we consider different operators, adding the edge $B \rightarrow D$ and $C \rightarrow D$. Since B is a parent of D in the original network, it will improve the prediction of D when combined with A. Thus, the score of A and B together as parents of D can be larger than the score of A alone. Similarly, if there are enough data to support adding parameters, we will also add the edge $C \rightarrow D$, and reach the structure shown in figure 18.5c. This is the correct structure, except for the redundant edge $A \rightarrow D$. Now the ability to delete edges comes in handy. Since in the original distribution B and C together separate A from D, we expect that choosing B, C as the parents of D will have higher score than choosing A, B, C. To see this, note that A cannot provide additional information on top of what B and C convey, and having it as an additional parent results in a penalty. After we delete the edge $A \rightarrow D$ we get the original structure.

A similar question can be raised about the edge reversal operator. Clearly, we can achieve the effect of reversing an edge $X \rightarrow Y$ in two steps, first deleting the edge $X \rightarrow Y$ and then adding the edge $Y \rightarrow X$. The problem is that when we delete the edge $X \rightarrow Y$, we usually reduce the score (assuming that there is some dependency between X and Y). Thus, these two operations require us to go "downhill" in the first step in order to get to a better structure in the next step. The reverse operation allows us to realize the trade-off between a worse parent set for Y and a better one for X.

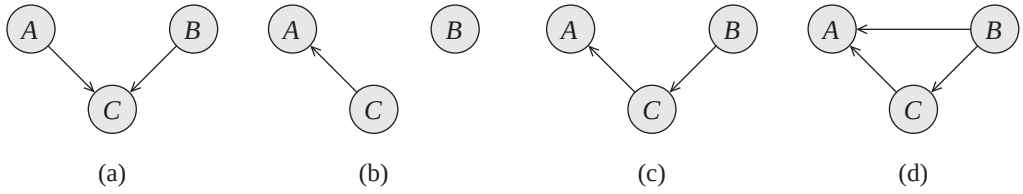


Figure 18.6 Example of a search problem requiring edge reversal. (a) original network that generated the data. (b) and (c) intermediate networks encountered during the search. (d) an undesirable outcome.

To see the utility of the edge reversal operator, consider the following simple example. Suppose the real network generating the data has the v-structure shown in figure 18.6a. Suppose that the dependency between A and C is stronger than that between B and C. Thus, a first step in a greedy-search procedure would add an edge between A and C. Note, however, that score equivalence implies that the network with the edge $A \rightarrow C$ has *exactly* the same score as the network with the edge $C \rightarrow A$. At this stage, we cannot distinguish between the two choices. Thus, the decision between them is arbitrary (or, in some implementations, randomized). It is thus conceivable that at this stage we have the network shown in figure 18.6b. The greedy procedure proceeds, and it decides to add the edge $B \rightarrow C$, resulting in the network of figure 18.6c. Now we are in the position to realize that reversing the edge $C \rightarrow A$ can improve the score (since A and B together should make the best predictions of C). However, if we do not have a reverse operator, a greedy procedure would not delete the edge $C \rightarrow A$, since that would definitely hurt the score.

In this example, note that when we do not perform the edge reversal, we might end up with the network shown in figure 18.6d. To realize why, recall that although A and B are marginally independent, they are dependent given C. Thus, B and C together make better predictions of A than C alone.

18.4.3.2 The Search Procedure

local search

Once we define the search space, we need to design a procedure to explore it and search for high-scoring states. There is a wide literature on heuristic search. The vast majority of the search methods used in structure learning are *local search* procedures such as greedy hill climbing, as described in appendix A.4.2. In the structure-learning setting, we pick an initial network structure $\mathcal{G}$ as a starting point; this network can be the empty one, a random choice, the best tree, or a network obtained from some prior knowledge. We compute its score. We then consider all of the neighbors of $\mathcal{G}$ in the space — all of the legal networks obtained by applying a single operator to $\mathcal{G}$ — and compute the score for each of them. We then apply the change that leads to the best improvement in the score. We continue this process until no modification improves the score.

There are two questions we can ask. First, how expensive is this process, and second, what can we say about the final network it returns?

Computational Cost We start by briefly considering the time complexity of the procedure. At each iteration, the procedure applies $|\mathcal{O}|$ operators and evaluates the resulting network. Recall that the space of operators we consider is quadratic in the number of variables. Thus, if we perform K steps before convergence, then we perform $O(K \cdot n^2)$ operator applications. Each operator application involves two steps. First, we need to check that the network is acyclic. This check can be done in time linear in the number of edges. If we are considering networks with indegree bounded by d, then there are at most nd edges. Second, if the network is legal, we need to evaluate it. For this, we need to collect sufficient statistics from the data. These might be different for each network and require $O(M)$ steps, and so our rough time estimate is $O(K \cdot n^2 \cdot (M + nd))$. The number of iterations, K, varies and depends on the starting network and on how different the final network is. However, we expect it not to be much larger than n^2 (since this is the diameter of the search space). We emphasize that this is a rough estimate, and not a formal statement. As we will show, we can make this process faster by using properties of the score that allow for smart caching.

When n is large, considering $O(n^2)$ neighbors at each iteration may be too costly. However, most operators attempt to perform a rather bad change to the network. So can we skip evaluating them? One way of avoiding this cost is to use search procedures that replace the exhaustive enumeration in line 5 of Greedy-Local-Search (algorithm A.5) by a randomized choice of operators. This *first-ascent hill climbing* procedure samples operators from $\mathcal{O}$ and evaluates them one by one. Once it finds one that leads to a better-scoring network, it applies it without considering other operators. In the initial stages of the search, this procedure requires relatively few random trials before it finds such an operator. As we get closer to the local maximum, most operators hurt the score, and more trials are needed before an upward step is found (if any).

first-ascent hill climbing

Local Maxima What can we say about the network returned by a greedy hill-climbing search procedure? Clearly, the resulting network cannot be improved by applying a single operator (that is, changing one edge). This implies that we are in one of two situations. We might have reached a *local maximum* from which all changes are score-reducing. The other option is that we have reached a *plateau*: a large set of neighboring networks that have the same score. By design, the greedy hill-climbing procedure cannot "navigate" through a plateau, since it relies on improvement in score to guide it to better structures.

local maximum

plateau

Upon reflection, we realize that greedy hill climbing will encounter plateaus quite often. Recall that we consider scores that satisfy score equivalence. Thus, all networks in an I-equivalence class will have the same score. Moreover, as shown in theorem 3.9, the set of I-equivalent networks forms a contiguous region in the space, which we can traverse using a set of covered edge-reversal operations. Thus, any I-equivalence class necessarily forms a plateau in the search space.

I-equivalence

Recall that equivalence classes can potentially be exponentially large. Ongoing work studies the average size of an equivalence class (when considering all networks) and the actual distributions of sizes encountered in realistic situations, such as during structure search.

It is clear, however, that most networks we encounter have at least a few equivalent networks. Thus, we conclude that most often, greedy hill climbing will converge to an equivalence class. There are two possible situations: Either there is another network in this equivalence class from which we can continue the upward climb, or the whole equivalence class is a local maximum. Greedy hill climbing cannot deal with either situation, since it cannot explore without upward

indications.

As we discussed in appendix A.4.2, there are several strategies to improve on the network $\mathcal{G}$ returned by a greedy search algorithm. One approach that deals with plateaus induced by equivalence classes is to enumerate explicitly all the network structures that are I-equivalent to $\mathcal{G}$, and for each one to examine whether it has neighbors with higher score. This enumeration, however, can be expensive when the equivalence class is large. An alternative solution, described in section 18.4.4, is to search directly over the space of equivalence classes. However, both of these approaches save us from only some of the plateaus, and not from local maxima.

basin flooding

Appendix A.4.2 describes other methods that help address problem of local maxima. For example, *basin flooding* keeps track of all previous networks and considers any operator leading from one of them to a structure that we have not yet visited. A key problem with this approach is that storing the list of networks we visited in the recent past can be expensive (recall that greedy hill climbing stores just one copy of the network). Moreover, we do not necessarily want to explore the whole region surrounding a local maximum, since it contains many variants of the same network. To see why, suppose that three different edges in a local maximum network can be removed with very little change in score. This means that all seven networks that contain at least one deletion will be explored before a more interesting change will be considered.

tabu search

A method that solves both problems is the *tabu search* of algorithm A.6. Recall that this procedure keeps a list of recent operators we applied, and in each step we do not consider operators that reverse the effect of recently applied operators. Thus, once the search decides to add an edge, say $X \rightarrow Y$, it cannot delete this edge in the next L steps (for some prechosen L). Similarly, once an arc is reversed, it cannot be reversed again. As for the basin-flooding approach, tabu search cannot use the termination criteria of greedy hill climbing. Since we want the search to proceed after reaching the local maxima, we do not want to stop when the score of the current candidate is smaller than the previous one. Instead, we continue the search with the hope of reaching a better structure. If this does not happen after a prespecified number of steps, we decide to abandon the search and select the best network encountered at any time during the search.

Finally, as we discussed, one can also use randomization to increase our chances of escaping local maxima. In the case of structure learning, these methods do help. In particular, simulated annealing was reported to outperform greedy hill-climbing search. However, in typical example domains (such as the ICU-Alarm domain) it appears that simple methods such as tabu search with random restarts find higher-scoring networks much faster.

Data Perturbation Methods So far, we have discussed only the application of general-purpose local search methods to the specific problem of structure search. We now discuss one class of methods — *data-perturbation* methods — that are more specific to the learning task. The idea is similar to random restarts: We want to perturb the search in a way that will allow it to overcome local obstacles and make progress toward the global maxima. Random restart methods achieve this perturbation by changing the network. Data perturbation methods, on the other hand, change the training data.

data perturbation

To understand the idea, consider a perturbation that duplicates some instances (say by random choice) and removes others (again randomly). If we do a reasonable number of these modifications, the resulting data set $\mathcal{D}'$ has most of the characteristics of the original data set $\mathcal{D}$. For example, the value of sufficient statistics in the perturbed data are close

Algorithm 18.1 Data perturbation search

```
Procedure Search-with-Data-Perturbation (
    G_∅,    // initial network structure
    D       // Fully observed data set
    score,  // Score
    O,      // A set of search operator
    Search, // Search procedure
    t_0,    // Initial perturbation size
    γ,      // Reduction in perturbation size

)

1   G ← Search(G_∅, D, score, O)
2   G_best ← G
3   t ← t_0
4   for i = 1, ... until convergence
5     D' ← Perturb(D, t)
6     G ← Search(G, D', score, O)
7     if score(G : D) > score(G_best : D) then
8       G_best ← G
9     t ← γ · t
10
11  return G_best
```

to the values in the original data. Thus, we expect that big differences between networks are preserved. That is, if $\text{score}(\mathcal{G}_{X\to Y} : \mathcal{D}) \gg \text{score}(\mathcal{G}_2 : \mathcal{D})$, then we expect that $\text{score}(\mathcal{G}_{X\to Y} : \mathcal{D}') \gg \text{score}(\mathcal{G}_2 : \mathcal{D}')$. On the other hand, the perturbation does change the comparison between networks that are similar. The basic intuition is that the score using $\mathcal{D}'$ has the same broad outline as the score using $\mathcal{D}$, yet might have different fine-grained topology. This suggests that a structure $\mathcal{G}$ that is a local maximum when using the score on $\mathcal{D}$ is no longer a local maximum when using $\mathcal{D}'$. The magnitude of perturbation determines the level of details that are preserved after the perturbation.

weighted data instances

We note that instead of duplicating and removing instances, we can achieve perturbation by *weighting* data instances. Much of the discussion on scoring networks and related topics applies without change if we assign weight to each instance. Formally, the only difference is the computation of sufficient statistics. If we have weights $w[m]$ for the m'th instance, then the sufficient statistics are redefined as:

$$M[\boldsymbol{z}] = \sum_m \mathbf{1}\{\boldsymbol{Z}[m] = \boldsymbol{z}\} \cdot w[m].$$

Note that when $w[m] = 1$, this reduces to the standard definition of sufficient statistics. Instance duplication and deletion lead to integer weights. However, we can easily consider perturbation that results in fractional weights. This leads to a continuous spectrum of data perturbations that

range from small changes to weights to drastic ones.

The actual search procedure is shown in algorithm 18.1. The heart of the procedure is the Perturb function. This procedure can implemented in different ways. A simple approach is to sample each $w[m]$ for a distribution whose variance is dictated by t, for example, using a Gamma distribution, with mean 1 and variance t. (Note that we need to use a distribution that attains nonnegative values. Thus, the Gamma distribution is more suitable than a Gaussian distribution.)

18.4.3.3 Score Decomposition and Search

The discussion so far has examined how generic ideas in heuristic search apply to structure learning. We now examine how the particulars of the problem impact the search.

The dominant factor in the cost of the search algorithm is the evaluation of neighboring networks at each stage. As discussed earlier, the number of such networks is approximately n^2. To evaluate each of these network structures, we need to score them. This process requires that we traverse all the different data cases, computing sufficient statistics relative to our new structure. This computation can get quite expensive, and it is the dominant cost in any structure learning algorithm.

score decomposability

This key task is where the *score decomposability* property turns out to be useful. Recall that the scores we examine decompose into a sum of terms, one for each variable X_i. Each of these family scores is computed relative only to the variables in the family of X_i. A local change — adding, deleting, or reversing an edge — leaves almost all of the families in the network unchanged. (Adding and deleting changes one family, and reversing changes two.) For families whose composition does not change, the associated component of the score also does not change. To understand the importance of this observation, assume that our current candidate network is $\mathcal{G}$. For each operator, we compute the improvement in the score that would result in making that change. We define the *delta score*

delta score

$$\delta(\mathcal{G} : o) = \text{score}(o(\mathcal{G}) : \mathcal{D}) - \text{score}(\mathcal{G} : \mathcal{D})$$

to be the change of score associated with applying o on $\mathcal{G}$. Using score decomposition, we can compute this quantity relatively efficiently.

Proposition 18.5 *Let $\mathcal{G}$ be a network structure and* score *be a decomposable score.*

- *If o is "Add $X \to Y$," and $X \to Y \notin \mathcal{G}$, then*

$$\delta(\mathcal{G} : o) = \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} \cup \{X\} : \mathcal{D}) - \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} : \mathcal{D}).$$

- *If o is "Delete $X \to Y$" and $X \to Y \in \mathcal{G}$, then*

$$\delta(\mathcal{G} : o) = \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} - \{X\} : \mathcal{D}) - \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} : \mathcal{D}).$$

- *If o is "Reverse $X \to Y$" and $X \to Y \in \mathcal{G}$, then*

$$\begin{aligned}\delta(\mathcal{G} : o) \quad = \quad & \text{FamScore}(X, \text{Pa}_X^{\mathcal{G}} \cup \{Y\} : \mathcal{D}) + \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} - \{X\} : \mathcal{D}) \\ & - \text{FamScore}(X, \text{Pa}_X^{\mathcal{G}} : \mathcal{D}) - \text{FamScore}(Y, \text{Pa}_Y^{\mathcal{G}} : \mathcal{D}).\end{aligned}$$

See exercise 18.18.

Note that these computations involve only the sufficient statistics for the particular family that changed. This requires a pass over only the appropriate columns in the table describing the training data.

Now, assume that we have an operator o, say "Add $X \to Y$," and instead of applying this edge addition, we have decided to apply another operator o' that changes the family of some variable Z (for $Z \neq Y$), producing a new graph $\mathcal{G}'$. The key observation is that $\delta(\mathcal{G}' : o)$ remains unchanged — we do not need to recompute it. We need only to recompute $\delta(\mathcal{G}' : o')$ for operators o' that involve Y.

Proposition 18.6 *Let $\mathcal{G}$ and $\mathcal{G}'$ be two network structures and* score *be a decomposable score.*

- *If o is either "Add $X \to Y$" or "Delete $X \to Y$" and $\mathrm{Pa}_Y^{\mathcal{G}} = \mathrm{Pa}_Y^{\mathcal{G}'}$, then $\delta(\mathcal{G} : o) = \delta(\mathcal{G}' : o)$.*
- *If o is "Reverse $X \to Y$," $\mathrm{Pa}_Y^{\mathcal{G}} = \mathrm{Pa}_Y^{\mathcal{G}'}$, and $\mathrm{Pa}_X^{\mathcal{G}} = \mathrm{Pa}_X^{\mathcal{G}'}$, then $\delta(\mathcal{G} : o) = \delta(\mathcal{G}' : o)$.*

See exercise 18.19.

This shows that we can cache the computed $\delta(\mathcal{G} : o)$ for different operators and then reuse most of them in later search steps. The basic idea is to maintain a data structure that records for each operator o the value of $\delta(\mathcal{G} : o)$ with respect to the current network $\mathcal{G}$. After we apply a step in the search, we have a new current network, and we need to update this data structure. Using proposition 18.6 we see that most of the computed values do not need to be changed. We need to recompute $\delta(\mathcal{G}' : o)$ only for operators that modify one of the families that we modified in the recent step. By careful data-structure design, this cache can save us a lot of computational time; see box 18.A for details. **Overall, the decomposability of the scoring function provides significant reduction in the amount of computation that we need to perform during the search. This observation is critical to making structure search feasible for high-dimensional spaces.**

Box 18.A — Skill: Practical Collection of Sufficient Statistics. *The passes over the training data required to compute sufficient statistics generally turn out to be the most computationally intensive part of structure learning. It is therefore crucial to take advantage of properties of the score in order to ensure efficient computations as well as use straightforward organizational tricks.*

One important source of computational savings derives from proposition 18.6. As we discussed, this proposition allows us to avoid recomputing many of the delta-scores after taking a step in the search. We can exploit this observation in a variety of ways. For example, if we are performing greedy hill climbing, we know that the search will necessarily examine all operators. Thus, after each step we can update the evaluation of all the operators that were "damaged" by the last move. The number of such operators is $O(n)$, and so this requires $O(n \cdot M)$ time (since we need to collect sufficient statistics from data). Moreover, if we keep the score of different operators in a heap, we spend $O(n \log n)$ steps to update the heap but then can retrieve the best operator in constant time. Thus, although the cost of a single step in the greedy hill-climbing procedure seems to involve quadratic number of operations of $O(n^2 \cdot M)$, we can perform it in time $O(n \cdot M + n \log n)$.

We can further reduce the time consumed by the collection of sufficient statistics by considering additional levels of caching. For example, if we use table-CPDs, then the counts needed for evalu-

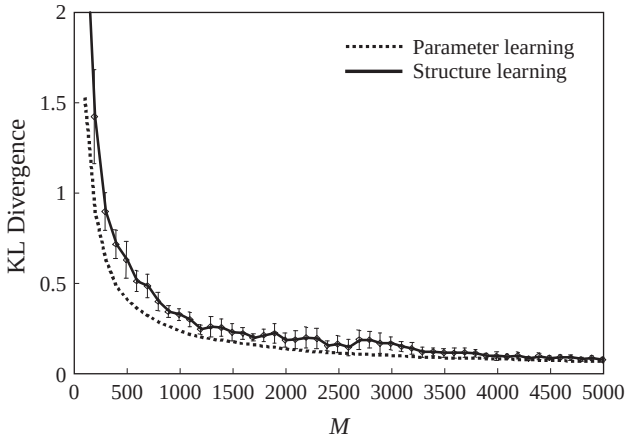


Figure 18.7 Performance of structure and parameter learning for instances generated from the ICU-Alarm **network.** The graph shows the KL-divergence of the learned to the true network, and compares two learning tasks: learning the parameters only, using a correct network structure, and learning both parameters and structure. The curves shows average performance over 10 data sets of the same size, with error bars showing +/- one standard deviation. The error bars for parameter learning are much smaller and are not shown.

ating X as a parent of Y and the counts needed to evaluate Y as a parent of X are the same. Thus, we can save time by caching previously computed counts, and also by marginalizing counts such as $M[x, y]$ to compute $M[x]$. A more elaborate but potentially very effective approach is one where we plan the collection of the entire set of sufficient statistics needed. In this case, we can use efficient algorithms for the set cover problem to choose a smaller set of sufficient statistics that covers all the needed computations. There are also efficient data structures (such as AD-trees *for discrete spaces and* KD-trees *or* metric trees *for continuous data) that are designed explicitly for maintaining and retrieving sufficient statistics; these data structures can significantly improve the performance of the algorithm, particularly when we are willing to* approximate *sufficient statistics in favor of dramatic speed improvements.*

One cannot overemphasize the importance of these seemingly trivial caching tricks. In practice, learning the structure of a network without making use of such tricks is infeasible even for a modest number of variables.

18.4.3.4 Empirical Evaluation

In practice, relatively cheap and simple algorithms, such as tabu search, work quite well. Figure 18.7 shows the results of learning a network from data generated from the ICU-Alarm network. The graph shows the KL-divergence to the true network and compares two learning tasks: learning the parameters only, using a correct network structure, and learning both parameters and structure. Although the graph does show that it is harder to recover both the structure and

the parameters, the difference in the performance achieved on the two tasks is surprisingly small. We see that structure learning is not necessarily a harder task than parameter estimation, although computationally, of course, it is more expensive. We note, however, that even the computational cost is not prohibitive. Using simple optimization techniques (such as tabu search with random restarts), learning a network with a hundred variables takes a few minutes on a standard machine.

We stress that the networks learned for different sample sizes in figure 18.7 are not the same as the original networks. They are usually simpler (with fewer edges). As the graph shows, they perform quite similarly to the real network. This means that for the given data, these networks seem to provide a better score, which means a good trade-off between complexity and fit to the data. As the graph suggests, this estimate (based on training data) is quite reasonable.

18.4.4 Learning with Equivalence Classes ⋆

The preceding discussion examined different search procedures that attempt to escape local maxima and plateaus in the search space. An alternative approach to avoid some of these pitfalls is to change the search space. In particular, as discussed, many of the plateaus we encounter during the search are a consequence of score equivalence — equivalent networks have equivalent scores. This observation suggests that we can avoid these plateaus if we consider searching over equivalence classes of networks.

To carry out this idea, we need to examine carefully how to construct the search space. Recall that an equivalence class of networks can be exponential in size. Thus, we need a compact representation of states (equivalence classes) in our search space. Fortunately, we already encountered such a representation. Recall that a *class PDAG* is a partially directed graph that corresponds to an equivalence class of networks. This representation is relatively compact, and thus, we can consider the search space over all possible class PDAGs.

class PDAG

Next, we need to answer the question how to score a given class PDAG. The scores we discussed are defined for over network structures (DAGs) and not over PDAGs. Thus, to score a class PDAG $\mathcal{K}$, we need to build a network $\mathcal{G}$ in the equivalence class represented by $\mathcal{K}$ and then score it. As we saw in section 3.4.3.3, this is a fairly straightforward procedure.

Finally, we need to decide on our search algorithm. Once again, we generally resort to a hill-climbing search using local graph operations. Here, we need to define appropriate search operations on the space of PDAGs. One approach is to use operations at the level of PDAGs. In this case, we need operations that add, remove, and reverse edges; moreover, since PDAGs contain both directed edges and undirected ones, we may wish to consider operations such as adding an undirected edge, orienting an undirected edge, and replacing a directed edge by an undirected one. An alternative approach is to use operators in DAG space that are guaranteed to change the equivalence class. In particular, consider an equivalence class $\mathcal{E}$ (represented as a class PDAG). We can define as our operators any step that takes a DAG $\mathcal{G} \in \mathcal{E}$, adds or deletes an edge from $\mathcal{G}$ to produce a new DAG $\mathcal{G}'$, and then constructs the equivalence class $\mathcal{E}'$ for $\mathcal{G}'$ (represented again as a class PDAG). Since both edge addition and edge deletion change the skeleton, we are guaranteed that $\mathcal{E}$ and $\mathcal{E}'$ are distinct equivalence classes.

GES algorithm

One algorithm based on this last approach is called the *GES algorithm*, for *greedy equivalence search*. GES starts out with the equivalence class for the empty graph and then takes greedy edge-addition steps until no additional edge-addition steps improve the score. It then executes

the reverse procedure, removing edges one at a time until no additional edge-removal steps improve the score.

consistent score

When used with a *consistent score* (as in definition 18.1), this simple two-pass algorithm has some satisfying guarantees. Assume that our distribution P^* is faithful for the graph $\mathcal{G}^*$ over $\mathcal{X}$; thus, as in section 18.2, there are no spurious independencies in P^*. Moreover, assume that we have (essentially) infinite data. Under these assumptions, our (consistent) scoring function gives only the correct equivalence class — the equivalence class of $\mathcal{G}^*$ — the highest score. For this setting, one can show that GES is guaranteed to produce the equivalence class of $\mathcal{G}^*$ as its output.

Although the assumptions here are fairly strong, this result is still important and satisfying. Moreover, empirical results suggest that GES works reasonably well even when some of its assumptions are violated (to an extent). Thus, it also provides a reasonable alternative in practice.

Although simple in principle, there are two significant computational issues associated with GES and other algorithms that work in the space of equivalence classes. The first is the cost of generating the equivalence classes that result from the local search operators discussed before. The second is the cost of evaluating their scores while reusing (to the extent possible) the sufficient statistics from our current graph. Although nontrivial (and outside the scope of this book), local operations that address both of these tasks have been constructed, making this algorithm a computationally feasible alternative to search over DAG space.

Box 18.B — Concept: Dependency Networks. *An alternative formalism for parameterizing a Markov network is by associating with each variable X_i a conditional probability distribution (CPD) $P_i(X_i \mid \mathcal{X} - \{X_i\}) = P_i(X_i \mid \mathrm{MB}_{\mathcal{H}}(X_i))$. Networks parameterized in this way are sometimes called* dependency networks *and are drawn as a cyclic directed graph, with edges to each variable from all of the variables in its Markov blanket.*

dependency networks

This representation offers certain trade-offs over other representations. In terms of semantics, a key limitation of this parameterization is that a set of CPDs $\{P_i(X_i \mid \mathrm{MB}_{\mathcal{H}}(X_i)) : X_i \in \mathcal{X}\}$ may not be consistent with any probability distribution P; that is, there may not be a distribution P such that $P_i(X_i \mid \mathrm{MB}_{\mathcal{H}}(X_i)) = P(X_i \mid \mathrm{MB}_{\mathcal{H}}(X_i))$ for all i (hence the use of the subscript i on P_i). Moreover, determining whether such a set of CPDs is consistent with some distribution P is a computationally difficult problem. Thus, eliciting or learning a consistent dependency network can be quite difficult, and the semantics of an inconsistent network is unclear.

However, in a noncausal domain, dependency networks arguably provide a more appropriate representation of the dependencies in the distribution than a Bayesian network. Certainly, for a lay user, understanding the notion of a Markov blanket in a Bayesian network is not trivial. On the other hand, in comparison to Markov networks, the CPD parameterization is much more natural and easy to understand. (As we discussed, there is no natural interpretation for a Markov network factor in isolation.)

From the perspective of inference, dependency networks provide a very easy mechanism for answering queries where all the variables except for a single query variable are observed (see box 18.C). However, answering other queries is not as obvious. The representation lends itself very nicely to Gibbs sampling, which requires precisely the distribution of individual variables given their Markov blanket. However, exact inference requires that we transform the network to a standard

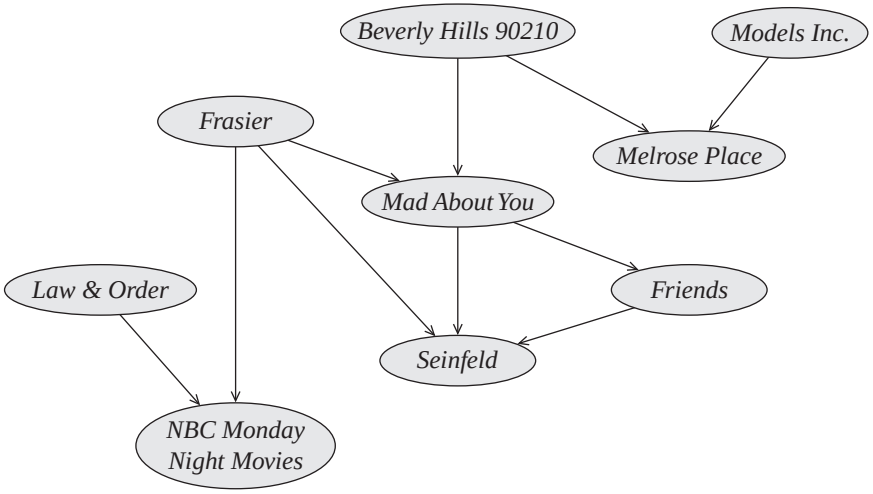


Figure 18.C.1 — Learned Bayesian network for collaborative filtering. A fragment of a Bayesian network for collaborative filtering, learned from Nielsen TV rating data, capturing the viewing record of sample viewers. The variables denote whether a TV program was watched.

parameterization, a task that requires a numerical optimization process.

The biggest advantage arises in the learning setting. If we are willing to relax the consistency requirement, the problem of learning such networks from data becomes quite simple: we simply have to learn a CPD independently for each variable, a task to which we can apply a wide variety of standard supervised learning algorithms. In this case, however, it is arguable whether the resulting network can be considered a unified probabilistic model, rather than a set of stand-alone predictors for individual variables.

Box 18.C — Case Study: Bayesian Networks for Collaborative Filtering. *In many marketing settings, we want to provide to a user a recommendation of an item that he might like, based on previous items that he has bought or liked. For example, a bookseller might want to recommend books that John might like to buy, using John's previous book purchases. Because we rarely have enough data for any single user to determine his or her preferences, the standard solution is an approach called* collaborative filtering, *which uses the observed preferences of other users to try to determine the preferences for any other user. There are many possible approaches to this problem, including ones that explicitly try to infer key aspects of a user's preference model.*

collaborative filtering

One approach is to learn the dependency structure between different purchases, as observed in the population. We treat each item i as a variable X_i in a joint distribution, and each user as an instance. Most simply, we view a purchase of an item i (or some other indication of preference) as one value for the variable X_i, and the lack of a purchase as a different value. (In certain settings,

we may get explicit ratings from the user, which can be used instead.) We can then use structure learning to obtain a Bayesian network model over this set of random variables. Dependency networks (see box 18.B) have also been used for this task; these arguably provide a more intuitive visualization of the dependency model to a lay user.

Both models can be used to address the collaborative filtering task. Given a set of purchases for a set of items S, we can compute the probability that the user would like a new item i. In general, this question is reduced to a probabilistic inference task where all purchases other than S and i are set to false; thus, all variables other than the query variable X_i are taken to be observed. In a Bayesian network, this query can be computed easily by simply looking at the Markov blanket of X_i. In a dependency network, the process is even simpler, since we need only consider the CPD for X_i. Bayesian networks and Markov networks offer different trade-offs. For example, the learning and prediction process for dependency networks is somewhat easier, and the models are arguably more understandable. However, Bayesian networks allow answering a broader range of queries — for example, queries where we distinguish between items that the user has viewed and chosen not to purchase and items that the user simply has not viewed (whose variables arguably should be taken to be unobserved).

Heckerman et al. (2000) applied this approach to a range of different collaborative filtering data sets. For example, figure 18.C.1 shows a fragment of a Bayesian network for TV-watching habits learned from Nielsen viewing data. They show that both the Bayesian network and the dependency network methods performed significantly better than previous approaches proposed for this task. The performance of the two methods in terms of predictive accuracy is roughly comparable, and both were fielded successfully as part of Microsoft's E-Commerce software system.

18.5 Bayesian Model Averaging ⋆

18.5.1 Basic Theory

We now reexamine the basic principles of the learning problem. Recall that the Bayesian methodology suggests that we treat unknown parameters as random variables and should consider all possible values when making predictions. When we do not know the structure, the Bayesian methodology suggests that we should consider all possible graph structures. Thus, according to Bayesian theory, given a data set $\mathcal{D} = \{\xi[1], \ldots, \xi[M]\}$ we should make predictions according to the *Bayesian estimation* rule:

Bayesian estimation

$$P(\xi[M+1] \mid \mathcal{D}) = \sum_{\mathcal{G}} P(\xi[M+1] \mid \mathcal{D}, \mathcal{G}) P(\mathcal{G} \mid \mathcal{D}), \tag{18.10}$$

where $P(\mathcal{G} \mid \mathcal{D})$ is posterior probability of different networks given the data

$$P(\mathcal{G} \mid \mathcal{D}) = \frac{P(\mathcal{G}) P(\mathcal{D} \mid \mathcal{G})}{P(\mathcal{D})}.$$

In our discussion so far, we searched for a single structure $\mathcal{G}$ that maximized the Bayesian score, and thus also the posterior probability. When is the focus on a single structure justified?

Recall that the logarithm of the marginal likelihood $\log P(\mathcal{D} \mid \mathcal{G})$ grows linearly with the number of samples. Thus, when M is large, there will be large differences between the top-scoring structure and all the rest. We used this property in the proof of theorem 18.2 to show that for asymptotically large M, the best-scoring structure is the true one. Even when M is not that large, the posterior probability of this particular equivalence class of structures will be exponentially larger than all other structures and dominate most of the mass of the posterior. In such a case, the posterior mass is dominated by this single equivalence class, and we can approximate equation (18.10) with

$$P(\xi[M+1] \mid \mathcal{D}) \approx P(\xi[M+1] \mid \mathcal{D}, \tilde{\mathcal{G}}),$$

where $\tilde{\mathcal{G}} = \arg\max_{\mathcal{G}} P(\mathcal{G} \mid \mathcal{D})$. The intuition is that $P(\tilde{\mathcal{G}} \mid \mathcal{D}) \approx 1$, and $P(\mathcal{G} \mid \mathcal{D}) \approx 0$ for any other structure.

What happens when we consider learning with smaller number of samples? In such a situation, the posterior mass might be distributed among many structures (which may not include the true structure). If we are interested only in density estimation, this might not be a serious problem. If $P(\xi[M+1] \mid \mathcal{D}, \mathcal{G})$ is similar for the different structure with high posterior, then picking one of them will give us reasonable performance. Thus, if we are doing density estimation, then we might get away with learning a single structure and using it. However, we need to remember that the theory suggests that we consider the whole set of structures when making predictions.

structure discovery

confidence estimation

If we are interested in *structure discovery*, we need to be more careful. The fact that several networks have similar scores suggests that one or several of them might be close to the "true" structure. However, we cannot really distinguish between them given the data. If this is the situation, then we should not be satisfied with picking one of these structures (say, even the one with the best score) and drawing conclusions about the domain. Instead, we want to be more cautious and quantify our confidence about the conclusions we make. Such *confidence estimates* are crucial in many domains where we use Bayesian network learning to learn about the structure of processes that generated data. This is particularly true when the available data is limited.

network features

To consider this problem, we need to specify more precisely what would help us understand the posterior over structures. In most cases, we do not want to quantify the posterior explicitly. Moreover, the set of networks with "high" posterior probability is usually large. To deal with this issue, there are various approaches we can take. A fairly general one is to consider *network feature* queries. Such a query can ask what is the probability that an edge, say $X \rightarrow Y$, appears in the "true" network. Another possible query might be about separation in the "true" network — for example, whether *d-sep*$(\boldsymbol{X}; \boldsymbol{Y} \mid \boldsymbol{Z})$ holds in this network. In general, we can formulate such a query as a function $f(\mathcal{G})$. For a binary feature, $f(\mathcal{G})$ can return 1 when the network structure contains the feature, and 0 otherwise. For other features, $f(\mathcal{G})$ may return a numerical quantity that is relevant to the graph structure (such as the length of the shortest trail from X to Y, or the number of nodes whose outdegree is greater than some threshold k).

Given a numerical feature $f(\mathcal{G})$, we can compute its expectation over all possible network structures $\mathcal{G}$:

$$\mathbf{E}_{P(\mathcal{G}|\mathcal{D})}[f(\mathcal{G})] = \sum_{\mathcal{G}} f(\mathcal{G}) P(\mathcal{G} \mid \mathcal{D}). \tag{18.11}$$

In particular, for a binary feature f, this quantity is simply the posterior probability $P(f \mid \mathcal{D})$.

The problem in computing either equation (18.10) or equation (18.11), of course, is that the number of possible structures is superexponential; see exercise 18.20.

We can reduce this number by restricting attention to structures $\mathcal{G}$ where there is a bound d on the number of parents per variable. This assumption, which we will make throughout this section, is a fairly innocuous one. There are few applications in which very large families are called for, and there are rarely enough data to support robust parameter estimation for such families. From a more formal perspective, networks with very large families tend to have low score. Let $\boldsymbol{\mathcal{G}}_d$ be the set of all graphs with indegree bounded by some constant d. Note that the number of structures in $\boldsymbol{\mathcal{G}}_d$ is still superexponential; see exercise 18.20.

Thus, an exhaustive enumeration over the set of possible network structures is feasible only for tiny domains (4–5 variables). In the next sections we consider several approaches to address this problem. In the following discussion, we assume that we are using a Bayesian score that decomposes according to the assumptions we discussed in section 18.3. Recall that the Bayesian score, as defined earlier, is equal to $\log P(\mathcal{D}, \mathcal{G})$. Thus, we have that

$$P(\mathcal{D}, \mathcal{G}) = \prod_i \exp\{\text{FamScore}_B(X_i \mid \text{Pa}_{X_i}^{\mathcal{G}} \ : \ \mathcal{D})\}. \tag{18.12}$$

18.5.2 Model Averaging Given an Order

In this section, we temporarily turn our attention to a somewhat easier problem. Rather than perform model averaging over the space of *all* structures, we restrict attention to structures that are consistent with some predetermined *variable ordering* $\prec$. As in section 18.4.2, we restrict attention to structures $\mathcal{G}$ such that there is an edge $X_i \rightarrow X_j$, only if $X_i \prec X_j$.

variable ordering

18.5.2.1 Computing the marginal likelihood

We first consider the problem of computing the *marginal likelihood* of the data given the order:

marginal likelihood

$$P(\mathcal{D} \mid \prec) = \sum_{\mathcal{G} \in \boldsymbol{\mathcal{G}}_d} P(\mathcal{G} \mid \prec) P(\mathcal{D} \mid \mathcal{G}). \tag{18.13}$$

Note that this summation, although restricted to networks with bounded indegree and consistent with $\prec$, is still exponentially large; see exercise 18.20.

Before we compute the marginal likelihood, we note that computing the marginal likelihood is equivalent to making predictions with equation (18.10). To see this, we can use the definition of probability to see that

$$P(\xi[M+1] \mid \mathcal{D}, \prec) = \frac{P(\xi[M+1], \mathcal{D} \mid \prec)}{P(\mathcal{D} \mid \prec)}.$$

Now both terms on the right are marginal likelihood terms, one for the original data and the other for original data extended by the new instance.

We now return to the computation of the marginal likelihood. The key insight is that when we restrict attention to structures consistent with a given order $\prec$, the choice of family for one variable places no additional constraints on the choice of family for another. Note that this

property does not hold without the restriction on the order; for example, if we pick X_i to be a parent of X_j, then X_j cannot in turn be a parent of X_i.

Therefore, we can choose a structure $\mathcal{G}$ consistent with $\prec$ by choosing, independently, a family $\boldsymbol{U}_i$ for each variable X_i. Global parameter modularity assumption states that the choice of parameters for the family of X_i is independent of the choice of family for another family in the network. Hence, summing over possible graphs consistent with $\prec$ is equivalent to summing over possible choices of family for each variable, each with its parameter prior. Given our constraint on the size of the family, the possible parent sets for the variable X_i are

$$\mathcal{U}_{i,\prec} = \{\boldsymbol{U} \ : \ \boldsymbol{U} \prec X_i, |\boldsymbol{U}| \leq d\},$$

where $\boldsymbol{U} \prec X_i$ is defined to hold when all variables in $\boldsymbol{U}$ precede X_i in $\prec$. Let $\boldsymbol{\mathcal{G}}_{d,\prec}$ be the set of structures in $\boldsymbol{\mathcal{G}}_d$ consistent with $\prec$. Using equation (18.12), we have that

$$\begin{aligned} P(\mathcal{D} \mid \prec) &= \sum_{\mathcal{G} \in \boldsymbol{\mathcal{G}}_{d,\prec}} \prod_i \exp\{\text{FamScore}_B(X_i \mid \text{Pa}^{\mathcal{G}}_{X_i} \ : \ \mathcal{D})\} \\ &= \prod_i \sum_{\boldsymbol{U}_i \in \mathcal{U}_{i,\prec}} \exp\{\text{FamScore}_B(X_i \mid \boldsymbol{U}_i \ : \ \mathcal{D})\}. \end{aligned} \tag{18.14}$$

Intuitively, the equality states that we can sum over all network structures consistent with $\prec$ by summing over the set of possible families for each variable, and then multiplying the results for the different variables. This transformation allows us to compute $P(\mathcal{D} \mid \prec)$ efficiently. The expression on the right-hand side consists of a product with a term for each variable X_i, each of which is a summation over all possible families for X_i. Given a bound d over the number of parents, the number of possible families for a variable X_i is at most $\binom{n}{d} \leq n^d$. Hence, the total cost of computing equation (18.14) is at most $n \cdot n^d = n^{d+1}$.

18.5.2.2 Probabilities of features

For certain types of features f, we can use the technique of the previous section to compute, in closed form, the probability $P(f \mid \prec, \mathcal{D})$ that f holds in a structure given the order and the data.

In general, if f is a feature. We want to compute

$$P(f \mid \prec, \mathcal{D}) = \frac{P(f, \mathcal{D} \mid \prec)}{P(\mathcal{D} \mid \prec)}.$$

We have just shown how to compute the denominator. The numerator is a sum over all structures that contain the feature and are consistent with the order:

$$P(f, \mathcal{D} \mid \prec) = \sum_{\mathcal{G} \in \boldsymbol{\mathcal{G}}_{d,\prec}} f(\mathcal{G}) P(\mathcal{G} \mid \prec) P(\mathcal{D} \mid \mathcal{G}). \tag{18.15}$$

The computation of this term depends on the specific type of feature f.

The simplest situation is when we want to compute the posterior probability of a particular choice of parents $\boldsymbol{U}$. This in effect requires us to sum over all graphs where $\text{Pa}^{\mathcal{G}}_{X_i} = \boldsymbol{U}$. In this case, we can apply the same closed-form analysis to (18.15). The only difference is that we restrict $\mathcal{U}_{i,\prec}$ to be the singleton $\{\boldsymbol{U}\}$. Since the terms that sum over the parents of X_j for $i \neq j$ are not disturbed by this constraint, they cancel out from the equation.

Proposition 18.7

$$P(\mathrm{Pa}^{\mathcal{G}}_{X_i} = \boldsymbol{U} \mid D, \prec) = \frac{\exp\{\mathrm{FamScore}_B(X_i \mid \boldsymbol{U} : \mathcal{D})\}}{\sum_{\boldsymbol{U}' \in \mathcal{U}_{i,\prec}} \exp\{\mathrm{FamScore}_B(X_i \mid \boldsymbol{U}' : \mathcal{D})\}}.$$

A slightly more complex situation is when we want to compute the posterior probability of the *edge feature* $X_i \rightarrow X_j$. Again, we can apply the same closed-form analysis to (18.15). The only difference is that we restrict $\mathcal{U}_{j,\prec}$ to consist only of subsets that contain X_i.

Proposition 18.8

$$P(X_j \in \mathrm{Pa}^{\mathcal{G}}_{X_i} \mid \prec, \mathcal{D}) = \frac{\sum_{\{\boldsymbol{U} \in \mathcal{U}_{i,\prec} : X_j \in \boldsymbol{U}\}} \exp\{\mathrm{FamScore}_B(X_i \mid \boldsymbol{U} : \mathcal{D})\}}{\sum_{\boldsymbol{U} \in \mathcal{U}_{i,\prec}} \exp\{\mathrm{FamScore}_B(X_i \mid \boldsymbol{U} : \mathcal{D})\}}.$$

Unfortunately, this approach cannot be used to compute the probability of arbitrary structural features. For example, we cannot compute the probability that there exists some directed path from X_i to X_j, as we would have to consider all possible ways in which a path from X_i to X_j could manifest itself through exponentially many structures.

We can overcome this difficulty using a simple sampling approach. Proposition 18.7 provides us with a closed-form expression for the exact posterior probability of the different possible families of the variable X_i. We can therefore easily sample entire networks from the posterior distribution given the order: we simply sample a family for each variable, according to the distribution specified in proposition 18.7. We can then use the sampled networks to evaluate any feature, such as X_i is ancestor of X_j.

18.5.3 The General Case

In the previous section, we made the simplifying assumption that we were given a predetermined order. Although this assumption might be reasonable in certain cases, it is clearly too restrictive in domains where we have very little prior knowledge. We therefore want to consider structures consistent with all possible orders. Here, unfortunately, we have no elegant tricks that allow an efficient, closed-form solution. As with search problems we discussed, the choices of parents for one variable can interfere with the choices of parents for another.

A general approach is try to approximate the exhaustive summation over structures in quantities of interest (that is, equation (18.10) and equation (18.11)) with approximate sums. For this purpose we utilize ideas that are similar to the ones we discuss in chapter 12 in the context of approximate inference.

A first-cut approach for approximating the sums in our case is to find a set $\boldsymbol{\mathcal{G}}'$ of high scoring structures, and then estimate the relative mass of the structures in $\boldsymbol{\mathcal{G}}'$. Thus, for example, we would approximate equation (18.11) with

$$P(f \mid \mathcal{D}) \approx \frac{\sum_{\mathcal{G} \in \boldsymbol{\mathcal{G}}'} P(\mathcal{G} \mid \mathcal{D}) f(\mathcal{G})}{\sum_{\mathcal{G} \in \boldsymbol{\mathcal{G}}'} P(\mathcal{G} \mid \mathcal{D})}. \tag{18.16}$$

This approach leaves open the question of how we construct $\boldsymbol{\mathcal{G}}'$. The simplest approach is to use model selection to pick a single high-scoring structure and then use that as our approximation. If the amount of data is large relative to the size of the model, then the posterior will be sharply peaked around a single model, and this approximation is a reasonable one. However, as we discussed, when the amount of data is small relative to the size of the model, there is usually a large number of high-scoring models, so that using a single model as our set $\boldsymbol{\mathcal{G}}'$ is a very poor approximation.

We can find a larger set of structures by recording all the structures examined during the search and returning the high-scoring ones. However, the set of structures found in this manner is quite sensitive to the search procedure we use. For example, if we use greedy hill climbing, then the set of structures we collect will all be quite similar. Such a restricted set of candidates also shows up when we consider multiple restarts of greedy hill climbing. This is a serious problem, since we run the risk of getting estimates of confidence that are based on a biased sample of structures.

18.5.3.1 MCMC Over Structures

An alternative approach is based on the use of sampling. As in chapter 12, we aim to approximate the expectation over graph structures in equation (18.10) and equation (18.16) by an empirical average. Thus, if we manage to sample graphs $\mathcal{G}_1, \ldots, \mathcal{G}_K$ from $P(\mathcal{G} \mid \mathcal{D})$, we can approximate equation (18.16) as

$$P(f \mid \mathcal{D}) \approx \frac{1}{K} \sum_k f(\mathcal{G}_k).$$

The question is how to sample from the posterior distribution. One possible answer is to use the general tool of Markov chain Monte Carlo (MCMC) simulation; see section 12.3. In this case, we define a Markov chain over the space of possible structures whose stationary distribution is the posterior distribution $P(G \mid \mathcal{D})$. We then generate a set of possible structures by doing a random walk in this Markov chain. Assuming that we continue this process until the chain converges to the stationary distribution, we can hope to get a set of structures that is representative of the posterior.

How do we construct a Markov chain over the space of structures? The idea is a fairly straightforward application of the principles discussed in section 12.3. The states of the Markov chain are graphs in the set $\boldsymbol{\mathcal{G}}$ of graphs we want to consider. We consider local operations (for example, add edge, delete edge, reverse edge) that transform one structure to another. We assume we have a proposal distribution $\mathcal{T}^Q$ over such operations. We then apply the Metropolis-Hastings acceptance rule. Suppose that the current state is $\mathcal{G}$, and we sample the transition to $\mathcal{G}'$ from the proposal distribution, then we accept this transition with probability

$$\min\left[1, \frac{P(\mathcal{G}', \mathcal{D})\mathcal{T}^Q(\mathcal{G}' \to \mathcal{G})}{P(\mathcal{G}, \mathcal{D})\mathcal{T}^Q(\mathcal{G} \to \mathcal{G}')}\right].$$

As discussed in section 12.3, this strategy ensures that we satisfy the detailed balance condition. To ensure that we have a regular Markov chain, we also need to verify that the space $\boldsymbol{\mathcal{G}}$ is connected, that is, that we can reach each structure in $\boldsymbol{\mathcal{G}}$ from any other structure in $\boldsymbol{\mathcal{G}}$ through a sequence of operations. This is usually easy to ensure with the set of operators we discussed.

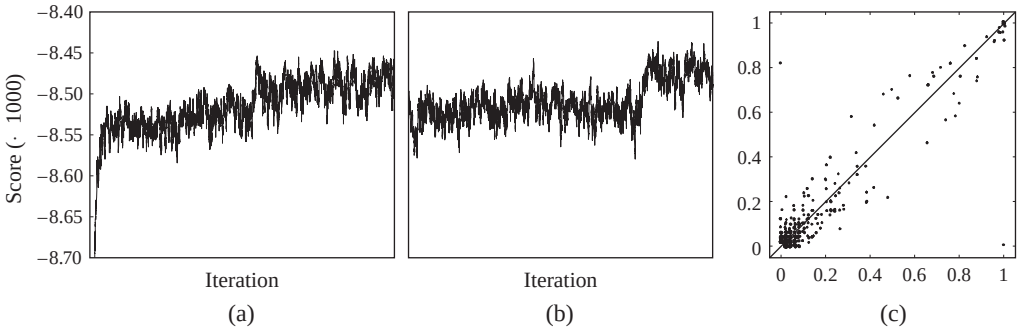


Figure 18.8 MCMC structure search using 500 instances from ICU-Alarm **network.** (a) & (b) Plots of the progression of the MCMC process for two runs for 500 instances sampled from the ICU-Alarm network. The x-axis denotes the iteration number, and the y-axis denotes the score of the current structure. (a) A run initialized from the empty structure, and (b) a run initialized from the structure found by structure search. (c) Comparison of the estimates of the posterior probability of edges using 100 networks sampled every 2,000 iterations from the each of the runs (after initial burn-in of 20,000 iterations). Each point denotes an edge, the x-coordinate is the estimate using networks from run (a) and the y-coordinate is the estimate using networks from run (b).

It is important to stress that if the operations we apply are local (such as edge addition, deletion, and reversal), then we can efficiently compute the ratio $\frac{P(\mathcal{G}',\mathcal{D})}{P(\mathcal{G},\mathcal{D})}$. To see this, note that the logarithm of this ratio is the difference in score between the two graphs. As we discussed in section 18.4.3.3, this difference involves only terms that relate to the families of the variables that are involved in the local move. Thus, performing MCMC over structures can use the same caching schemes discussed in section 18.4.3.3, and thus it can be executed efficiently.

The final detail we need to consider is the choice the proposal distribution $\mathcal{T}^Q$. Many choices are reasonable. The simplest one to use, and one that is often used in practice, is the uniform distribution over all possible operations (excluding ones that violate acyclicity or other constraints we want to impose).

To demonstrate the use of MCMC over structures, we sampled 500 instances from the ICU-Alarm network. This is a fairly small training set, and so we do not hope to recover one network. Instead, we run the MCMC sampling to collect 100 networks and use these to estimate the posterior of different edges. One of the hard problems in using such an approach is checking whether the MCMC simulation converged to the stationary distribution. One ad-hoc test we can perform is to compare the results of running several independent simulations. For example, in figure 18.8a and 18.8b we plot the progression of two runs, one starting from the empty structure and the other from the structure found by structure search on the same data set. We see that initially the two runs are very different; they sample networks with rather different scores. In later stages of the simulation, however, the two runs are in roughly the same range of scores. This suggests that they might be exploring the same region. Another way of testing this is to compare the estimate we computed based on each of the two runs. As we see in figure 18.8c,

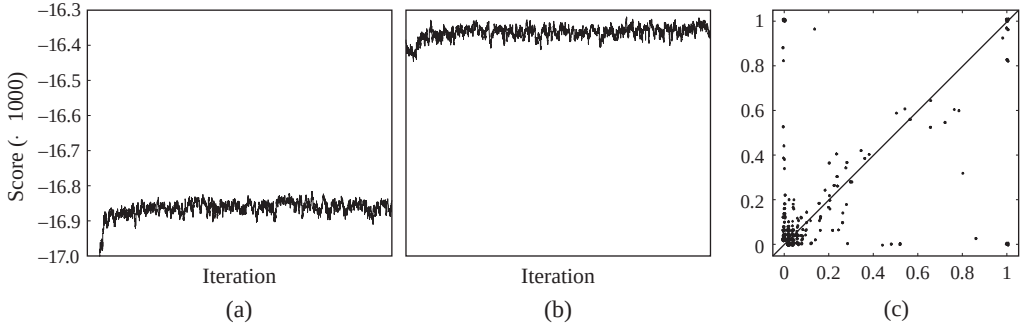


Figure 18.9 MCMC structure search using 1,000 instances from ICU-Alarm **network.** The protocol is the same as in figure 18.8. (a) & (b) Plots two MCMC runs. (c) A comparison of the estimates of the posterior probability of edges.

although the estimates are definitely not identical, they are mostly in agreement with each other.

Variants of MCMC simulation have been applied successfully to this task for a variety of small domains, typically with 4–14 variables. However, there are several issues that potentially limit its effectiveness for large domains involving many variables. As we discussed, the space of network structures grows superexponentially with the number of variables. Therefore, the domain of the MCMC traversal is enormous for all but the tiniest domains. More importantly, the posterior distribution over structures is often quite peaked, with neighboring structures having very different scores. The reason is that even small perturbations to the structure — a removal of a single edge — can cause a huge reduction in score. Thus, the "posterior landscape" can be quite jagged, with high "peaks" separated by low "valleys." In such situations, MCMC is known to be slow to mix, requiring many samples to reach the posterior distribution.

To see this effect, consider figure 18.9, where we repeated the same experiment we performed before, but this time for a data set of 1000 instances. As we can see, although we considered a large number of iterations, different MCMC runs converge to quite different ranges of scores. This suggests that the different runs are sampling from different regions of the search space. Indeed, when we compare estimates in figure 18.9c, we see that some edges have estimated posterior of almost 1 in one run and of 0 in another. We conclude that each of the runs became stuck in a local "hill" of the search space and explored networks only in that region.

18.5.3.2 MCMC Over Orders

collapsed MCMC

To avoid some of the problems with MCMC over structures, we consider an approach that utilizes *collapsed particles*, as described in section 12.4. Recall that a collapsed particle consists of two components, one an assignment to a set of random variables that we sampled, and the other a distribution over the remaining variables.

In our case, we utilize the notion of collapsed particle as follows. Instead of working in the space of graphs $\mathcal{G}$, we work in the space of pairs $\langle \prec, \mathcal{G} \rangle$ so that $\mathcal{G}$ is consistent with the ordering $\prec$. As we have seen, we can use closed-form equations to deal the distribution over $\mathcal{G}$ given an

ordering $\prec$. Thus, each ordering can represent a collapsed particle.

We now construct a Markov chain over the state of all $n!$ orders. Our construction will guarantee that this chain has the stationary distribution $P(\prec | \mathcal{D})$. We can then simulate this Markov chain, obtaining a sequence of samples $\prec_1, \ldots, \prec_T$. We can now approximate the expected value of any function f as

$$P(f \mid \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} P(f \mid \mathcal{D}, \prec_t),$$

where we compute $P(f \mid \prec_t, \mathcal{D})$ as described in section 18.5.2.2.

Metropolis-Hastings

It remains only to discuss the construction of the Markov chain. Again, we use a standard *Metropolis-Hastings* algorithm. For each order $\prec$, we define a *proposal probability* $\mathcal{T}^Q(\prec \to \prec')$, which defines the probability that the algorithm will "propose" a move from $\prec$ to $\prec'$. The algorithm then *accepts* this move with probability

$$\min \left[1, \frac{P(\prec', \mathcal{D}) \mathcal{T}^Q(\prec' \to \prec)}{P(\prec, \mathcal{D}) \mathcal{T}^Q(\prec \to \prec')} . \right]$$

As we discussed in section 12.3, this suffices to ensure the detailed balance condition, and thus the resulting chain is reversible and has the desired stationary distribution.

We can consider several specific constructions for the proposal distribution, based on different neighborhoods in the space of orders. In one very simple construction, we consider only operators that flip two variables in the order (leaving all others unchanged):

$$(X_{i_1} \ldots X_{i_j} \ldots X_{i_d} \ldots X_{i_n}) \mapsto (X_{i_1} \ldots X_{i_d} \ldots X_{i_j} \ldots X_{i_n}).$$

Clearly, such operators allow to get from any ordering to another in relatively few moves.

We note that, again, we can use decomposition to avoid repeated computation during the evaluation of candidate operators. Let $\prec$ be an order and let $\prec'$ be the order obtained by flipping X_{i_j} and X_{i_k}. Now, consider the terms in equation (18.14); those terms corresponding to variables X_{i_ℓ} in the order $\prec$ that precede X_{i_j} or succeed X_{i_k} do not change, since the set of potential parent sets $\mathcal{U}_{i_l, \prec}$ is the same.

Performing MCMC on the space of orderings is much more expensive than MCMC on the space of networks. Each proposed move requires performing summation over a fairly large set of possible parents for each variable. On the other hand, since each ordering corresponds to a large number of networks, a few moves in the space of orderings correspond to a much larger number of moves in the space of networks.

Empirical results show that using MCMC over orders alleviate some of the problems we discussed with MCMC over network structures. For example, figure 18.10 shows two runs of this variant of MCMC for the same data set as in figure 18.9. Although these two runs involve much fewer iterations, they quickly converge to the same "area" of the search space and agree on the estimation of the posterior. This is an empirical indication that these are reasonably close to converging on the posterior.

18.6 Learning Models with Additional Structure

So far, our discussion of structure learning has defined the task as one of finding the best graph structure $\mathcal{G}$, where a graph structure is simply a specification of the parents to each of the

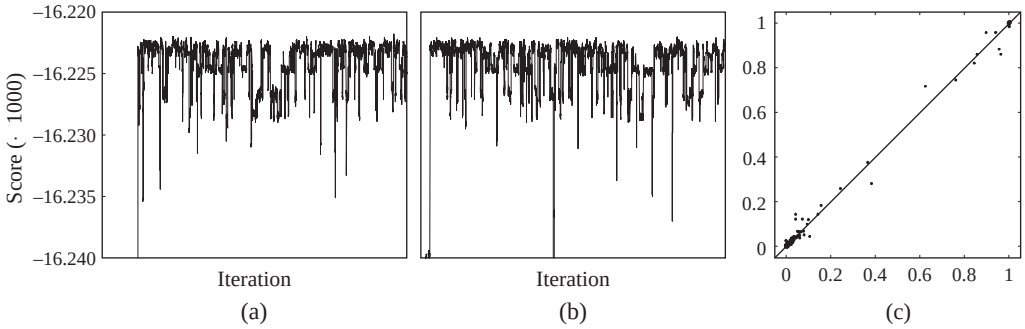


Figure 18.10 MCMC order search using 1,000 instances from ICU-Alarm **network.** The protocol is the same as in figure 18.8. (a) & (b) Plots two MCMC runs. (c) A comparison of the estimates of the posterior probability of edges.

random variables in the network. However, some classes of models have additional forms of structure that we also need to identify. For example, when learning networks with structured CPDs, we may need to select the structure for the CPDs. And when learning template-based models, our model is not even a graph over $\mathcal{X}$, but rather a set of dependencies over a set of template attributes.

Although quite different, the approach in both of these settings is analogous to the one we used for learning a simple graph structure: we define a hypothesis space of potential models and a scoring function that allows us to evaluate different models. We then devise a search procedure that attempts to find a high-scoring model. Even the choice of scoring functions is essentially the same: we generally use either a penalized likelihood (such as the BIC score) or a Bayesian score based on the marginal likelihood. Both of these scoring functions can be computed using the likelihood function for models with shared parameters that we developed in section 17.5. As we now discuss, the key difference in this new setting is the structure of the search space, and thereby of the search procedure. In particular, our new settings require that we make decisions in a space that is not the standard one of deciding on the set of parents for a variable in the network.

18.6.1 Learning with Local Structure

As we discussed, we can often get better performance in learning if we use more compact models of CPDs rather than ones that require a full table-based parameterization. More compact models decrease the number of parameters and thereby reduce overfitting. Some of the techniques that we described earlier in this chapter are applicable to various forms of structured CPDs, including table-CPDs, noisy-or CPDs, and linear Gaussian CPDs (although the closed-form Bayesian score is not applicable to some of those representations). In this section, we discuss the problem of learning CPDs that explicitly encode local parameter sharing within a CPD, focusing on CPD trees as a prototypical (and much-studied) example. Here, we need to make decisions about the local structure of the CPDs as well as about the network structure. Thus, our search space is

now much larger: it consists both of "global" decisions — the assignment of parents for each variable — and of "local" decisions — the structure of the CPD tree for each variable.

18.6.1.1 Scoring Networks

We first consider how the development of the score of a network changes when we consider tree-CPDs instead of table-CPDs. Assume that we are given a network structure $\mathcal{G}$; moreover, for each variable X_i, we are given a description of the CPD tree $\mathcal{T}_i$ for $P(X_i \mid \mathrm{Pa}_{X_i}^{\mathcal{G}})$. We view the choice of the trees as part of the specification of the model. Thus, we consider the score of $\mathcal{G}$ with $\mathcal{T}_1, \ldots, \mathcal{T}_n$

$$\mathrm{score}_B(\mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n \ : \ \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n) + \log P(\mathcal{G}) + \sum_i \log P(\mathcal{T}_i \mid \mathcal{G}),$$

where $P(\mathcal{D} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ is the marginal likelihood when we integrate out the parameters in all the CPDs, and $P(\mathcal{T}_i \mid \mathcal{G})$ is the prior probability over the tree structure.

structure prior

We usually assume that the *structure prior* over trees does not depend on the graph, but only on the choice of parents. Possible priors include the uniform prior

$$P(\mathcal{T}_i \mid \mathcal{G}) \propto 1$$

and a prior that penalizes larger trees

$$P(\mathcal{T}_i \mid \mathcal{G}) \propto c^{|\mathcal{T}_i|}$$

(for some constant $c < 1$).

We now turn to the marginal likelihood term. As in our previous discussion, we assume global and local parameter independence. This means that we have an independent prior over the parameters in each leaf of each tree. For each X_i and each assignment pa_{X_i} to X_i's parents, let $\lambda(\mathrm{pa}_{X_i})$ be the leaf in $\mathcal{T}_i$ to which pa_{X_i} is assigned.

Using an argument that parallels our development of proposition 18.2, we have:

Proposition 18.9 *Let $\mathcal{G}$ be a network structure, $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be CPD trees, and $P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ be a parameter prior satisfying global and local parameter independence. Then,*

$$P(\mathcal{D} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n) = \prod_i \prod_{\ell \in \mathrm{Leaves}(\mathcal{T}_i)} \int \prod_{m \ : \ \lambda(\mathrm{pa}_{X_i}[m]) = \ell} P(X_i[m] \mid \ell, \boldsymbol{\theta}_{X_i|\ell}, \mathcal{G}, \mathcal{T}_i) P(\boldsymbol{\theta}_{X_i|\ell} \mid \mathcal{G}, \mathcal{T}_i) d\boldsymbol{\theta}_{X_i|\ell}.$$

Thus, the score over a tree-CPD decomposes according to the structure of each of the trees. Each of the terms that corresponds to a leaf is the marginal likelihood of a single variable distribution and can be solved using standard techniques. Usually, we use Dirichlet priors at the leaves, and so the term for each leaf will be a term of the form of equation (18.9).

Similar to the developments in table-CPDs, we can extend the notion of score decomposability to networks with tree-CPDs. Recall that score decomposability implies that identical substructures receive the same score (regardless of structure of other parts of the network). Suppose we have two possible tree-CPDs for X (not necessarily with the same set of parents in each), and suppose that there are two leaves ℓ_1 and ℓ_2 in the two trees such that $\boldsymbol{c}_{\ell_1} = \boldsymbol{c}_{\ell_2}$; that is, the

same conditions on the parents of X in each of the two CPDs lead to the respective leaves. Thus, the two leaves represent a similar situation (even though the tree structures can differ elsewhere), and intuitively they should receive the same score.

To ensure this property, we need to extend the notion of parameter modularity:

Definition 18.6 tree parameter modularity

Let $\{P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n)\}$ be a set of parameter priors over networks with tree-CPDs that satisfy global and local parameter independence. The prior satisfies tree parameter modularity *if for each $\mathcal{G}, \mathcal{T}_i$ and $\mathcal{G}', \mathcal{T}_i'$ and $\ell \in \text{Leaves}(\mathcal{T}_i)$, $\ell' \in \text{Leaves}(\mathcal{T}_i')$ such that $\boldsymbol{c}_\ell = \boldsymbol{c}_{\ell'}$, then $P(\boldsymbol{\theta}_{X_i|\ell} \mid \mathcal{G}, \mathcal{T}_i) = P(\boldsymbol{\theta}_{X_i|\ell'} \mid \mathcal{G}', \mathcal{T}_i')$.* ■

BDe prior

A natural extension of the *BDe prior* satisfies this condition. Suppose we choose a prior distribution P' and an equivalent sample size α; then we choose hyperparameters for $P(\boldsymbol{\theta}_{X_i|\ell} \mid \mathcal{G}, \mathcal{T}_i)$ to be $\alpha_{x_i|\ell} = \alpha \cdot P'(x_i, \boldsymbol{c}_\ell)$. Using this assumption, we can now decompose the Bayesian score for a tree-CPD; see exercise 18.17.

18.6.1.2 Search with Local Structure

Our next task is to describe how to search our space of possible hypotheses for one that has high score. Recall that we now have a much richer hypothesis space over which we need to search.

The key question in the case of learning tree-CPDs is that of choosing a tree structure for $P(X \mid \boldsymbol{U})$ for some set of possible parents $\boldsymbol{U}$. (We will discuss different schemes for choosing $\boldsymbol{U}$.) There are two natural operators in this space.

- Split – replace a leaf in the tree by an internal variable that leads to a leaf. This step increases the tree height by 1 on a selected branch.
- Prune – replace an internal variable by a leaf.

Starting from the "empty" tree (that consists of single leaf), we can reach any other tree by a sequence of split operations.

The prune operations allow searches to retract some of the moves. This capability is critical, since there are many local maxima in growing trees. For example, it is often the case that a sequence of two splits leads to a high-scoring tree, but the intermediate step (after performing only the first split) has low score. Thus, greedy hill-climbing search can get stuck in a local maximum early on. As a consequence, the search algorithm for tree-CPDs is often not a straightforward hill climbing. In particular, one common strategy is to choose the best-scoring split at each point, even if that split actually decreases the score. Once we have finished constructing the tree, we go back and evaluate earlier splitting decisions that we made.

The search space over trees can explored using a "divide and conquer" approach. Once we decide to split our tree on one variable, say Y, the choices we make in one subtree, say the one corresponding to $Y = y^0$, are independent of the choices we make in other subtrees. Thus, we can explore for the best structure for each one of the subsets independently of the other. Thus, we can devise recursive search procedures for trees, which works roughly as follows. The procedure receives a set of instances to learn from, a target variable X, and a set $\boldsymbol{U}$ of possible parents. It evaluates each $Y \in \boldsymbol{U}$ as a potential question by scoring the tree-CPD that has Y as a root and has no further question. This *myopic* evaluation can miss pairs or longer sequences

of questions that lead to high accuracy predictions and hence to a high scoring model. However, it can be performed very efficiently (see exercise 18.17), and so it is often used, under the hope that other approaches (such as random restarts or tabu search) can help us deal with local optima.

After choosing the root, the procedure divides the data set into smaller data sets, each one corresponding to one value of the variable Y. Using the decomposition property we just discussed, the procedure is recursively invoked to learn a subtree in each D_y. These subtrees are put together into a tree-CPD that has Y as a root. The final test compares the score of the constructed subtree to that of the empty subtree. This test is necessary, since our construction algorithm selected the best split at this point, but without checking that this split actually improves the score. This strategy helps avoid local maxima arising from the myopic nature of the search, but it does potentially lead to unnecessary splits that actually hurt our score. This final check helps avoid those. In effect, this procedure evaluates, as it climbs back up the tree in the recursion, all of the possible merge steps relative to our constructed tree.

There are two standard strategies for applying this procedure. One is an *encapsulated search*. Here, we perform one of the network-structure search procedures we described in earlier sections of this chapter. Whenever we perform a search operation (such as adding or removing an edge), we use a local procedure to find a representation for the newly created CPD. The second option is to use a *unified search*, where we do not distinguish between operators that modify the network and operators that modify the local structure. Instead, we simply apply a local search that modifies the joint representation of the network and local structure for each CPD. Here, each state in the search space consists of a collection of n trees $\langle \mathcal{T}_1, \ldots, \mathcal{T}_n \rangle$, which define both the structure and the parameters of the network. We can structure the operations in the search in many ways: we can update an entire CPD tree for one of the variables, or we can evaluate the delta-score of the various split and merge operations in tree-CPDs all across the network and choose the best one. In either case, we must remember that not every collection of trees defines an acyclic network structure, and so we must construct our search carefully to ensure acyclic structures, as well as any other constraints that we want to enforce (such as bounded indegree).

Each of these two options for learning with local structure has its benefits and drawbacks. Encapsulated search spaces decouple the problem of network structure learning from that of learning CPD structures. This modularity allows us to "plug in" any CPD structure learning procedure within generic search procedure for network learning. Moreover, since we consider the structure of each CPD as a separate problem, we can exploit additional structure in the CPD representation. A shortcoming of encapsulated search spaces is that they can easily cause us to repeat a lot of effort. In particular, we redo the local structure search for X's CPD every time we consider a new parent for X. This new parent is often irrelevant, and so we end up discarding the local structure we just learned. Unified search spaces alleviate this problem. In this search formulation, adding a new parent Y to X is coupled with a proposal as to the specific position in the tree-CPD of X where Y is used. This flexibility comes at a cost: the number of possible operators at a state in the search is very large — we can add a split on any variable at each leaf of the n trees. Moreover, key operations such as edge reversal or even edge deletion can require many steps in the finer-grained search space, and therefore they are susceptible to forming local optima. Overall, there is no clear winner between these two methods, and the best choice is likely to be application-specific.

18.6.2 Learning Template Models

We now briefly discuss the task of structure learning for template-based models. Here, once we define the hypothesis space, the solution becomes straightforward. Importantly, when learning template-based models, our hypothesis space is the space of structures in the template representation, not in the ground network from which we are learning.

For DBNs, we are learning a representation as in definition 6.4: a time 0 network $\mathcal{B}_0$, and a transition network $\mathcal{B}_{\rightarrow}$. Importantly, the latter network is specified in terms of template attributes; thus, the learned structure will be used for all time slices in the unrolled DBN. The learned structure must satisfy the constraints on our template-based representation. For example, $\mathcal{B}_{\rightarrow}$ must be a valid conditional Bayesian network for $\mathcal{X}'$ given $\mathcal{X}$: it must be acyclic, and there must be no incoming edges into any variables in $\mathcal{X}$.

For object-relational models, each hypothesis in our space specifies an assignment Pa_A for every $A \in \aleph$. Here also, the learned structure is at the template level and is applied to all instantiations of A in the ground network. The learned structure must satisfy the constraints of the template-based language with which we are dealing. For example, in a plate model, the structure must satisfy the constraints of definition 6.9, for example, the fact that for any template parent $B_i(\boldsymbol{U}_i) \in \mathrm{Pa}_A$, we have that the parent's arguments $\boldsymbol{U}_i$ must be a subset of the attribute's argument signature $\alpha(A)$.

Importantly, in both plate models and DBNs, the set of possible structures is finite. This fact is obvious for DBNs. For plate models, note that the set of template attributes is finite; the possible argument signatures of the parents is also finite, owing to the constraints that $\boldsymbol{U}_i \subseteq \alpha(A)$. However, this constraint does not necessarily hold for richer languages. We will return to this point.

Given a hypothesis space, we need to determine a scoring function and a search algorithm. Based on our analysis in section 17.5.1.2 and section 17.5.3, we can easily generalize any of our standard scoring functions (say the BIC score or the marginal likelihood) to the case of template-based representations. The analysis in section 17.5.1.2 provides us with a formula for a decomposed likelihood function, with terms corresponding to each of the model parameters at the template level. From this formula, the derivation of the BIC score is immediate. We use the decomposed likelihood function and penalize with the number of parameters *in the template model*. For a Bayesian score, we can follow the lines of section 17.5.3 and assume global parameter independence at the template level. The posterior now decomposes in the same way as the likelihood, giving us, once again, a decomposable score.

With a decomposable score, we can now search over the set of legal structures in our hypothesis space, as defined earlier. In this case, the operators that we typically apply are the natural variants of those used in standard structure search: edge addition, deletion, and (possibly) reversal. The only slight subtlety is that we must check, at each step, that the model resulting from the operator satisfies the constraints of our template-based representation. In particular, in many cases for both DBNs and PRMs, edge reversal will lead to an illegal structure. For example, in DBNs, we cannot reverse an edge $A \rightarrow A'$. In plate models, we cannot reverse an edge $B(U) \rightarrow A(U, U')$. Indeed, the notion of edge reversal only makes sense when $\alpha(A) = \alpha(B)$. Thus, a more natural search space for plate models (and other object-relational template models) is one that simply searches for a parent set for each $A \in \aleph$, using operators such as parent addition and deletion.

We conclude this discussion by commenting briefly on the problem of structure learning for more expressive object-relational languages. Here, the complexity of our hypothesis space can be significantly greater. Consider, for example, a PRM. Most obviously, we see that the specification here involves not only possible parents but also possibly a guard and an aggregation function. Thus, our model specification requires many more components. Subtler, but also much more important, is the fact that the space of possible structures is potentially infinite. The key issue here is that the parent signature $\alpha(\mathrm{Pa}_A)$ can be a superset of $\alpha(A)$, and therefore the set of possible parents we can define is unboundedly large.

Example 18.2

Returning to the Genetics *domain, we can, if we choose, allow the genotype of a person depending on the genotype of his or her mother, or of his or her grandmother, or of his or her paternal uncles, or on any type of relative arbitrarily far away in the family tree (as long as acyclicity is maintained). For example, the dependence on paternal uncle (not by marriage) can be written as a dependence of* Genotype(U) *on* Genotype(U') *with the guard*

$$\textit{Father}(V, U) \wedge \textit{Brother}(U', V),$$

assuming Brother has already been defined. ■

In general, by increasing the number of logical variables in the attribute's parent argument signature, we can introduce dependencies over objects that are arbitrarily far away. Thus, when learning PRMs, we generally want to introduce a structure prior that penalizes models that are more complex, for example, as an exponential penalty on the number of logical variables in $\alpha(\mathrm{Pa}_A) - \alpha(A)$, or on the number of clauses in the guard.

In summary, **the key aspect to learning structure of template-based models is that both the structure and the parameters are specified at the template level, and therefore that is the space over which our structure search is conducted. It is this property that allows us to learn a model from one skeleton (for example, one pedigree, or one university) and apply that same model to a very different skeleton.**

18.7 Summary and Discussion

In this chapter, we considered the problem of structure learning from data. As we have seen, there are two main issues that we need to deal with: the statistical principles that guide the choice between network structures, and the computational problem of applying these principles.

The statistical problem is easy to state. Not all dependencies we can see in the data are real. Some of them are artifacts of the finite sample we have at our disposal. Thus, to learn (that is, generalize to new examples), we must apply caution in deciding which dependencies to model in the learned network.

We discussed two approaches to address this problem. The first is the constraint-based approach. This approach performs statistical tests of independence to collect a set of dependencies that are strongly supported by the data. Then it searches for the network structure that "explains" these dependencies and no other dependencies. The second is the score-based approach. This approach scores whole network structures against the data and searches for a network structure that maximize the score.

What is the difference between these two approaches? Although at the outset they seem quite different, there are some similarities. In particular, if we consider a choice between the two

possible networks over two variables, then both approaches use a similar decision rule to make the choice; see exercise 18.27.

When we consider more than two variables, the comparison is less direct. **At some level, we can view the score-based approach as performing a test that is similar to a hypothesis test. However, instead of testing each pair of variables locally, it evaluates a function that is somewhat like testing the complete network structure against the null hypothesis of the empty network. Thus, the score-based approach takes a more global perspective, which allows it to trade off approximations in different part of the network.**

The second issue we considered was the computational issue. Here there are clear differences. In the constraint-based approach, once we collect the independence tests, the construction of the network is an efficient (low-order polynomial) procedure. On the other hand, we saw that the optimization problem in the score-based approach is NP-hard. Thus, we discussed various approaches for heuristic search.

When discussing this computation issue, one has to remember how to interpret the theoretical results. In particular, the NP-hardness of score-based optimization does not mean that the problem is hopeless. When we have a lot of data, the problem actually becomes *easier*, since one structure stands out from the rest. In fact, recent results indicates that there might be search procedures that when applied to sufficiently large data sets are guaranteed to reach the global optimum. This suggests that the hard cases might be the ones where the differences between the maximal scoring network and that other local maxima might not be that dramatic. This is a rough intuition, and it is an open problem to characterize formally the trade-off between quality of solution and hardness of the score-based learning problem.

Another open direction of research attempts to combine the best of both worlds. Can we use the efficient procedures developed for constraint-based learning to find high-scoring network structure? The high-level motivation that the Build-PDAG we discussed uses knowledge about Bayesian networks to direct its actions. On the other hand, the search procedures we discussed so far are fairly uninformed about the problem. A simpleminded combination of these two approaches uses a constraint-based method to find starting point for the heuristic search. More elaborate strategies attempt to use the insight from constraint-based learning to reformulate the search space — for example, to avoid exploring structures that are clearly not going to score well, or to consider global operators.

Another issue that we touched on is estimating the confidence in the structures we learned. We discussed MCMC approaches for answering questions about the posterior. This gives us a measure of our confidence in the structures we learned. In particular, we can see whether a part of the learned network is "crucial" in the sense that it has high posterior probability, or closer to arbitrary when it has low posterior probability. Such an evaluation, however, compares structures only within the class of models we are willing to learn. It is possible that the data do not match any of these structures. In such situations, the posterior may not be informative about the problem. The statistical literature addresses such questions under the name of *goodness of fit* tests, which we briefly described in box 16.A. These tests attempt to evaluate whether a given model would have data such as the one we observed. This topic is still underdeveloped for models such as Bayesian networks.

goodness of fit

18.8 Relevant Literature

We begin by noting that many of the works on learning Bayesian networks involve both parameter estimation and structure learning; hence most of the references discussed in section 17.8 are still relevant to the discussion in this chapter.

The constraint-based approaches to learning Bayesian networks were already discussed in chapter 3, under the guise of algorithms for constructing a perfect map for a given distribution. Section 3.6 provides the references relevant to that work; some of the key algorithms of this type include those of Pearl and Verma (1991); Verma and Pearl (1992); Spirtes, Glymour, and Scheines (1993); Meek (1995a); Cheng, Greiner, Kelly, Bell, and Liu (2002). The application of these methods to the task of learning from an empirical distribution requires the use of statistical independence tests (see, for example, Lehmann and Romano (2008)). However, little work has been devoted to analyzing the performance of these algorithms in that setting, when some of the tests may fail.

Much work has been done on the development and analysis of different scoring functions for probabilistic models, including the BIC/MDL score (Schwarz 1978; Rissanen 1987; Barron et al. 1998) and the Bayesian score (Dawid 1984; Kass and Raftery 1995), as well as other scores, such as the AIC (Akaike 1974). These papers also establish the basic properties of these scores, such as the consistency of the BIC/MDL and Bayesian scores.

The Bayesian score for discrete Bayesian networks, using a Dirichlet prior, was first proposed by Buntine (1991) and Cooper and Herskovits (1992) and subsequently generalized by Spiegelhalter, Dawid, Lauritzen, and Cowell (1993); Heckerman, Geiger, and Chickering (1995). In particular, Heckerman et al. propose the BDe score, and they show that the BDe prior is the only one satisfying certain natural assumptions, including global and local parameter independence and score-equivalence. Geiger and Heckerman (1994) perform a similar analysis for Gaussian networks, resulting in a formal justification for a Normal-Wishart prior that they call the *BGe prior*. The application of MDL principles to Bayesian network learning was developed in parallel (Bouckaert 1993; Lam and Bacchus 1993; Suzuki 1993). These papers also defined the relationship between the maximum likelihood score and information theoretic scores. These connections in a more general setting were explored in early works in information theory Kullback (1959), as well as in early work on decomposable models Chow and Liu (1968).

BGe prior

Buntine (1991) first explored the use of nonuniform priors over Bayesian network structures, utilizing a prior over a fixed node ordering, where all edges were included independently. Heckerman, Mamdani, and Wellman (1995) suggest an alternative approach that uses the extent of deviation between a candidate structure and a "prior network."

Perhaps the earliest application of structure search for learning in Bayesian networks was the work of Chow and Liu (1968) on learning tree-structured networks. The first practical algorithm for learning general Bayesian network structure was proposed by Cooper and Herskovits (1992). Their algorithm, known as the *K2 algorithm*, was limited to the case where an ordering on the variables is given, allowing families for different variables to be selected independently. The approach of using local search over the space of general network structures was proposed and studied in depth by Chickering, Geiger, and Heckerman (1995) (see also Heckerman et al. 1995), although initial ideas along those lines were outlined by Buntine (1991). Chickering et al. compare different search algorithms, including K2 (with different orderings), local search, and simulated annealing. Their results suggest that unless a good ordering is known, local search offers the best time-accuracy trade-off. Tabu search is discussed by Glover and Laguna (1993). Several

works considered the combinatorial problem of searching over all network structures (Singh and Moore 2005; Silander and Myllymaki 2006) based on ideas of Koivisto and Sood (2004).

Another line of research proposes local search over a different space, or using different operators. Best known in this category are algorithms, such as the GES algorithm, that search over the space of I-equivalence classes. The foundations for this type of search were developed by Chickering (1996b, 2002a). Chickering shows that GES is guaranteed to learn the optimal Bayesian network structure at the large sample limit, if the data is sampled from a graphical model (directed or undirected). Other algorithms that guarantee identification of the correct structure at the large-sample limit include the constraint-based SGS method of Spirtes, Glymour, and Scheines (1993) and the KES algorithm of Nielsen, Kočka, and Peña (2003).

Other search methods that use alternative search operators or search spaces include the *optimal reinsertion* algorithm of Moore and Wong (2003), which takes a variable and moves it to a new position in the network, and the *ordering-based search* of Teyssier and Koller (2005), which searches over the space of orderings, selecting, for each ordering the optimal (bounded-indegree) network consistent with it. Both of these methods take much larger steps in the space than search over the space of network structures; although each step is also more expensive, empirical results show that these algorithms are nevertheless faster. More importantly, they are significantly less susceptible to local optima. A very different approach to avoiding local optima is taken by the data perturbation method of Elidan et al. (2002), in which the sufficient statistics are perturbed to move the algorithm out of local optima.

Much work has been done on efficient caching of sufficient statistics for machine learning tasks in general, and for Bayesian networks in particular. Moore and Lee (1997); Komarek and Moore (2000) present AD-trees and show their efficacy for learning Bayesian networks in high dimension. Deng and Moore (1989); Moore (2000); Indyk (2004) present some data structures for continuous spaces.

Several papers also study the theoretical properties of the Bayesian network structure learning task. Some of these papers involve the computational feasibility of the task. In particular, Chickering (1996a) showed that the problem of finding the network structure with indegree $\leq d$ that optimizes the Bayesian score for a given data set is $\mathcal{NP}$-hard, for any $d \geq 2$. $\mathcal{NP}$-hardness is also shown for finding the maximum likelihood structures within the class of polytree networks (Dasgupta 1999) and path-structured networks (Meek 2001). Chickering et al. (2003) show that the problem of finding the optimal structure is also $\mathcal{NP}$-hard at the large-sample limit, even when: the generating distribution is perfect with respect to some DAG containing hidden variables, we are given an independence oracle, we are given an inference oracle, and we restrict potential solutions to structures in which each node has at most d parents (for any $d \geq 3$). Importantly, all of these $\mathcal{NP}$-hardness results hold only in the inconsistent case, that is, where the generating distribution is not perfect for some DAG. In the case where the generating distribution is perfect for some DAG over the observed variables, the problem is significantly easier. As we discussed, several algorithms can be guaranteed to identify the correct structure. In fact, the constraint-based algorithms of Spirtes et al. (1993); Cheng et al. (2002) can be shown to have polynomial-time performance if we assume a bounded indegree (in both the generating and the learned network), providing a sharp contrast to the $\mathcal{NP}$-hardness result in the inconsistent setting.

Little work has been done on analyzing the PAC-learnability of Bayesian network structures, that is, their learnability as a function of the number of samples. A notable exception is the work

of Höffgen (1993), who analyzes the problem of PAC-learning the structure of Bayesian networks with bounded indegree. He focuses on the maximum likelihood network subject to the indegree constraints and shows that this network has, with high probability, low KL-divergence to the true distribution, if we learn with a number of samples M that grows logarithmically with the number of variables in the network. Friedman and Yakhini (1996) extend this analysis for search with penalized likelihood — for example, MDL/BIC scores. We note that, currently, no efficient algorithm is known for finding the maximum-likelihood Bayesian network of bounded indegree, although the work of Abbeel, Koller, and Ng (2006) does provide a polynomial-time algorithm for learning a low-degree factor graph under these assumptions.

Bayesian model averaging has been used in the context of density estimation, as a way of gaining more robust predictions over those obtained from a single model. However, most often it is used in the context of knowledge discovery, to obtain a measure of confidence in predictions relating to structural properties. In a limited set of cases, the space of legal networks is small enough to allow a full enumeration of the set of possible structures (Heckerman et al. 1999). Buntine (1991) first observed that in the case of a fixed ordering, the exponentially large summation over model structures can be reformulated compactly. Meila and Jaakkola (2000) show that one can efficiently infer and manipulate the full Bayesian posterior over all the superexponentially many tree-structured networks. Koivisto and Sood (2004) suggest an exact method for summing over all network structure. Although this method is exponential in the number of variables, it can deal with domains of reasonable size.

Other approaches attempt to approximate the superexponentially large summation by considering only a subset of possible structures. Madigan and Raftery (1994) propose a heuristic approximation, but most authors use an MCMC approach over the space of Bayesian network structure (Madigan and York 1995; Madigan et al. 1996; Giudici and Green 1999). Friedman and Koller (2003) propose the use of MCMC over orderings, and show that it achieves much faster mixing and therefore more robust estimates than MCMC over network space. Ellis and Wong (2008) improve on this algorithm, both in removing bias in its prior distribution and in improving its computational efficiency.

The idea of using local learning of tree-structured CPDs for learning global network structure was proposed by Friedman and Goldszmidt (1996, 1998), who also observed that reducing the number of parameters in the CPD can help improve the global structure reconstruction. Chickering et al. (1997) extended these ideas to CPDs structured as a decision graph (a more compact generalization of a decision tree). Structure learning in dynamic Bayesian networks was first proposed by Friedman, Murphy, and Russell (1998). Structure learning in object-relational models was first proposed by Friedman, Getoor, Koller, and Pfeffer (1999); Getoor, Friedman, Koller, and Taskar (2002). Segal et al. (2005) present the module network framework, which combines clustering with structure learning.

tree-augmented naive Bayes

Learned Bayesian networks have also been used for specific prediction and classification tasks. Friedman et al. (1997) define a *tree-augmented naive Bayes* structure, which extends the traditional naive Bayes classifier by allowing a tree-structure over the features. They demonstrate that this enriched model provides significant improvements in classification accuracy. Dependency networks were introduced by Heckerman et al. (2000) and applied to a variety of settings, including collaborative filtering. This latter application extended the earlier work of Breese et al. (1998), which demonstrated the success of Bayesian network learning (with tree-structured CPDs) to this task.

18.9 Exercises

Exercise 18.1

Show that the $\chi^2(\mathcal{D})$ statistic of equation (18.1) is approximately twice of $d_{\mathbf{I}}(\mathcal{D})$ of equation (18.2). Hint: examine the first-order Taylor expansion of $\frac{z}{t} \approx 1 + \frac{z-t}{t}$ in z around t.

Exercise 18.2

Derive equation (18.3). Show that this value is the sum of the probability of all possible data sets that have the given empirical counts.

Exercise 18.3⋆

multiple hypothesis testing

Suppose we are testing *multiple hypotheses* $1, \ldots, N$ for a large value. Each hypothesis has an observed deviance measure D_i, and we computed the associated p-value p_i. Recall that under the null hypothesis, p_i has a uniform distribution between 0 and 1. Thus, $P(p_i < t \mid H_0) = t$ for every $t \in [0, 1]$.

We are worried that one of our tests received a small p-value by chance. Thus, we want to consider the distribution of the *best* p-value out of all of our tests under the assumption that the null hypothesis is true in all of the tests. More formally, we want to examine the behavior of $\min_i p_i$ under H_0.

a. Show that

$$P(\min_i p_i < t \mid H_0) \leq t \cdot N.$$

(Hint: Do *not* assume that the variables p_i are independent of each other.)

b. Suppose we want to ensure that the probability of a random rejection is below 0.05, what p-value should we use in individual hypothesis tests?

c. Suppose we assume that the tests (that is, the variables p_i) are independent of each other. Derive the bound in this case. Does this bound give better (higher) decision p-value when we use $N = 100$ and global rejection rate below 0.05 ? How about $N = 1,000$?

Bonferroni correction

The bound you derive in [b] is called *Bonferroni bound*, or more often *Bonferroni correction* for a multiple-hypothesis testing scenario.

Exercise 18.4⋆

Consider again the Build-PDAG procedure of algorithm 3.5, but now assume that we apply it in a setting where the independence tests might return incorrect answers owing to limited and noisy data.

a. Provide an example where Build-PDAG can fail to reconstruct the true underlying graph $\mathcal{G}^*$ even in the presence of a single incorrect answer to an independence question.

b. Now, assume that the algorithm constructs the correct skeleton but can encounter a single incorrect answer when extracting the immoralities.

Exercise 18.5

Prove corollary 18.1. Hint: Start with the result of proposition 18.1, and use the chain rule of entropies and the chain rule mutual information.

Exercise 18.6

Show that adding edges to a network increases the likelihood.

Exercise 18.7⋆

Stirling's approximation

Prove theorem 18.1. Hint: You can use *Stirling's approximation* for the Gamma function

$$\Gamma(x) \approx \sqrt{2\pi}\, x^{x-\frac{1}{2}} e^{-x}$$

or

$$\log \Gamma(x) \approx \frac{1}{2}\log(2\pi)x\log(x) - \frac{1}{2}\log(x) - x$$

Exercise 18.8

Show that if $\mathcal{G}$ is I-equivalent to $\mathcal{G}'$, then if we use table-CPDs, we have that $\text{score}_L(\mathcal{G} : \mathcal{D}) = \text{score}_L(\mathcal{G}' : \mathcal{D})$ for any choice of $\mathcal{D}$.

Hint: Consider the set of distributions that can be represented by parameterization each network structure.

Exercise 18.9

Show that if $\mathcal{G}$ is I-equivalent to $\mathcal{G}'$, then if we use table-CPDs, we have that $\text{score}_{BIC}(\mathcal{G} : \mathcal{D}) = \text{score}_{BIC}(\mathcal{G}' : \mathcal{D})$ for any choice of $\mathcal{D}$. You can use the results of exercise 3.18 and exercise 18.8 in your proof.

Exercise 18.10

Show that the Bayesian score with a K2 prior in which we have a Dirichlet prior $Dirichlet(1, 1, \ldots, 1)$ for each set of multinomial parameters is not score-equivalent.

Hint: Construct a data set for which the score of the network $X \rightarrow Y$ differs from the score of the network $X \leftarrow Y$.

Exercise 18.11⋆

We now examine how to prove score equivalence for the BDe score. Assume that we have a prior specified by an equivalent sample size α and prior distribution P'. Prove the following:

a. Consider networks over the variables X and Y. Show that the BDe score of $X \rightarrow Y$ is equal to that of $X \leftarrow Y$.
b. Show that if $\mathcal{G}$ and $\mathcal{G}'$ are identical except for a covered edge reversal of $X \rightarrow Y$, then the BDe score of both networks is equal.
c. Show that the proof of score equivalence follows from the last result and theorem 3.9.

Exercise 18.12⋆

In section 18.3.2, we have seen that the Bayesian score can be posed a sequential procedure that estimates the performance on new unseen examples. In this example, we consider another score that is based on this motivation.

Recall that leave-one-out cross-validation (LOOCV), described in box 16.A, is a procedure for estimating the performance of a learning method on new samples. In our context, this defines the following score:

$$\textit{LOOCV}(\mathcal{D} \mid \mathcal{G}) = \prod_{m=1}^{M} P(\xi[m] \mid \mathcal{G}, \mathcal{D}_{-m}),$$

where $\mathcal{D}_{-m}$ is the data with the m'th instance removed.

a. Consider the setting of section 18.3.3 where we observe a series of values of a binary variable. Develop a closed-form equation for $\textit{LOOCV}(\mathcal{D} \mid \mathcal{G})$ as a function of the number of heads and the number of tails.
b. Now consider the network $\mathcal{G}_{X\rightarrow Y}$ and a data set $\mathcal{D}$ that consists of observations of two binary variables X and Y. Develop a closed-form equation for $\textit{LOOCV}(\mathcal{D} \mid \mathcal{G}_{X\rightarrow Y})$ as a function of the sufficient statistics of X and Y in $\mathcal{D}$.
c. Based on these two examples, what are the properties of the LOOCV score? Is it decomposable?

Exercise 18.13

Consider the algorithm for learning tree-structured networks in section 18.4.1. Show that the weight $w_{i \to j} = w_{j \to i}$ if the score satisfies score equivalence.

Exercise 18.14⋆⋆

We now consider a situation where we can find the high-scoring structure in a polynomial time. Suppose we are given a directed graph $\mathcal{C}$ that imposes constraints on the possible parent-child relationships in the learned network: an edge $X—Y$ in $\mathcal{C}$ implies that X might be considered as a parent of Y. We define $\boldsymbol{\mathcal{G}}_{\mathcal{C}} = \{\mathcal{G} : \mathcal{G} \subseteq \mathcal{C}\}$ to be the set of graphs that are consistent with the constraints imposed by C.

Describe an algorithm that, given a decomposable score score and a data set $\mathcal{D}$, finds the maximal scoring network in $\boldsymbol{\mathcal{G}}_{\mathcal{C}}$. Show that your algorithm is exponential in the tree-width of the graph C.

Exercise 18.15⋆

Consider the problem of learning a Bayesian network structure over two random variables X and Y.

a. Show a data set — an empirical distribution and a number of samples M — where the optimal network structure according to the BIC scoring function is different from the optimal network structure according to the ML scoring function.
b. Assume that we continue to get more samples that exhibit precisely the same empirical distribution. (For simplicity, we restrict attention to values of M that allow that empirical distribution to be achieved; for example, an empirical distribution of 50 percent heads and 50 percent tails can be achieved only for an even number of samples.) At what value of M will the network that optimizes the BIC score be the same as the network that optimizes the likelihood score?

Exercise 18.16

This problem considers the performance of various types of structure search algorithms. Suppose we have a general network structure search algorithm, $\boldsymbol{A}$, that takes a set of basic operators on network structures as a parameter. This set of operators defines the search space for $\boldsymbol{A}$, since it defines the candidate network structures that are the "immediate successors" of any current candidate network structure—that is, the successor states of any state reached in the search. Thus, for example, if the set of operators is {add an edge not currently in the network}, then the successor states of any candidate network $\mathcal{G}$ is the set of structures obtained by adding a single edge anywhere in $\mathcal{G}$ (so long as acyclicity is maintained).

Given a set of operators, $\boldsymbol{A}$ does a simple greedy search over the set of network structures, starting from the empty network (no edges), using the BIC scoring function. Now, consider two sets of operators we can use in $\boldsymbol{A}$. Let $\boldsymbol{A}_{[add]}$ be $\boldsymbol{A}$ using the set of operations {add an edge not currently in the network}, and let $\boldsymbol{A}_{[add,delete]}$ be $\boldsymbol{A}$ using the set of operations {add an edge not currently in the network, delete an edge currently in the network}.

a. Show a distribution where, regardless of the amount of data in our training set (that is, even with infinitely many samples), the answer produced by $\boldsymbol{A}_{[add]}$ is worse (that is, has a lower BIC score) than the answer produced by $\boldsymbol{A}_{[add,delete]}$. (It is easiest to represent your true distribution in the form of a Bayesian network; that is, a network from which the sample data are generated.)
b. Show a distribution where, regardless of the amount of data in our training set, $\boldsymbol{A}_{[add,delete]}$ will converge to a local maximum. In other words, the answer returned by the algorithm has a lower score than the optimal (highest-scoring) network. What can we conclude about the ability of our algorithm to find the optimal structure?

Exercise 18.17⋆

This problem considers the problem of learning a CPD tree structure for a variable in a Bayesian network, using the Bayesian score. Assume that the network structure $\mathcal{G}$ includes a description of the CPD trees in

it; that is, for each variable X_i, we have a CPD tree $\mathcal{T}_i$ for $P(X_i \mid \mathrm{Pa}_{X_i}^{\mathcal{G}})$. We view the choice of the trees as part of the specification of the model. Thus, we consider the score of $\mathcal{G}$ with $\mathcal{T}_1, \ldots, \mathcal{T}_n$

$$\mathrm{score}_B(\mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n \ : \ \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n) + \log P(\mathcal{G}) + \sum_i \log P(\mathcal{T}_i \mid \mathcal{G}).$$

Here, we assume for simplicity that the two structure priors are uniform, so that we focus on the marginal likelihood term $P(\mathcal{D} \mid \mathcal{G}, \mathcal{T}_1, \ldots, \mathcal{T}_n)$. Assume we have selected a fixed choice of parents $\boldsymbol{U}_i$ for each variable X_i. We would like to find a set of trees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ that together maximizes the Bayesian score.

a. Show how you can decompose the Bayesian score in this case as a sum of simpler terms; make sure you state the assumptions necessary to allow this decomposition.
b. Assume that we consider only a single type of operator in our search, a *split*(X_i, ℓ, Y) operator, where ℓ is a leaf in the current CPD tree for X_i, and $Y \in \boldsymbol{U}_i$ is a possible parent of X_i. This operator replaces the leaf ℓ by an internal variable that splits on the values of Y. Derive a simple formula for the delta-score $\delta(\mathcal{T} \ : \ o)$ of such an operator $o =$ *split*(X_i, ℓ, Y). (Hint: Use the representation of the decomposed score to simplify the formula.)
c. Suppose our greedy search keeps track of the delta-score for all the operators. After we take a step in the search space by applying operator $o =$ *split*(X, ℓ, Y), how should we update the delta score for another operator $o' =$ *split*(X', ℓ', Y')? (Hint: Use the representation of the delta-score in terms of decomposed score in the previous question.)
d. Now, consider applying the same process using the likelihood score rather than the Bayesian score. What will the resulting CPD trees look like in the general case? You can make any assumption you want about the behavior of the algorithm in case of ties in the score.

Exercise 18.18

Prove proposition 18.5.

Exercise 18.19

Prove proposition 18.6.

Exercise 18.20⋆

Recall that the $\Theta(f(n))$ denotes both an asymptotic lower bound and an asymptotic upper bound (up to a constant factor).

a. Show that the number of DAGs with n vertices is $2^{\Theta(n^2)}$.
b. Show that the number of DAGs with n vertices and indegree bounded by d is $2^{\Theta(dn \log n)}$.
c. Show that the number of DAGs with n vertices and indegree bounded by d that are consistent with a given order is $2^{\Theta(dn \log n)}$.

Exercise 18.21

Consider the problem of learning the structure of a 2-TBN over $\mathcal{X} = \{X_1, \ldots, X_n\}$. Assume that we are learning a model with bounded indegree k. Explain, using the argument of asymptotic complexity, why the problem of learning the 2-TBN structure is considerably easier if we assume that there are no intra-time-slice edges in the 2-TBN.

Exercise 18.22⋆

In this problem, we will consider the task of learning a generalized type of Bayesian networks that involves shared structure and parameters. Let $\mathcal{X}$ be a set of variables, which we assume are all binary-valued. A *module network* over $\mathcal{X}$ partitions the variables $\mathcal{X}$ into K disjoint clusters, for $K \ll n = |\mathcal{X}|$. All of the variables assigned to the same cluster have precisely the same parents and CPD. More precisely, such a network defines:

module network

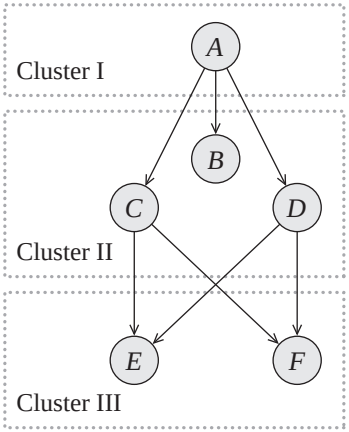


Figure 18.11 A simple module network

- An assignment function $\mathcal{A}$, which defines for each variable X, a cluster assignment $\mathcal{A}(X) \in \{C_1, \ldots, C_K\}$.
- For each cluster C_k ($k = 1, \ldots, K$), a graph $\mathcal{G}$ that defines a set of parents $\mathrm{Pa}_{C_k} = \boldsymbol{U}_k \subset \mathcal{X}$ and a CPD $P_k(X \mid \boldsymbol{U}_k)$.

The module network structure defines a ground Bayesian network where, for each variable X, we have the parents $\boldsymbol{U}_k$ for $k = \mathcal{A}(X)$ and the CPD $P_k(X \mid \boldsymbol{U}_k)$. Figure 18.11 shows an example of such a network.

Assume that our goal is to learn a module network that maximizes the Bayesian score given a data set $\mathcal{D}$, where we need to learn both the assignment of variables to clusters and the graph structure.

a. Define an appropriate set of parameters and an appropriate notion of sufficient statistics for this class of models, and write down a precise formula for the likelihood function of a pair $(\mathcal{A}, \mathcal{G})$ in terms of the parameters and sufficient statistics.
b. Draw the meta-network for the module network shown in figure 18.11. Assuming a uniform prior over each parameter, write down exactly (normalizing constants included) the appropriate parameter posterior given a data set $\mathcal{D}$.
c. We now turn to the problem of learning the structure of the cluster network. We will use local search, using the following types of operators:
   - **Add** operators that add a parent for a cluster;
   - **Delete** operators that delete a parent for a cluster;
   - **Node-Move** operators $o_{k \to k'}(X)$ that change from $\mathcal{A}(X) = k$ to $\mathcal{A}(X) = k'$.

   Describe an efficient implementation of the **Node-Move** operator.
d. For each type of operator, specify (precisely) which other operators need to be reevaluated once the operator has been taken? Briefly justify your response.
e. Why did we not include edge reversal in our set of operators?

Exercise 18.23⋆

reinsertion operator

It is often useful when learning the structure of a Bayesian network to consider more global search operations. In this problem we will consider an operator called *reinsertion*, which works as follows: For the current structure $\mathcal{G}$, we choose a variable X_i to be our *target variable*. The first step is to remove

the variable from the network by severing all connections to its children and parents. We then select the optimal set of at most K_p parents and at most K_c children for X and reinsert it into the network with edges from the selected parents and to the selected children. Throughout this problem, assume the use of the BIC score for structure evaluation.

a. Let X_i be our current target variable, and assume for the moment that we have somehow chosen $\boldsymbol{U}_i$ to be optimal parents of X_i. Consider the case of $K_c = 1$, where we want to choose the *single* optimal child for X_i. Candidate children — those that do not introduce a cycle in the graph — are $Y_1, \ldots, Y_\ell$. Write an argmax expression for finding the optimal child C. Explain your answer.
b. Now consider the case of $K_c = 2$. How do we find the optimal pair of children? Assuming that our family score for any $\{X_k, \boldsymbol{U}_k\}$ can be computed in a constant time f, what is the best asymptotic computational complexity of finding the optimal pair of children? Explain. Extend your analysis to larger values of K_c. What is the computational complexity of this task?
c. We now consider the choice of parents for X_i. We now assume that we have already somehow chosen the optimal set of children and will hold them fixed. Can we do the same trick when choosing the parents? If so, show how. If not, argue why not.

Exercise 18.24

Prove proposition 18.7.

Exercise 18.25

Prove proposition 18.8.

Exercise 18.26⋆

Consider the idea of searching for a high-scoring network by searching over the space of orderings $\prec$ over the variables. Our task is to search for a high-scoring network that has bounded indegree k. For simplicity, we focus on the likelihood score. For a given order $\prec$, let $\mathcal{G}^*_\prec$ be the highest-likelihood network consistent with the ordering $\prec$ of bounded indegree k. We define $\text{score}_L(\prec : \mathcal{D}) = \text{score}_L(\mathcal{G}^*_\prec : \mathcal{D})$. We now search over the space of orderings using operators o that swap two adjacent nodes in the ordering, that is:

$$X_1, \ldots, X_{i-1}, X_i, X_{i+1}, \ldots, X_n \mapsto X_1, \ldots, X_{i-1}, X_{i+1}, X_i, \ldots, X_n.$$

Show how to use score decomposition properties to search this space efficiently.

Exercise 18.27⋆

Consider the choice between $\mathcal{G}_\emptyset$ and $\mathcal{G}_{X\to Y}$ given a data set of joint observations of binary variables X and Y.

a. Show that $\text{score}_{BIC}(\mathcal{G}_{X\to Y} : \mathcal{D}) > \text{score}_{BIC}(\mathcal{G}_\emptyset : \mathcal{D})$ if and only if $\boldsymbol{I}_{\hat{P}_\mathcal{D}}(X;Y) > c$. What is this constant c?
b. Suppose that we have BDe prior with uniform P' and $\alpha = 0$. Write the condition on the counts when $\text{score}_{BDe}(\mathcal{G}_{X\to Y} : \mathcal{D}) > \text{score}_{BDe}(\mathcal{G}_\emptyset : \mathcal{D})$.
c. Consider empirical distributions of the form discussed in figure 18.3. That is, $\hat{P}(x,y) = 0.25 + 0.5 \cdot p$ if x and y are equal, and $\hat{P}(x,y) = 0.25 - 0.5 \cdot p$ otherwise. For different values of p, plot as a function of M both the χ^2 deviance measure and the mutual information. What do you conclude about these different functions?
d. Implement the exact p-value test described in section 18.2.2.3 for the χ^2 and the mutual information deviance measures.
e. Using the same empirical distribution, plot for different values of M the *decision boundary* between $\mathcal{G}_\emptyset$ and $\mathcal{G}_{X\to Y}$ for each of the three methods we considered in this exercise. That is, find the value of p at which the two alternatives have (approximately) equal score, or at which the p-value of rejecting the null hypothesis is (approximately) 0.05.

   What can you conclude about the differences between these structure selection methods?

19 *Partially Observed Data*

Until now, our discussion of learning assumed that the training data are *fully observed*: each instance assigns values to all the variables in our domain. This assumption was crucial for some of the technical developments in the previous two chapters. Unfortunately, this assumption is clearly unrealistic in many settings. In some cases, data are missing by accident; for example, some fields in the data may have been omitted in the data collection process. In other cases, certain observations were simply not made; in a medical-diagnosis setting, for example, one never performs all possible tests or asks all of the possible questions. Finally, some variables are *hidden*, in that their values are never observed. For example, some diseases are not observed directly, but only via their symptoms.

hidden variable

In fact, in many real-life applications of learning, the available data contain missing values. Hence, we must address the learning problem in the presence of *incomplete data*. As we will see, incomplete data pose both foundational problems and computational problems. The foundational problems are in formulating an appropriate learning task and determining when we can expect to learn from such data. The computational problems arise from the complications incurred by incomplete data and the construction of algorithms that address these complications.

incomplete data

In the first section, we discuss some of the subtleties encountered in learning from incomplete data and in formulating an appropriate learning problem. In subsequent sections, we examine techniques for addressing various aspects of this task. We focus initially on the parameter-learning task, assuming first that the network structure is given, and then treat the more complex structure-learning question at the end of the chapter.

19.1 Foundations

19.1.1 Likelihood of Data and Observation Models

A central concept in our discussion of learning so far was the likelihood function that measures the probability of the data induced by different choices of models and parameters. The likelihood function plays a central role both in maximum likelihood estimation and in Bayesian learning. In these developments, the likelihood function was determined by the probabilistic model we are learning. Given a choice of parameters, the model defined the probability of each instance. In the case of fully observed data, we assumed that each instance $\xi[m]$ in our training set $\mathcal{D}$ is simply a random sample from the model.

It seems straightforward to extend this idea to incomplete data. Suppose our domain consists

of two random variables X and Y, and in one particular instance we observed only the value of X to be $x[m]$, but not the value of Y. Then, it seems natural to assign the instance the probability $P(x[m])$. More generally, the likelihood of an incomplete instance is simply the marginal probability given our model. Indeed, the most common approach to define the likelihood of an incomplete data set is to simply marginalize over the unobserved variables.

This approach, however, embodies some rather strong assumptions about the nature of our data. To learn from incomplete data, we need to understand these assumptions and examine the situation much more carefully. Recall that when learning parameters for a model, we assume that the data were generated according to the model, so that each instance is a sample from the model. **When we have missing data, the data-generation process actually involves two steps. In the first step, data are generated by sampling from the model. In this step, values of all the variables are selected. The next step determines which values we get to observe and which ones are hidden from us.** In some cases, this process is simple; for example, some particular variable may always be hidden. In other situations, this process might be much more complex.

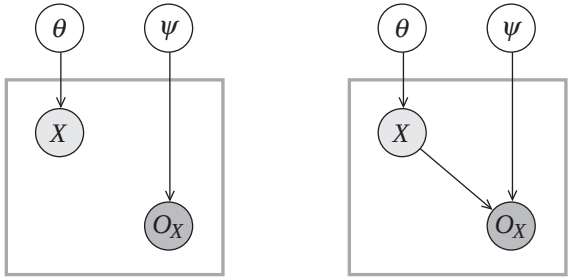
To analyze the probabilistic model of the observed training set, we must consider not only the data-generation mechanism, but also the mechanism by which data are hidden. Consider the following two examples.

Example 19.1 *We flip a thumbtack onto a table, and every now and then it rolls off the table. Since a fall from the table to the floor is quite different from our desired experimental setup, we do not use results from these flips (they are missing). How would that change our estimation? The simple solution is to ignore the missing values and simply use the counts from the flips that we did get to observe. That is, we pretend that missing flips never happened. As we will see, this strategy can be shown to be the correct one to use in this case.* ■

Example 19.2 *Now, assume that the experiment is performed by a person who does not like "tails" (because the point that sticks up might be dangerous). So, in some cases when the thumbtack lands "tails," the experimenter throws the thumbtack on the floor and reports a missing value. However, if the thumbtack lands "heads," he will faithfully report it. In this case, the solution is also clear. We can use our knowledge that every missing value is "tails" and count it as such. Note that this leads to very different likelihood function (and hence estimated parameters) from the strategy that we used in the previous case.*

While this example may seem contrived, many real-life scenarios have very similar properties. For example, consider a medical trial evaluating the efficacy of a drug, but one where patients can drop out of the trial, in which case their results are not recorded. If patients drop out at random, we are in the situation of example 19.1; on the other hand, if patients tend to drop out only when the drug is not effective for them, the situation is essentially analogous to the one in this example. ■

Note that in both examples, we observe sequences of the form $H, T, H, ?, T, ?, \ldots$, but nevertheless we treat them differently. The difference between these two examples is our knowledge about the observation mechanism. As we discussed, each observation is derived as a combination of two mechanisms: the one that determines the outcome of the flip, and the one that determines whether we observe the flip. Thus, our training set actually consists of two variables for each flip: the flip outcome X, and the observation variable O_X, which tells us whether we observed the value of X.



(a) Random missing values (b) Deliberate missing values

Figure 19.1 Observation models in two variants of the thumbtack example

Definition 19.1

observability variable

observability model

Let $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ be some set of random variables, and let $O_{\boldsymbol{X}} = \{O_{X_1}, \ldots, O_{X_n}\}$ be their observability variable. *The* observability model *is a joint distribution $P_{missing}(\boldsymbol{X}, O_{\boldsymbol{X}}) = P(\boldsymbol{X}) \cdot P_{missing}(O_{\boldsymbol{X}} \mid \boldsymbol{X})$, so that $P(\boldsymbol{X})$ is parameterized by parameters $\boldsymbol{\theta}$, and $P_{missing}(O_{\boldsymbol{X}} \mid \boldsymbol{X})$ is parameterized by parameters $\boldsymbol{\psi}$.*

We define a new set of random variables $\boldsymbol{Y} = \{Y_1, \ldots, Y_n\}$, where $Val(Y_i) = Val(X_i) \cup \{?\}$. The actual observation is $\boldsymbol{Y}$, which is a deterministic function of $\boldsymbol{X}$ and $O_{\boldsymbol{X}}$,

$$Y_i = \begin{cases} X_i & O_{X_i} = o^1 \\ ? & O_{X_i} = o^0. \end{cases}$$

The variables $Y_1, \ldots, Y_n$ represent the values we actually observe, either an actual value or a ? that represents a missing value. ■

Thus, we observe the $\boldsymbol{Y}$ variable. This observation always implies that we know the value of the $O_{\boldsymbol{X}}$ variables, and whenever $Y_i \neq ?$, we also observe the value of X_i. To illustrate the definition of this concept, we consider the probability of the observed value Y in the two preceding examples.

Example 19.3

In the scenario of example 19.1, we have a parameter θ that describes the probability of $X = 1$ (Heads), and another parameter ψ that describes the probability of $O_X = o^1$. Since we assume that the hiding mechanism is random, we can describe this scenario by the meta-network of figure 19.1a. This network describes how the probability of different instances (shown as plates) depend on the parameters. As we can see, this network consists of two independent *subnetworks. The first relates the values of X in the different examples to the parameter θ, and the second relates the values of O_X to ψ.*

Recall from our earlier discussion that if we can show that θ and ψ are independent given the evidence, then the likelihood decomposes into a product. We can derive this decomposition as follows. Consider the three values of Y and how they could be attained. We see that

$$\begin{aligned} P(Y = 1) &= \theta\psi \\ P(Y = 0) &= (1 - \theta)\psi \\ P(Y = ?) &= (1 - \psi). \end{aligned}$$

Thus, if we see a data set $\mathcal{D}$ of tosses with $M[1]$, $M[0]$, and $M[?]$ instances that are Heads, Tails, and ?, respectively, then the likelihood is

$$L(\theta, \psi : \mathcal{D}) = \theta^{M[1]}(1-\theta)^{M[0]}\psi^{M[1]+M[0]}(1-\psi)^{M[?]}.$$

As we expect, the likelihood function in this example is a product of two functions: a function of θ, and a function of ψ. We can easily see that the maximum likelihood estimate of θ is $\frac{M[1]}{M[1]+M[0]}$, while the maximum likelihood estimate of ψ is $\frac{M[1]+M[0]}{M[1]+M[0]+M[?]}$.

We can also reach the conclusion regarding independence using a more qualitative analysis. At first glance, it appears that observing Y activates the v-structure between X and O_X, rendering them dependent. However, the CPD of Y has a particular structure, which induces context-specific independence. In particular, we see that X and O_X are conditionally independent given both values of Y: when $Y = ?$, then O_X is necessarily o^0, in which case the edge $X \to Y$ is spurious (as in definition 5.7); if $Y \neq ?$, then Y deterministically establishes the values of both X and O_X, in which case they are independent. ■

Example 19.4

Now consider the scenario of example 19.2. Recall that in this example, the missing values are a consequence of an action of the experimenter after he sees the outcome of the toss. Thus, the probability of missing values depends on the value of X. To define the likelihood function, suppose θ is the probability of $X = 1$. The observation parameters ψ consist of two values: $\psi_{O_X|x^1}$ is probability $O_X = o^1$ when $X = 1$, and $\psi_{O_X|x^0}$ is the probability of $O_X = o^1$ when $X = 0$.

We can describe this scenario by the meta-network of figure 19.1b. In this network, O_X depends directly on X. When we get an observation $Y = ?$, we essentially observe the value of O_X but not of X. In this case, due to the direct edge between X and O_X, the context-specific independence properties of Y do not help: X and O_X are correlated, and therefore so are their associated parameters. Thus, we cannot conclude that the likelihood decomposes.

Indeed, when we examine the form of the likelihood, this becomes apparent. Consider the three values of Y and how they could be attained. We see that

$$\begin{aligned} P(Y=1) &= \theta\psi_{O_X|x^1} \\ P(Y=0) &= (1-\theta)\psi_{O_X|x^0} \\ P(Y=?) &= \theta(1-\psi_{O_X|x^1}) + (1-\theta)(1-\psi_{O_X|x^0}). \end{aligned}$$

And so, if we see a data set $\mathcal{D}$ of tosses with $M[1]$, $M[0]$, and $M[?]$ instances that are Heads, Tails, and ?, respectively, then the likelihood is

$$\begin{aligned} &L(\theta, \psi_{O_X|x^1}, \psi_{O_X|x^0} : \mathcal{D}) \\ &\quad = \theta^{M[1]}(1-\theta)^{M[0]}\psi_{O_X|x^1}^{M[1]}\psi_{O_X|x^0}^{M[0]} \\ &\qquad (\theta(1-\psi_{O_X|x^1}) + (1-\theta)(1-\psi_{O_X|x^0}))^{M[?]}. \end{aligned}$$

As we can see, the likelihood function in this example is more complex than the one in the previous example. In particular, there is no easy way of decoupling the likelihood of θ from the likelihood of $\psi_{O_X|x^1}$ and $\psi_{O_X|x^0}$. This makes sense, since different values of these parameters imply different possible values of X when we see a missing value and so affect our estimate of θ; see exercise 19.1. ■

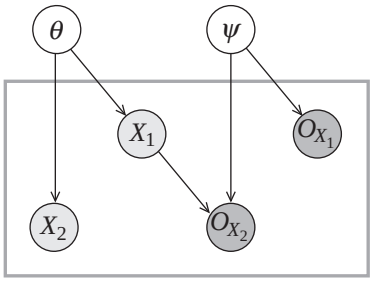


Figure 19.2 An example satisfying MAR but not MCAR. Here, the observability pattern depends on the value of underlying variables.

19.1.2 Decoupling of Observation Mechanism

As we saw in the last two examples, modeling the observation variables, that is, the process that generated the missing values, can result in nontrivial modeling choices, which in some cases result in complex likelihood functions. Ideally, we would hope to avoid dealing with these issues and instead focus on the likelihood of the process that we are interested in (the actual random variables). When can we ignore the observation variables? In the simplest case, the observation mechanism is completely independent of the domain variables. This case is precisely the one we encountered in example 19.1.

Definition 19.2

missing completely at random

A missing data model $P_{missing}$ is missing completely at random (MCAR) *if* $P_{missing} \models (\boldsymbol{X} \perp O_{\boldsymbol{X}})$. ■

In this case, the likelihood of X and O_X decomposes as a product, and we can maximize each part separately. We have seen this decomposition in the likelihood function of example 19.3. The implications of the decoupling is that we can maximize the likelihood of the parameters of the distribution of $\boldsymbol{X}$ without considering the values of the parameters governing the distribution of $O_{\boldsymbol{X}}$. Since we are usually only interested in the former parameters, we can simply ignore the later parameters.

The MCAR assumption is a very strong one, but it holds in certain settings. For example, momentary sensor failures in medical/scientific imaging (for example, flecks of dust) are typically uncorrelated with the relevant variables being measured, and they induce MCAR observation models. Unfortunately, in many other domains the MCAR simply does not hold. For example, in medical records, the pattern of missing values owes to the tests the patient underwent. These, however, are determined by some of the relevant variables, such as the patient's symptoms, the initial diagnosis, and so on.

As it turns out, MCAR is sufficient but not necessary for the decomposition of the likelihood function. We can provide a more general condition where, rather than assuming marginal independence between $O_{\boldsymbol{X}}$ and the values of $\boldsymbol{X}$, we assume only that the observation mechanism is *conditionally independent* of the underlying variables given other observations.

Example 19.5

Consider a scenario where we flip two coins in sequence. We always observe the first toss X_1, and based on its outcome, we decide whether to hide the outcome of the second toss X_2. See figure 19.2

for the corresponding model. In this case, $P_{missing} \models (O_{X_2} \perp X_2 \mid X_1)$. In other words, the true values of both coins are independent of whether they are hidden or not, given our observations.

To understand the issue better, let us write the model and likelihood explicitly. Because we assume that the two coins are independent, we have two parameters θ_{X_1} and θ_{X_2} for the probability of the two coins. In this example, the first coin is always observed, and the observation of the second one depends on the value of the first. Thus, we have parameters $\psi_{O_{X_2}|x_1^1}$ and $\psi_{O_{X_2}|x_1^0}$ that represent the probability of observing X_2 given that X_1 is heads or tails, respectively.

To derive the likelihood function, we need to consider the probability of all possible observations. There are six possible cases, which fall in two categories.

In the first category are the four cases where we observe both coins. By way of example, consider the observation $Y_1 = y_1^1$ and $Y_2 = y_2^0$. The probability of this observation is clearly $P(X_1 = x_1^1, X_2 = x_2^0, O_{X_1} = o^1, O_{X_2} = o^1)$. Using our modeling assumption, we see that this is simply the product $\theta_{X_1}(1 - \theta_{X_2})\psi_{O_{X_2}|x_1^1}$.

In the second category are the two cases where we observe only the first coin. By way of example, consider the observation $Y_1 = y_1^1, Y_2 = ?$. The probability of this observation is $P(X_1 = x_1^1, O_{X_1} = o^1, O_{X_2} = o^0)$. Note that the value of X_2 does not play a role here. This probability is simply the product $\theta_{X_1}(1 - \psi_{O_{X_2}|x_1^1})$.

If we write all six possible cases and then rearrange the products, we see that we can write the likelihood function as

$$\begin{aligned} L(\boldsymbol{\theta} : \mathcal{D}) \quad = \quad & \theta_{X_1}^{M[y_1^1]}(1 - \theta_{X_1})^{M[y_1^0]} \\ & \theta_{X_2}^{M[y_2^1]}(1 - \theta_{X_2})^{M[y_2^0]} \\ & \psi_{O_{X_2}|x_1^1}^{M[y_1^1, y_2^1] + M[y_1^1, y_2^0]}(1 - \psi_{O_{X_2}|x_1^1})^{M[y_1^1, y_2^?]} \\ & \psi_{O_{X_2}|x_1^0}^{M[y_1^0, y_2^1] + M[y_1^0, y_2^0]}(1 - \psi_{O_{X_2}|x_1^0})^{M[y_1^0, y_2^?]}. \end{aligned}$$

This likelihood is a product of four different functions, each involving just one parameter. Thus, we can estimate each parameter independently of the rest. ■

As we saw in the last example, conditional independence can help us decouple the estimate of parameters of $P(\boldsymbol{X})$ from these of $P(O_{\boldsymbol{X}} \mid \boldsymbol{X})$. Is this a general phenomenon? To answer this question, we start with a definition.

Definition 19.3

Let $\boldsymbol{y}$ be a tuple of observations. These observations partition the variables $\boldsymbol{X}$ into two sets, the observed variables $\boldsymbol{X}_{obs}^{\boldsymbol{y}} = \{X_i : y_i \neq ?\}$ and the hidden ones $\boldsymbol{X}_{hidden}^{\boldsymbol{y}} = \{X_i : y_i = ?\}$. The values of the observed variables are determined by $\boldsymbol{y}$, while the values of the hidden variables are not.

missing at random

We say that a missing data model $P_{missing}$ is missing at random (MAR) *if for all observations $\boldsymbol{y}$ with $P_{missing}(\boldsymbol{y}) > 0$, and for all $\boldsymbol{x}_{hidden}^{\boldsymbol{y}} \in Val(\boldsymbol{X}_{hidden}^{\boldsymbol{y}})$, we have that*

$$P_{missing} \models (o_{\boldsymbol{X}} \perp \boldsymbol{x}_{hidden}^{\boldsymbol{y}} \mid \boldsymbol{x}_{obs}^{\boldsymbol{y}})$$

where $o_{\boldsymbol{X}}$ are the specific values of the observation variables given $\boldsymbol{Y}$. ■

In words, the MAR assumption requires independence between the events $o_{\boldsymbol{X}}$ and $\boldsymbol{x}_{\text{hidden}}^{\boldsymbol{y}}$ given $\boldsymbol{x}_{\text{obs}}^{\boldsymbol{y}}$. Note that this statement is written in terms of event-level conditional independence

rather than conditional independence between random variables. This generality is necessary since every instance might have a different pattern of observed variables; however, if the set of observed variables is known in advance, we can state MAR as conditional independence between random variables.

This statement implies that the observation pattern gives us no additional information about the hidden variables *given the observed variables*:

$$P_{missing}(\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}, o_{\boldsymbol{X}}) = P_{missing}(\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}).$$

Why should we require the MAR assumption? If $P_{missing}$ satisfies this assumption, then we can write

$$\begin{aligned} P_{missing}(\boldsymbol{y}) &= \sum_{\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}} \left[P(\boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}, \boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}) P_{missing}(o_{\boldsymbol{X}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}, \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}})\right] \\ &= \sum_{\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}} \left[P(\boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}, \boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}) P_{missing}(o_{\boldsymbol{X}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}})\right] \\ &= P_{missing}(o_{\boldsymbol{X}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}) \sum_{\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}} P(\boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}, \boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}) \\ &= P_{missing}(o_{\boldsymbol{X}} \mid \boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}) P(\boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}). \end{aligned}$$

The first term depends only on the parameters ψ, and the second term depends only on the parameters $\boldsymbol{\theta}$. Since we write this product for every observed instance, we can write the likelihood as a product of two likelihoods, one for the observation process and the other for the underlying distribution.

Theorem 19.1

If $P_{missing}$ satisfies MAR, then $L(\boldsymbol{\theta}, \psi : \mathcal{D})$ can be written as a product of two likelihood functions $L(\boldsymbol{\theta} : \mathcal{D})$ and $L(\psi : \mathcal{D})$.

This theorem implies that we can optimize the likelihood function in the parameters $\boldsymbol{\theta}$ of the distribution $P(\boldsymbol{X})$ independently of the exact value the observation model parameters. In other words, the MAR assumption is a license to ignore the observation model while learning parameters.

The MAR assumption is applicable in a broad range of settings, but it must be considered with care. For example, consider a sensor that measures blood pressure B but can fail to record a measurement when the patient is overweight. Obesity is a very relevant factor for blood pressure, so that the sensor failure itself is informative about the variable of interest. However, if we always have observations W of the patient's body weight and H of the height, then O_B is conditionally independent of B given W and H. As another example, consider the patient description in hospital records. If the patient does not have an X-ray result X, he probably does not suffer from broken bones. Thus, O_X gives us information about the underlying domain variables. However, assume that the patient's chart also contains a "primary complaint" variable, which was the factor used by the physician in deciding which tests to perform; in this case, the MAR assumption does hold.

In both of these cases, we see that the MAR assumption does not hold given a limited set of observed attributes, but if we expand our set of observations, we can get the MAR assumption to hold. In fact, one can show that we can always extend our model to produce one where the

MAR assumption holds (exercise 19.2). Thus, from this point onward we assume that the data satisfy the MAR assumption, and so our focus is only on the likelihood of the observed data.

However, before applying the methods described later in this chapter, we always need to consider the possible correlations between the variables and the observation variables, and possibly to expand the model so as to guarantee the MAR assumption.

19.1.3 The Likelihood Function

Throughout our discussion of learning, the likelihood function has played a major role, either on its own, or with the prior in the context of Bayesian estimation. Under the assumption of MAR, we can continue to use the likelihood function in the same roles. From now on, assume we have a network $\mathcal{G}$ over a set of variables $\boldsymbol{X}$. In general, each instance has a different set of observed variables. We will denote by $\boldsymbol{O}[m]$ and $\boldsymbol{o}[m]$ the observed variables and their values in the m'th instance, and by $\boldsymbol{H}[m]$ the missing (or hidden) variables in the m'th instance. We use $L(\boldsymbol{\theta} : \mathcal{D})$ to denote the probability of the observed variables in the data, marginalizing out the hidden variables, and ignoring the observability model:

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_{m=1}^{M} P(\boldsymbol{o}[m] \mid \boldsymbol{\theta}).$$

As usual, we use $\ell(\boldsymbol{\theta} : \mathcal{D})$ to denote the logarithm of this function.

With this definition, it might appear that the problem of learning with missing data does not differ substantially from the problem of learning with complete data. We simply use the likelihood function in exactly the same way. Although this intuition is true to some extent, the computational issues associated with the likelihood function are substantially more complex in this case.

To understand the complications, we consider a simple example on the network $\mathcal{G}_{X \to Y}$ with the edge $X \to Y$. When we have complete data, the likelihood function for this network has the following form:

$$L(\boldsymbol{\theta}_X, \boldsymbol{\theta}_{Y|x^0}, \boldsymbol{\theta}_{Y|x^1} : \mathcal{D}) = \theta_{x^1}^{M[x^1]} \theta_{x^0}^{M[x^0]} \cdot \theta_{y^1|x^0}^{M[x^0,y^1]} \theta_{y^0|x^0}^{M[x^0,y^0]} \cdot \theta_{y^1|x^1}^{M[x^1,y^1]} \theta_{y^0|x^1}^{M[x^1,y^0]}.$$

In the binary case, we can use the constraints to rewrite $\theta_{x^0} = 1 - \theta_{x^1}$, $\theta_{y^0|x^0} = 1 - \theta_{y^1|x^0}$, and $\theta_{y^0|x^1} = 1 - \theta_{y^1|x^1}$. Thus, this is a function of three parameters. For example, if we have a data set with the following sufficient statistics:

$$\begin{aligned} x^1, y^1&: \ 13 \\ x^1, y^0&: \ 16 \\ x^0, y^1&: \ 10 \\ x^0, y^0&: \ 4, \end{aligned}$$

then our likelihood function has the form:

$$\theta_{x^1}^{29}(1 - \theta_{x^1})^{14} \cdot \theta_{y^1|x^0}^{10}(1 - \theta_{y^1|x^0})^4 \cdot \theta_{y^1|x^1}^{13}(1 - \theta_{y^1|x^1})^{16}. \tag{19.1}$$

This function is well-behaved: it is log-concave, and it has a unique global maximum that has a simple analytic closed form.

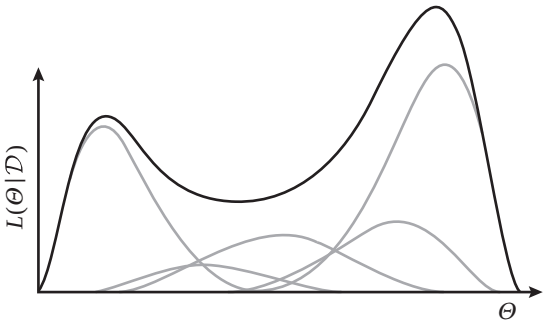


Figure 19.3 A visualization of a multimodal likelihood function with incomplete data. The data likelihood is the sum of complete data likelihoods (shown in gray lines). Each of these is unimodal, yet their sum is multimodal.

Assume that the first instance in the data set was $X[1] = x^0, Y[1] = y^1$. Now, consider a situation where, rather than observing this instance, we observed only $Y[1] = y^1$. We now have to reason that this particular data instance could have arisen in two cases: one where $X[1] = x^0$ and one where $X[1] = x^1$. In the former case, our likelihood function is as before. In the second case, we have

$$\theta_{x^1}^{30}(1-\theta_{x^1})^{13} \cdot \theta_{y^1|x^0}^{9}(1-\theta_{y^1|x^0})^{4} \cdot \theta_{y^1|x^1}^{14}(1-\theta_{y^1|x^1})^{16}. \tag{19.2}$$

When we do not observe $X[1]$, the likelihood is the marginal probability of the observations. That is, we need to sum over possible assignments to the unobserved variables. This implies that the likelihood function is the *sum* of the two complete likelihood functions of equation (19.1) and equation (19.2). Since both likelihood functions are quite similar, we can rewrite this sum as

$$\theta_{x^1}^{29}(1-\theta_{x^1})^{13} \cdot \theta_{y^1|x^0}^{9}(1-\theta_{y^1|x^0})^{4} \cdot \theta_{y^1|x^1}^{13}(1-\theta_{y^1|x^1})^{16} \left[\theta_{x^1}\theta_{y^1|x^1} + (1-\theta_{x^1})\theta_{y^1|x^0}\right].$$

This form seems quite nice, except for the last sum, which couples the parameter θ_{x^1} with $\theta_{y^1|x^1}$ and $\theta_{y^1|x^0}$.

If we have more missing values, there are other cases we have to worry about. For example, if $X[2]$ is also unobserved, we have to consider all possible combinations for $X[1]$ *and* $X[2]$. This results in a sum over four terms similar to equation (19.1), each one with different counts. In general, the likelihood function with incomplete data is the sum of likelihood functions, one for each possible *joint* assignment of the missing values. Note that the number of possible assignments is exponential in the total number of missing values.

We can think of the situation using a geometric intuition. Each one of the complete data likelihood defines a unimodal function. Their sum, however, can be multimodal. In the worst case, the likelihood of each of the possible assignments to the missing values contributes to a different peak in the likelihood function. The total likelihood function can therefore be quite complex. It takes the form of a "mixture of peaks," as illustrated pictorially in figure 19.3.

parameter independence

likelihood decomposability

To make matters even more complicated, we lose the property of *parameter independence*, and thereby the *decomposability* of the likelihood function. Again, we can understand this phenomenon either qualitatively, from the perspective of graphical models, or quantitatively, by

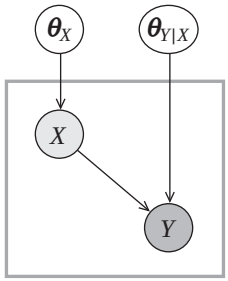


Figure 19.4 The meta-network for parameter estimation for $X \to Y$. When $X[m]$ is hidden but $Y[m]$ is observed, the trail $\theta_X \to X[m] \to Y[m] \leftarrow \theta_{Y|X}$ is active. Thus, the parameters are not independent in the posterior distribution.

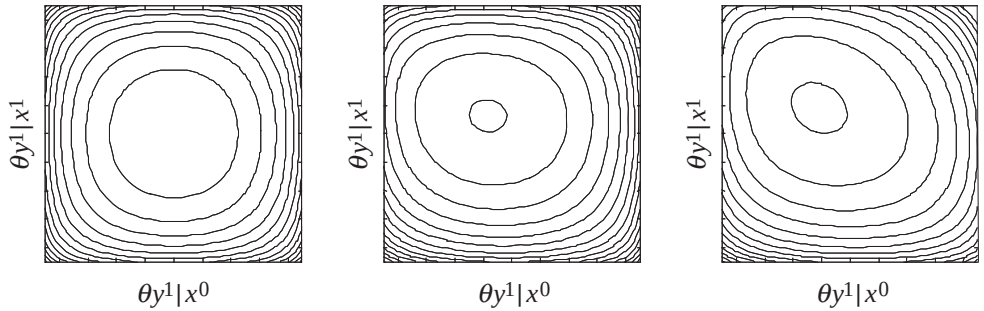


Figure 19.5 Contour plots for the likelihood function for the network $X \to Y$, over the parameters $\theta_{y^1|x^0}$ and $\theta_{y^1|x^1}$. The total number of data points is 8. (a) No X values are missing. (b) Two X values missing. (c) Three X values missing.

looking at the likelihood function. Qualitatively, recall from section 17.4.2 that, in the complete data case, $\boldsymbol{\theta}_{Y|x^1}$ and $\boldsymbol{\theta}_{Y|x^0}$ are independent given the data, because they are independent given $Y[m]$ and $X[m]$. But if $X[m]$ is unobserved, they are clearly dependent. This fact is clearly illustrated by the meta-network (as in figure 17.7) that represents the learning problem. For example, in a simple network over two variables $X \to Y$, we see that missing data can couple the two parameters' variables; see figure 19.4.

We can also see this phenomenon numerically. Assume for simplicity that $\boldsymbol{\theta}_X$ is known. Then, our likelihood is a function of two parameters $\boldsymbol{\theta}_{y^1|x^1}$ and $\boldsymbol{\theta}_{y^1|x^0}$. Intuitively, if our missing $X[1]$ is H, then it cannot be T. Thus, the likelihood functions of the two parameters are correlated; the more missing data we have, the stronger the correlation. This phenomenon is shown in figure 19.5.

local decomposability

global decomposability

This example shows that we have lost the *local decomposability* property in estimating the CPD $P(Y \mid X)$. What about *global decomposability* between different CPDs? Consider a simple model where there is one hidden variable H, and two observed variables X and Y, and edges $H \rightarrow X$ and $H \rightarrow Y$. Thus, the probability of observing the values x and y is

$$P(x, y) = \sum_h P(h)P(x \mid h)P(y \mid h).$$

The likelihood function is a product of such terms, one for each observed instance $x[m], y[m]$, and thus has the form

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_{x,y} \left(\sum_h P(h)P(x \mid h)P(y \mid h) \right)^{M[x,y]}.$$

When we had complete data, we rewrote the likelihood function as a product of local likelihood functions, one for each CPD. This decomposition was crucial for estimating each CPD independently of the others. In this example, we see that the likelihood is a product of sum of products of terms involving different CPDs. The interleaving of products and sums means that we cannot write the likelihood as a product of local likelihood functions. Again, this result is intuitive: Because we do not observe the variable H, we cannot decouple the estimation of $P(X \mid H)$ from that of $P(Y \mid H)$. Roughly speaking, both estimates depend on how we "reconstruct" H in each instance.

We now consider the general case. Assume we have a network $\mathcal{G}$ over a set of variables $\boldsymbol{X}$. In general, each instance has a different set of observed variables. We use $\mathcal{D}$ to denote, as before, the actual observed data values; we use $\mathcal{H} = \cup_m \boldsymbol{h}[m]$ to denote a possible assignment to all of the missing values in the data set. Thus, the pair $(\mathcal{D}, \mathcal{H})$ defines an assignment to all of the variables in all of our instances.

The likelihood function is

$$L(\boldsymbol{\theta} : \mathcal{D}) = P(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{\mathcal{H}} P(\mathcal{D}, \mathcal{H} \mid \boldsymbol{\theta}).$$

Unfortunately, the number of possible assignments in this sum is exponential in the number of missing values in the entire data set. Thus, although each of the terms $P(\mathcal{D}, \mathcal{H} \mid \boldsymbol{\theta})$ is a unimodal distribution, the sum can have, in the worst case, an exponential number of modes.

However, unimodality is not the only property we lose. Recall that our likelihood function in the complete data case was compactly represented as a product of local terms. This property was important both for the analysis of the likelihood function and for the task of evaluating the likelihood function. What about the incomplete data likelihood? If we use a straightforward representation, we get an exponential sum of terms, which is clearly not useful. Can we use additional properties of the data to help in representing the likelihood? Recall that we assume that different instances are independent of each other. This allows us to write the likelihood function as a product over the probability of each partial instance.

Proposition 19.1 *Assuming IID data, the likelihood can be written as*

$$L(\boldsymbol{\theta} : \mathcal{D}) = \prod_m P(\boldsymbol{o}[m] \mid \boldsymbol{\theta}) = \prod_m \sum_{\boldsymbol{h}[m]} P(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}).$$

This proposition shows that, to compute the likelihood function, we need to perform inference for each instance, computing the probability of the observations. As we discussed in section 9.1, this problem can be intractable, depending on the network structure and the pattern of missing values. Thus, for some learning problems, even the task of evaluating likelihood function for a particular choice of parameters is a difficult computational problem. This observation suggests that optimizing the choice of parameters for such networks can be computationally challenging.

To conclude, **in the presence of partially observed data, we have lost all of the important properties of our likelihood function: its unimodality, its closed-form representation, and the decomposition as a product of likelihoods for the different parameters. Without these properties, the learning problem becomes substantially more complex.**

19.1.4 Identifiability

Another issue that arises in the context of missing data is our ability to identify uniquely a model from the data.

Example 19.6 *Consider again our thumbtack tossing experiments. Suppose the experimenter can randomly choose to toss one of two thumbtacks (say from two different brands). Due to a miscommunication between the statistician and the experimenter, only the toss outcomes were recorded, but not the brand of thumbtack used.*

To model the experiment, we assume that there is a hidden variable H, so that if $H = h^1$, the experimenter tossed the first thumbtack, and if $H = h^2$, the experimenter tossed the second thumbtack. The parameters of our model are θ_H, $\theta_{X|h^1}$, and $\theta_{X|h^2}$, denoting the probability of choosing the first thumbtack, and the probability of heads in each thumbtack. This setting satisfies MCAR (since H is hidden). It is straightforward to write the likelihood function:

$$L(\boldsymbol{\theta} : \mathcal{D}) = P(x^1)^{M[1]}(1 - P(x^1))^{M[0]},$$

where

$$P(x^1) = \theta_H \theta_{X|h^1} + (1 - \theta_H)\theta_{X|h^2}.$$

If we examine this term, we see that $P(x^1)$ is the weighted average of $\theta_{X|h^1}$ and $\theta_{X|h^2}$. There are multiple choices of these two parameters and θ_H that achieve the same value of $P(x^1)$. For example, $\theta_H = 0.5$,$\theta_{X|h^1} = 0.5$,$\theta_{X|h^2} = 0.5$ leads to the same behavior as $\theta_H = 0.5$,$\theta_{X|h^1} = 0.8$,$\theta_{X|h^2} = 0.2$. Because the likelihood of the data is a function only of $P(x^1)$, we conclude that there is a continuum of parameter choices that achieve the maximum likelihood. ■

This example illustrates a situation where the learning problem is underconstrained: Given the observations, we cannot hope to recover a unique set of parameters. Recall that in previous sections, we showed that our estimates are consistent and thus will approach the true parameters

when sufficient data are available. In this example, we cannot hope that more data will let us recover the true parameters.

Before formally treating the issue, let us examine another example that does not involve hidden variables.

Example 19.7

Suppose we conduct an experiment where we toss two coins X and Y that may be correlated with each other. After each toss, one of the coins is hidden from us using a mechanism that is totally unrelated to the outcome of the coins. Clearly, if we have sufficient observations (that is, the mechanism does not hide one of the coins consistently), then we can estimate the marginal probability of each of the coins. Can we, however, learn anything about how they depend on each other? Consider some pair of marginal probabilities $P(X)$ and $P(Y)$; because we never get to observe both coins together, any joint distribution that has these marginals has the same likelihood. In particular, a model where the two coins are independent achieves maximum likelihood but is not the unique point. In fact, in some cases a model where one is a deterministic function of the other also achieves the same likelihood (for example, if we have the same frequency of observed X heads as of observed Y heads). ■

These two examples show that in some learning situations we cannot resolve all aspects of the model by learning from data. This issue has been examined extensively in statistics, and is known as *identifiability*, and we briefly review the relevant notions here.

identifiability

Definition 19.4 identifiability

Suppose we have a parametric model with parameters $\boldsymbol{\theta} \in \Theta$ that defines a distribution $P(\boldsymbol{X} \mid \boldsymbol{\theta})$ over a set $\boldsymbol{X}$ of measurable variables. A choice of parameters $\boldsymbol{\theta}$ is identifiable *if there is no $\boldsymbol{\theta}' \neq \boldsymbol{\theta}$ such that $P(\boldsymbol{X} \mid \boldsymbol{\theta}) = P(\boldsymbol{X} \mid \boldsymbol{\theta}')$. A model is* identifiable *if all parameter choices $\boldsymbol{\theta} \in \Theta$ are identifiable.* ■

In other words, a model is identifiable if each choice of parameters implies a different distribution over the observed variables. Nonidentifiability implies that there are parameter settings that are indistinguishable given the data, and therefore cannot be identified from the data. Usually this is a sign that the parameterization is redundant with respect to the actual observations. For example, the model we discuss in example 19.6 is unidentifiable, since there are regions in the parameters space that induce the same probability on the observations.

Another source of nonidentifiability is hidden variables.

Example 19.8

Consider now a different experiment where we toss two thumbtacks from two different brands: Acme (A) and Bond (B). In each round, both thumbtacks are tossed and the entries are recorded. Unfortunately, due to scheduling constraints, two different experimenters participated in the experiment; each used a slightly different hand motion, changing the probability of heads and tails. Unfortunately, the experimenter name was not recorded, and thus we only have measurements of the outcome in each experiment.

To model this situation, we have three random variables to describe each round. Suppose A denotes the outcome of the toss of the Acme thumbtack and B the outcome of the toss of the Bond thumbtack. Because these outcomes depend on the experimenter, we add another (hidden) variable H that denotes the name of the experimenter. We assume that the model is such that A and B are independent given H. Thus,

$$P(A, B) = \sum_h P(h)P(A \mid h)P(B \mid h).$$

Because we never observe H, the parameters of this model can be reshuffled by "renaming" the values of the hidden variable. If we exchange the roles of h^0 and h^1, and change the corresponding entries in the CPDs, we get a model with exactly the same likelihood, but with different parameters. In this case, the likelihood surface is duplicated. For each parameterization, there is an equivalent parameterization by exchanging the names of the hidden variable. We conclude that this model is not identifiable. ■

This type of unidentifiability exists in any model where we have hidden variables we never observe. When we have several hidden variables, the problem is even worse, and the number of equivalent "reflections" of each solution is exponential in the number of hidden variables.

Although such a model is not identifiable due to "renaming" transformations, it is in some sense better than the model of example 19.6, where we had an entire region of equivalent parameterizations. To capture this distinction, we can define a weaker version of identifiability.

Definition 19.5 locally identifiable

Suppose we have a parametric model with parameters $\boldsymbol{\theta} \in \Theta$ that defines a distribution $P(\boldsymbol{X} \mid \boldsymbol{\theta})$ over a set $\boldsymbol{X}$ of measurable variables. A choice of parameters $\boldsymbol{\theta}$ is locally identifiable *if there is a constant $\epsilon > 0$ such that there is no $\boldsymbol{\theta}' \neq \boldsymbol{\theta}$ such that $\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2 < \epsilon$ and $P(\boldsymbol{X} \mid \boldsymbol{\theta}) = P(\boldsymbol{X} \mid \boldsymbol{\theta}')$. A model is* locally identifiable *if all parameter choices $\boldsymbol{\theta} \in \Theta$ are locally identifiable.* ■

In other words, a model is locally identifiable if each choice of parameters defines a distribution that is different than the distribution of neighboring parameterization in a sufficiently small neighborhood. This definition implies that, from a local perspective, the model is identifiable. The model of example 19.8 is locally identifiable, while the model of example 19.6 is not.

It is interesting to note that we have encountered similar issues before: As we discussed in chapter 18, our data do not allow us to distinguish between structures in the same I-equivalence class. This limitation did not prevent us from trying to learn a model from data, but we needed to avoid ascribing meaning to directionality of edges that are not consistent throughout the I-equivalence class. The same approach holds for unidentifiability due to missing data: **A nonidentifiable model does not mean that we should not attempt to learn models from data. But it does mean that we should be careful not to read into the learned model more than what can be distinguished given the available data.**

19.2 Parameter Estimation

As for the fully observable case, we first consider the parameter estimation task. As with complete data, we consider two approaches to estimation, maximum likelihood estimation (MLE), and Bayesian estimation. We start with a discussion of methods for MLE estimation, and then consider the Bayesian estimation problem in the next section.

More precisely, suppose we are given a network structure $\mathcal{G}$ and the form of the CPDs. Thus, we only need to set the parameters $\boldsymbol{\theta}$ to define a distribution $P(\mathcal{X} \mid \boldsymbol{\theta})$. We are also given a data set $\mathcal{D}$ that consists of M partial instances to $\mathcal{X}$. We want to find the values $\hat{\boldsymbol{\theta}}$ that maximize the log-likelihood function: $\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta} : \mathcal{D})$. As we discussed, in the presence of incomplete data, the likelihood does not decompose. And so the problem requires optimizing a highly nonlinear and multimodal function over a high-dimensional space (one consisting of parameter assignments to all CPDs). There are two main classes of methods for performing

this optimization: a generic nonconvex optimization algorithm, such as gradient ascent; and *expectation maximization*, a more specialized approach for optimizing likelihood functions.

19.2.1 Gradient Ascent

gradient ascent

One approach to handle this optimization task is to apply some variant of *gradient ascent*, a standard function-optimization technique applied to the likelihood function (see appendix A.5.2). These algorithms are generic and can be applied if we can evaluate the gradient function at different parameter choices.

19.2.1.1 Computing the Gradient

The main technical question we need to tackle is how to compute the gradient. We begin with considering the derivative relative to a single CPD entry $P(x \mid \boldsymbol{u})$. We can then use this result as the basis for computing derivatives relative to other parameters, which arise when we have structured CPDs.

Lemma 19.1

Let $\mathcal{B}$ be a Bayesian network with structure $\mathcal{G}$ over $\mathcal{X}$ that induces a probability distribution P, let $\boldsymbol{o}$ be a tuple of obserations for some of the variables, and let $X \in \mathcal{X}$ be some random variable. Then

$$\frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\boldsymbol{o}) = \frac{1}{P(x \mid \boldsymbol{u})} P(x, \boldsymbol{u}, \boldsymbol{o})$$

if $P(x \mid \boldsymbol{u}) > 0$, where $x \in Val(X)$, $\boldsymbol{u} \in Val(\mathrm{Pa}_X)$.

Proof We start by considering the case where the evidence is a full assignment ξ to all variables. The probability of such an assignment is a product of the relevant CPD entries. Thus, the gradient of this product with respect to the parameter $P(x \mid \boldsymbol{u})$ is simply

$$\frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\xi) = \begin{cases} \frac{1}{P(x|\boldsymbol{u})} P(\xi) & \text{if } \xi\langle X, \mathrm{Pa}_X\rangle = \langle x, \boldsymbol{u}\rangle \\ 0 & \text{otherwise.} \end{cases}$$

We now consider the general case where the evidence is a partial assignment. As usual, we can write $P(\boldsymbol{o})$ as a sum over all full assignments consistent with $P(\boldsymbol{o})$

$$P(\boldsymbol{o}) = \sum_{\xi : \xi\langle \boldsymbol{O}\rangle = \boldsymbol{o}} P(\xi).$$

Applying the differentiation formula to each of these full assignments, we get

$$\begin{aligned} \frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\boldsymbol{o}) &= \sum_{\xi : \xi\langle \boldsymbol{O}\rangle = \boldsymbol{o}} \frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\xi) \\ &= \sum_{\xi : \xi\langle \boldsymbol{O}\rangle = \boldsymbol{o}, \xi\langle X, \mathrm{Pa}_X\rangle = \langle x, \boldsymbol{u}\rangle} \frac{1}{P(x \mid \boldsymbol{u})} P(\xi) \\ &= \frac{1}{P(x \mid \boldsymbol{u})} P(x, \boldsymbol{u}, \boldsymbol{o}). \end{aligned}$$

■