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

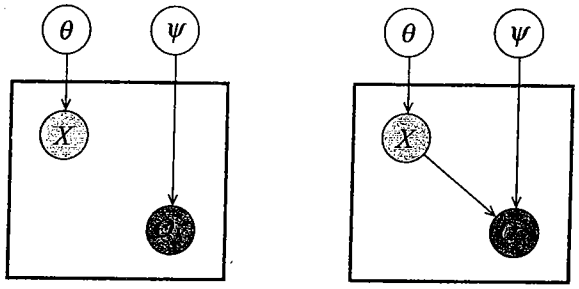
We flip a thumbtack onto a table, and every now and then it rolls off the table. Since a fall from the table to the floor is quite different from our desired experimental setup, we do not use results from these flips (they are missing). How would that change our estimation? The simple solution is to 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 ψ.*

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_X 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

$$\begin{aligned} &L(\theta, \psi_{O_X|x^1}, \psi_{O_X|x_1^0} : \mathcal{D}) \\ &\quad = \theta^{M[1]}(1-\theta)^{M[0]}\psi_{O_X|x^1}^{M[1]}\psi_{O_X|x^0}^{M[0]} \\ &\qquad (\theta(1-\psi_{O_X|x^1}) + (1-\theta)(1-\psi_{O_X|x_1^0}))^{M[?]}. \end{aligned}$$

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. ■

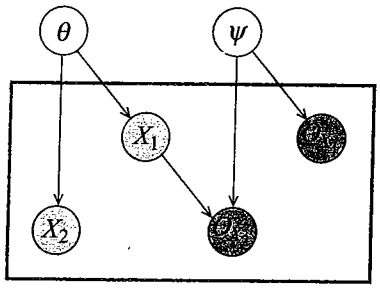


Figure 19.2 An example satisfying MAR but not MCAR. Here, the observability pattern depends on the value of underlying variables.

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 $\psi_{O_{X_2}|x_2^1}$ and $\psi_{O_{X_2}|x_2^0}$ 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 $\theta_{X_1}(1 - \theta_{X_2})\psi_{O_{X_2}|x_2^1}$.

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 $\theta_{X_1}(1 - \psi_{O_{X_1}|x_1^1})$.

If we write all six possible cases and then rearrange the products, we see that we can write the likelihood function as

$$\begin{aligned} L(\boldsymbol{\theta} : \mathcal{D}) \quad = \quad & \theta_{X_1}^{M[y_1^1]}(1 - \theta_{X_1})^{M[y_1^0]} \\ & \theta_{X_2}^{M[y_2^1]}(1 - \theta_{X_2})^{M[y_2^0]} \\ & \psi_{O_{X_2}|x_2^1}^{M[y_1^1, y_2^1] + M[y_1^1, y_2^0]}(1 - \psi_{O_{X_2}|x_2^1})^{M[y_1^1, y_2^1]} \\ & \psi_{O_{X_2}|x_2^0}^{M[y_1^0, y_2^1] + M[y_1^0, y_2^0]}(1 - \psi_{O_{X_2}|x_2^0})^{M[y_1^0, y_2^1]}. \end{aligned}$$

This likelihood is a product of a 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 $\boldsymbol{X}^{\boldsymbol{y}}_{obs} = \{X_i : y_i \neq ?\}$ and the hidden ones $\boldsymbol{X}^{\boldsymbol{y}}_{hidden} = \{X_i : y_i = ?\}$. 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 $\boldsymbol{x}^{\boldsymbol{y}}_{hidden} \in Val(\boldsymbol{X}^{\boldsymbol{y}}_{hidden})$, we have that*

$$P_{missing} \models (o_{\boldsymbol{X}} \perp \boldsymbol{x}^{\boldsymbol{y}}_{hidden} \mid \boldsymbol{x}^{\boldsymbol{y}}_{obs})$$

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 $\boldsymbol{x}^{\boldsymbol{y}}_{\text{hidden}}$ given $\boldsymbol{x}^{\boldsymbol{y}}_{\text{obs}}$. 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 θ. 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 $L(\theta, \psi : \mathcal{D})$ can be written as a product of two likelihood functions $L(\theta : \mathcal{D})$ and $L(\psi : \mathcal{D})$.

This theorem implies that we can optimize the likelihood function in the parameters θ 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:

x^1, y^1: 13
x^1, y^0: 16
x^0, y^1: 10
x^0, y^0: 4,

then our likelihood function has the form:

$$\theta_{x^1}^{29}(1 - \theta_{x^1})^{14} \cdot \theta_{y^1|x^0}^{10}(1 - \theta_{y^1|x^0})^4 \cdot \theta_{y^1|x^1}^{13}(1 - \theta_{y^1|x^1})^{16}. \tag{19.1}$$

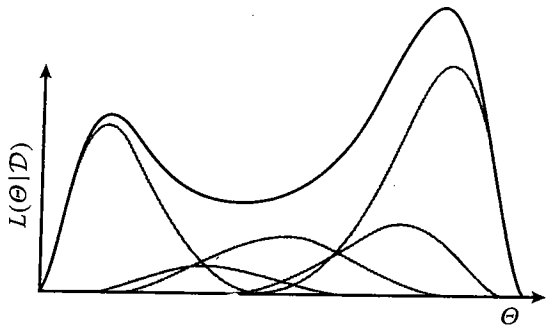


Figure 19.3 A visualization of a multimodal likelihood function with incomplete data. The data likelihood is the sum of complete data likelihoods (shown in gray lines). Each of these is unimodal, yet their sum is multimodal.

This function is well-behaved: it is log-concave, and it has a unique global maximum that has a simple analytic closed form.

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

$$\theta_{x^1}^{29}(1-\theta_{x^1})^{13} \cdot \theta_{y^1|x^0}^{10}(1-\theta_{y^1|x^0})^{4} \cdot \theta_{y^1|x^1}^{12}(1-\theta_{y^1|x^1})^{16}\left[\theta_{x^1}\theta_{y^1|x^1} + (1-\theta_{x^1})\theta_{y^1|x^0}\right].$$

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

To make matters even more complicated, we lose the property of *parameter independence*,

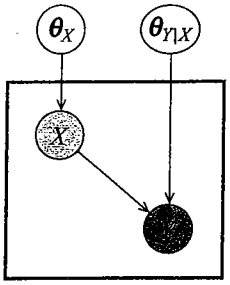


Figure 19.4 The meta-network for parameter estimation for $X \rightarrow Y$. When $X[m]$ is hidden but $Y[m]$ is observed, the trail $\theta_X \rightarrow X[m] \rightarrow Y[m] \leftarrow \theta_{Y|X}$ is active. Thus, thus the parameters are not independent in the posterior distribution.

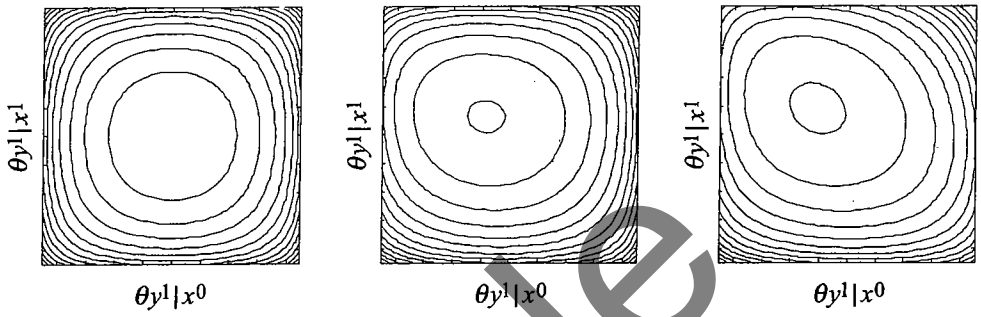


Figure 19.5 Contour plots for the likelihood function for the network $X \rightarrow Y$, over the parameters $\theta_{y^1|x^0}$ and $\theta_{y^1|x^1}$. The total number of data points is 8. (a) No X values are missing. (b) Two X values missing. (c) Three X values missing.

likelihood decomposability

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 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 $X \rightarrow Y$, 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

$$L(\theta : \mathcal{D}) = \prod_{x,y} \left(\sum_h P(h)P(x \mid h)P(y \mid h) \right)^{M[x,y]}.$$

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 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 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 $\mathcal{D}$ 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

$$L(\theta : \mathcal{D}) = P(\mathcal{D} \mid \theta) = \sum_{\mathcal{H}} P(\mathcal{D}, \mathcal{H} \mid \theta).$$

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 $P(\mathcal{D}, \mathcal{H} \mid \theta)$ 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*

$$L(\theta : \mathcal{D}) = \prod_m P(o[m] \mid \theta) = \prod_m \sum_{h[m]} P(o[m], h[m] \mid \theta).$$

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 kikelihood 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:

$$L(\theta : \mathcal{D}) = P(x^1)^{M[1]}(1 - P(x^1))^{M[0]},$$

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 arises in from 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 $\theta \in \Theta$ that defines a distribution $P(\boldsymbol{X} \mid \theta)$ over a set $\boldsymbol{X}$ of measurable variables. A choice of parameters θ is locally identifiable *if there is a constant $\epsilon > 0$ such that there is no $\theta' \neq \theta$ such that $\|\theta - \theta'\|_2 < \epsilon$ and $P(\boldsymbol{X} \mid \theta) = P(\boldsymbol{X} \mid \theta')$. A model is* locally identifiable *if all parameter choices $\theta \in \Theta$ 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 θ to define a distribution $P(\mathcal{X} \mid \theta)$. We are also given a data set $\mathcal{D}$ that consists of M partial instances to $\mathcal{X}$. We want to find the values $\hat{\theta}$ that maximize the log-likelihood function: $\hat{\theta} = \arg\max_\theta \ell(\theta : \mathcal{D})$. 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 $P(\boldsymbol{o})$

$$P(\boldsymbol{o}) = \sum_{\xi : \xi\langle E\rangle = \boldsymbol{o}} P(\xi).$$

Applying the differentiation formula to each of these full assignments, we get

$$\begin{aligned} \frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\boldsymbol{o}) &= \sum_{\xi : \xi\langle \boldsymbol{O}\rangle = \boldsymbol{o}} \frac{\partial}{\partial P(x \mid \boldsymbol{u})} P(\xi) \\ &= \sum_{\xi : \xi\langle \boldsymbol{O}\rangle = \boldsymbol{o}, \xi\langle X, \mathrm{Pa}_X\rangle = \langle x, \boldsymbol{u}\rangle} \frac{1}{P(x \mid \boldsymbol{u})} P(\xi) \\ &= \frac{1}{P(x \mid \boldsymbol{u})} P(x, \boldsymbol{u}, \boldsymbol{o}). \end{aligned}$$

■

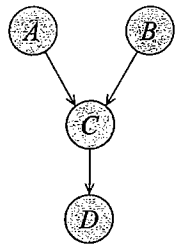


Figure 19.6 A simple network used to illustrate learning algorithms for missing data

When $\boldsymbol{o}$ is inconsistent with x or $\boldsymbol{u}$, then the gradient is 0, since the probability $P(x, \boldsymbol{u}, \boldsymbol{o})$ is 0 in this case. When $\boldsymbol{o}$ is consistent with x and $\boldsymbol{u}$, the gradient is the ratio between the probability $P(x, \boldsymbol{u}, \boldsymbol{o})$ and the parameter $P(x \mid \boldsymbol{u})$. Intuitively, this ratio takes into account the weight of the cases where $P(x \mid \boldsymbol{u})$ is "used" in the computation of $P(\boldsymbol{o})$. Increasing $P(x \mid \boldsymbol{u})$ by a small amount will increase the probability of these cases by a multiplicative factor.

The lemma does not deal with the case where $P(x \mid \boldsymbol{u}) = 0$, since we cannot divide by 0. Note, however, that the proof shows that this division is mainly a neat manner of writing the product of all terms *except* $P(x \mid \boldsymbol{u})$. Thus, even in this extreme case we can use a similar proof to compute the gradient, although writing the term explicitly is less elegant. Since in learning we usually try to avoid extreme parameter assignments, we will continue our discussion with the assumption that $P(x \mid \boldsymbol{u}) > 0$.

An immediate consequence of lemma 19.1 is the form of the gradient of the log-likelihood function.

Theorem 19.2 *Let $\mathcal{G}$ be a Bayesian network structure over $\mathcal{X}$, and let $\mathcal{D} = \{\boldsymbol{o}[1], \ldots, \boldsymbol{o}[M]\}$ be a partially observable data set. Let X be a variable and $\boldsymbol{U}$ its parents in $\mathcal{G}$. Then*

$$\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(x \mid \boldsymbol{u})} = \frac{1}{P(x \mid \boldsymbol{u})} \sum_{m=1}^{M} P(x, \boldsymbol{u} \mid \boldsymbol{o}[m], \boldsymbol{\theta}).$$

The proof is left as an exercise (exercise 19.5).

This theorem provides the form of the gradient for table-CPDs. For other CPDs, such as noisy-or CPDs, we can use the *chain rule of derivatives* to compute the gradient. Suppose that the CPD entries of $P(X \mid \boldsymbol{U})$ are written as functions of some set of parameters $\boldsymbol{\theta}$. Then, for a specific parameter $\theta \in \boldsymbol{\theta}$, we have

chain rule of derivatives

$$\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial \theta} = \sum_{x,\boldsymbol{u}} \frac{\partial \ell(P_{\boldsymbol{\theta}} : \mathcal{D})}{\partial P(x \mid \boldsymbol{u})} \frac{\partial P(x \mid \boldsymbol{u})}{\partial \theta},$$

where the first term is the derivative of the log-likelihood function when parameterized in terms of the table-CPDs induced by $\boldsymbol{\theta}$. For structured CPDs, we can use this formula to compute the gradient with respect to the CPD parameters. For some CPDs, however, this may not be the most efficient way of computing these gradients; see exercise 19.4.

19.2.1.2 An Example

We consider a simple example to clarify the concept. Consider the network shown in figure 19.6, and a partially specified data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0 \rangle$.

We want to compute the gradient of one family of parameters $P(D \mid c^0)$ given the observation $\boldsymbol{o}$. Using theorem 19.2, we know that

$$\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^0)} = \frac{P(d^0, c^0 \mid \boldsymbol{o})}{P(d^0 \mid c^0)},$$

and similarly for other values of D and C.

Assume that our current $\boldsymbol{\theta}$ is:

$$\begin{aligned}
\boldsymbol{\theta}_{a^1} &= 0.3 \\
\boldsymbol{\theta}_{b^1} &= 0.9 \\
\boldsymbol{\theta}_{c^1 \mid a^0, b^0} &= 0.83 \\
\boldsymbol{\theta}_{c^1 \mid a^0, b^1} &= 0.09 \\
\boldsymbol{\theta}_{c^1 \mid a^1, b^0} &= 0.6 \\
\boldsymbol{\theta}_{c^1 \mid a^1, b^1} &= 0.2 \\
\boldsymbol{\theta}_{d^1 \mid c^0} &= 0.1 \\
\boldsymbol{\theta}_{d^1 \mid c^1} &= 0.8.
\end{aligned}$$

In this case, the probabilities of the four data cases that are consistent with $\boldsymbol{o}$ are

$$\begin{aligned}
P(\langle a^1, b^1, c^1, d^0 \rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.2 = 0.0108 \\
P(\langle a^1, b^1, c^0, d^0 \rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.9 = 0.1944 \\
P(\langle a^1, b^0, c^1, d^0 \rangle) &= 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.2 = 0.0036 \\
P(\langle a^1, b^0, c^0, d^0 \rangle) &= 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.9 = 0.0108.
\end{aligned}$$

To compute the *posterior* probability of these instances given the partial observation $\boldsymbol{o}$, we divide the probability of each instance with the total probability, which is 0.2196, that is,

$$\begin{aligned}
P(\langle a^1, b^1, c^1, d^0 \rangle \mid \boldsymbol{o}) &= 0.0492 \\
P(\langle a^1, b^1, c^0, d^0 \rangle \mid \boldsymbol{o}) &= 0.8852 \\
P(\langle a^1, b^0, c^1, d^0 \rangle \mid \boldsymbol{o}) &= 0.0164 \\
P(\langle a^1, b^0, c^0, d^0 \rangle \mid \boldsymbol{o}) &= 0.0492.
\end{aligned}$$

Using these computations, we see that

$$\begin{aligned}
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^1 \mid c^0)} &= \frac{P(d^1, c^0 \mid \boldsymbol{o})}{P(d^1 \mid c^0)} = \frac{0}{0.1} = 0 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^0)} &= \frac{P(d^0, c^0 \mid \boldsymbol{o})}{P(d^0 \mid c^0)} = \frac{0.8852 + 0.0492}{0.9} = 1.0382 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^1 \mid c^1)} &= \frac{P(d^1, c^1 \mid \boldsymbol{o})}{P(d^1 \mid c^1)} = \frac{0}{0.8} = 0 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^1)} &= \frac{P(d^0, c^1 \mid \boldsymbol{o})}{P(d^0 \mid c^1)} = \frac{0.0492 + 0.0164}{0.2} = 0.328.
\end{aligned}$$

These computations show that we can increase the probability of the observations $\boldsymbol{o}$ by either increasing $P(d^0 \mid c^0)$ or $P(d^0 \mid c^1)$. Moreover, increasing the former parameter will lead to a bigger change in the probability of $\boldsymbol{o}$ than a similar increase in the latter parameter.

Now suppose we have an observation $\boldsymbol{o}' = \langle a^0, ?, ?, d^1 \rangle$. We can repeat the same computation as before and see that

$$\begin{aligned}
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^1 \mid c^0)} &= \frac{P(d^1, c^0 \mid \boldsymbol{o}')}{P(d^1 \mid c^0)} = \frac{0.2836}{0.1} = 2.8358 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^0 \mid c^0)} &= \frac{P(d^0, c^0 \mid \boldsymbol{o}')}{P(d^0 \mid c^0)} = \frac{0}{0.9} = 0 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^1 \mid c^1)} &= \frac{P(d^1, c^1 \mid \boldsymbol{o}')}{P(d^1 \mid c^1)} = \frac{0.7164}{0.8} = 0.8955 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^0 \mid c^1)} &= \frac{P(d^0, c^1 \mid \boldsymbol{o}')}{P(d^0 \mid c^1)} = \frac{0}{0.2} = 0.
\end{aligned}$$

Suppose our data set consists only of these two instances. The gradient of the log-likelihood function is the sum of the gradient with respect to the two instances. We get that

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} &= 2.8358 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} &= 1.0382 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^1)} &= 0.8955 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^1)} &= 0.328.
\end{aligned}$$

Note that all the gradients are nonnegative. Thus, increasing any of the parameters in the CPD $P(D \mid C)$ will increase the likelihood of the data. It is clear, however, that we cannot increase both $P(d^1 \mid c^0)$ and $P(d^0 \mid c^0)$ at the same time, since this will lead to an illegal conditional probability. One way of solving this is to use a single parameter $\theta_{d^1 \mid c^0}$ and write

$$P(d^1 \mid c^0) = \theta_{d^1 \mid c^0} \quad P(d^0 \mid c^0) = 1 - \theta_{d^1 \mid c^0}.$$

Using the chain rule of conditional probabilities, we have that

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial \theta_{d^1 \mid c^0}} &= \frac{\partial P(d^1 \mid c^0)}{\partial \theta_{d^1 \mid c^0}} \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} + \frac{\partial P(d^0 \mid c^0)}{\partial \theta_{d^1 \mid c^0}} \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} \\
&= \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} - \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} \\
&= 2.8358 - 1.0382 = 1.7976.
\end{aligned}$$

Thus, in this case, we prefer to increase $P(d^1 \mid c^0)$ and decrease $P(d^0 \mid c^0)$, since the resulting increase in the probability of $\boldsymbol{o}'$ will be larger than the decrease in the probability of $\boldsymbol{o}$.

Algorithm 19.1 Computing the gradient in a network with table-CPDs

Procedure Compute-Gradient (
  $\mathcal{G}$, // Bayesian network structure over $X_1, \ldots, X_n$
  $\boldsymbol{\theta}$, // Set of parameters for $\mathcal{G}$
  $\mathcal{D}$ // Partially observed data set
)

```
1    // Initialize data structures
2    for each i = 1, ..., n
3      for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
4        M̄[x_i, u_i] ← 0
5    // Collect probabilities from all instances
6    for each m = 1 ... M
7      Run clique tree calibration on ⟨G, θ⟩ using evidence o[m]
8      for each i = 1, ..., n
9        for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
10         M̄[x_i, u_i] ← M̄[x_i, u_i] + P(x_i, u_i | o[m])
11   // Compute components of the gradient vector
12   for each i = 1, ..., n
13     for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
14       δ_{x_i|u_i} ← (1 / θ_{x_i|u_i}) M̄[x_i, u_i]
15   return {δ_{x_i,|u_i} : ∀i = 1, ..., n, ∀(x_i, u_i) ∈ Val(X_i, Pa_{X_i}^G)}
```

19.2.1.3 Gradient Ascent Algorithm

We now generalize these ideas to case of an arbitrary network. For now we focus on the case of table-CPDs. In this case, the gradient is given by theorem 19.2. To compute the gradient for the CPD $P(X \mid \boldsymbol{U})$, we need to compute the joint probability of x and $\boldsymbol{u}$ relative to our current parameter setting $\boldsymbol{\theta}$ and each observed instance $\boldsymbol{x}[m]$. In other words, we need to compute the joint distribution $P(X[m], \boldsymbol{U}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$ for each m. We can do this by running an inference procedure for each data case. Importantly, **we can do all of the required inference for each data case using one clique tree calibration, since the family preservation property guarantees that X and its parents $\boldsymbol{U}$ will be together in some clique in the tree.** Procedure Compute-Gradient, shown in algorithm 19.1, performs these computations.

Once we have a procedure for computing the gradient, it seems that we can simply plug it into a standard package for gradient ascent and optimize the parameters. As we have illustrated, however, there is one issue that we need to deal with. It is not hard to confirm that all components of the gradient vector are nonnegative. This is natural, since increasing each of the parameters will lead to higher likelihood. Thus, a step in the gradient direction will *increase* all the parameters. Remember, however, that we want to ensure that our parameters describe a legal probability distribution. That is, the parameters for each conditional probability are nonnegative and sum to one.

In the preceding example, we saw one approach that works well when we have binary variables. In general networks, there are two common approaches to deal with this issue. The first approach is to modify the gradient ascent procedure we use (for example, conjugate gradient) to respect these constraints. First, we must project each gradient vector onto the hyperplane that satisfies the linear constraints on the parameters; this step is fairly straightforward (see exercise 19.6). Second, we must ensure that parameters are nonnegative; this requires restricting possible steps to avoid stepping out of the allowed bounds.

reparameterization

The second approach is to *reparameterize* the problem. Suppose we introduce new parameters $\lambda_{x|\boldsymbol{u}}$, and define

$$P(x \mid \boldsymbol{u}) = \frac{e^{\lambda_{x|\boldsymbol{u}}}}{\sum_{x' \in Val(X)} e^{\lambda_{x'|\boldsymbol{u}}}}, \tag{19.3}$$

for each X and its parents $\boldsymbol{U}$. Now, any choice of values for the λ parameters will lead to legal conditional probabilities. We can compute the gradient of the log-likelihood with respect to the λ parameters using the chain rule of partial derivatives, and then use standard (unmodified) conjugate gradient ascent procedure. See exercise 19.7.

Lagrange multipliers

Another way of dealing with the constraints implied by conditional probabilities is to use the method of *Lagrange multipliers*, reviewed in appendix A.5.3. Applying this method to the optimization of the log-likelihood leads to the method we discuss in the next section, and we defer this discussion; see also exercise 19.8.

Having dealt with this subtlety, we can now apply any gradient ascent procedure to find a local maximum of the likelihood function. As discussed, in most missing value problems, the likelihood function has many local maxima. Unfortunately, gradient ascent procedures are guaranteed to achieve only a local maximum of the function. Many of the techniques we discussed earlier in the book can be used to avoid local maxima and increase our chances of finding a global maximum, or at least a better local maximum: the general-purpose methods of appendix A.4.2, such as multiple random starting points, or applying random perturbations to convergence points; and the more specialized data perturbation methods of algorithm 18.1.

19.2.2 Expectation Maximization (EM)

expectation maximization

An alternative algorithm for optimizing a likelihood function is the *expectation maximization* algorithm. Unlike gradient ascent, EM is not a general-purpose algorithm for nonlinear function optimization. Rather, it is tailored specifically to optimizing likelihood functions, attempting to build on the tools we had for solving the problem with complete data.

19.2.2.1 Intuition

Recall that when learning from complete data, we can collect sufficient statistics for each CPD. We can then estimate parameters that maximize the likelihood with respect to these statistics. As we saw, in the case of missing data, we do not have access to the full sufficient statistics. Thus, we cannot use the same strategy for our problem. For example, in a simple $X \rightarrow Y$ network, if we see the training instance $\langle ?, y^1 \rangle$, then we do not know whether to count this instance toward the count $M[x^1, y^1]$ or toward the count $M[x^0, y^1]$.

A simple approach is to "fill in" the missing values arbitrarily. For example, there are strategies that fill in missing values with "default values" (say *false*) or by randomly choosing a value. Once we fill in all the missing values, we can use standard, complete data learning procedure. Such approaches are called called *data imputation* methods in statistics.

data imputation

The problem with such an approach is that the procedure we use for filling in the missing values introduces a bias that will be reflected in the parameters we learn. For example, if we fill all missing values with *false*, then our estimate will be skewed toward higher (conditional) probability of *false*. Similarly, if we use a randomized procedure for filling in values, then the probabilities we estimate will be skewed toward the distribution from which we sample missing values. This might be better than a skew toward one value, but it still presents a problem. Moreover, when we consider learning with hidden variables, it is clear that an imputation procedure will not help us. The values we fill in for the hidden variable are conditionally independent from the values of the other variables, and thus, using the imputed values, we will not learn any dependencies between the hidden variable and the other variables in the network.

A different approach to filling in data takes the perspective that, when learning with missing data, we are actually trying to solve two problems at once: learning the parameters, and hypothesizing values for the unobserved variables in each of the data cases. Each of these tasks is fairly easy when we have the solution to the other. Given complete data, we have the statistics, and we can estimate parameters using the MLE formulas we discussed in chapter 17. Conversely, given a choice of parameters, we can use probabilistic inference to hypothesize the likely values (or the distribution over possible values) for unobserved variables. Unfortunately, because we have neither, the problem is difficult.

The EM algorithm solves this "chicken and egg" problem using a bootstrap approach. We start out with some arbitrary starting point. This can be either a choice of parameters, or some initial assignment to the hidden variables; these assignments can be either random, or selected using some heuristic approach. Assuming, for concreteness, that we begin with a parameter assignment, the algorithm then repeats two steps. First, we use our current parameters to *complete* the data, using probabilistic inference. We then treat the completed data as if it were observed and learn a new set of parameters.

data completion

More precisely, suppose we have a guess θ^0 about the parameters of the network. The resulting model defines a joint distribution over all the variables in the domain. Given a partial instance, we can compute the posterior (using our putative parameters) over all possible assignments to the missing values in that instance. The EM algorithm uses this probabilistic completion of the different data instances to estimate the *expected* value of the sufficient statistics. It then finds the parameters θ^1 that maximize the likelihood with respect to these statistics.

Somewhat surprisingly, this sequence of steps provably improves our parameters. In fact, as we will prove formally, unless our parameters have not changed due to these steps (such that $\theta^0 = \theta^1$), our new parameters θ^1 necessarily have a higher likelihood than θ^0. But now we can iteratively repeat this process, using θ^1 as our new starting point. Each of these operations can be thought of as taking an "uphill" step in our search space. More precisely, we will show (under very benign assumptions) that: each iteration is guaranteed to improve the log-likelihood function; that this process is guaranteed to converge; and that the convergence point is a fixed point of the likelihood function, which is essentially always a local maximum. Thus, the guarantees of the EM algorithm are similar to those of gradient ascent.

19.2.2.2 An Example

We start with a simple example to clarify the concepts. Consider the simple network shown in figure 19.6. In the fully observable case, our maximum likelihood parameter estimate for the parameter $\hat{\theta}_{d^1|c^0}$ is:

$$\hat{\theta}_{d^1|c^0} = \frac{M[d^1, c^0]}{M[c^0]} = \frac{\sum_{m=1}^{M} \mathbf{I}\{\xi[m]\langle D, C\rangle = \langle d^1, c^0\rangle\}}{\sum_{m=1}^{M} \mathbf{I}\{\xi[m]\langle C\rangle = c^0\}},$$

where $\xi[m]$ is the m'th training example. In the fully observable case, we knew exactly whether the indicator variables were 0 or 1. Now, however, we do not have complete data cases, so we no longer know the value of the indicator variables.

Consider a partially specified data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$. There are four possible instantiations to the missing variables B, C which could have given rise to this partial data case: $\langle b^1, c^1\rangle$, $\langle b^1, c^0\rangle$, $\langle b^0, c^1\rangle$, $\langle b^0, c^0\rangle$. We do not know which of them is true, or even which of them is more likely.

However, assume that we have some estimate $\boldsymbol{\theta}$ of the values of the parameters in the model. In this case, we can compute how likely each of these completions is, given our distribution. That is, we can define a distribution $Q(B, C) = P(B, C \mid \boldsymbol{o}, \boldsymbol{\theta})$ that induces a distribution over the four data cases. For example, if our parameters $\boldsymbol{\theta}$ are:

$$\begin{array}{llll} \boldsymbol{\theta}_{a^1} & = 0.3 & \boldsymbol{\theta}_{b^1} & = 0.9 \\ \boldsymbol{\theta}_{d^1|c^0} & = 0.1 & \boldsymbol{\theta}_{d^1|c^1} & = 0.8 \\ \boldsymbol{\theta}_{c^1|a^0,b^0} & = 0.83 & \boldsymbol{\theta}_{c^1|a^1,b^0} & = 0.6 \\ \boldsymbol{\theta}_{c^1|a^0,b^1} & = 0.09 & \boldsymbol{\theta}_{c^1|a^1,b^1} & = 0.2, \end{array}$$

then $Q(B, C) = P(B, C \mid a^1, d^0, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned} Q(\langle b^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.2/0.2196 = 0.0492 \\ Q(\langle b^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.9/0.2196 = 0.8852 \\ Q(\langle b^0, c^1\rangle) &= 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.2/0.2196 = 0.0164 \\ Q(\langle b^0, c^0\rangle) &= 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.9/0.2196 = 0.0492, \end{aligned}$$

where 0.2196 is a normalizing factor, equal to $P(a^1, d^0 \mid \boldsymbol{\theta})$.

If we have another example $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then $Q'(A, C) = P(A, C \mid b^1, d^1, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned} Q'(\langle a^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.8/0.1675 = 0.2579 \\ Q'(\langle a^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.1/0.1675 = 0.1290 \\ Q'(\langle a^0, c^1\rangle) &= 0.7 \cdot 0.9 \cdot 0.09 \cdot 0.8/0.1675 = 0.2708 \\ Q'(\langle a^0, c^0\rangle) &= 0.7 \cdot 0.9 \cdot 0.91 \cdot 0.1/0.1675 = 0.3423. \end{aligned}$$

Intuitively, now that we have estimates for how likely each of the cases is, we can treat these estimates as truth. That is, we view our partially observed data case $\langle a^1, ?, ?, d^0\rangle$ as consisting of four complete data cases, each of which has some *weight* lower than 1. The weights correspond to our estimate, based on our current parameters, on how likely is this particular completion of the partial instance. (This approach is somewhat reminiscent of the weighted particles in the likelihood weighting algorithm.) Importantly, as we will discuss, we do

weighted data instances

not usually explicitly generate these completed data cases; however, this perspective is the basis for the more sophisticated methods.

More generally, let $\boldsymbol{H}[m]$ denote the variables whose values are missing in the data instance $\boldsymbol{o}[m]$. We now have a data set $\mathcal{D}^+$ consisting of

$$\cup_m \{\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle \ : \ \boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])\},$$

where each data case $\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle$ has weight $Q(\boldsymbol{h}[m]) = P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$.

We can now do standard maximum likelihood estimation using these completed data cases. We compute the *expected sufficient statistics*:

expected sufficient statistics

$$\bar{M}_{\boldsymbol{\theta}}[\boldsymbol{y}] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \boldsymbol{I}\{\xi\langle \boldsymbol{Y}\rangle = \boldsymbol{y}\}.$$

We then use these expected sufficient statistics as if they were real in the MLE formula. For example:

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{\bar{M}_{\boldsymbol{\theta}}[d^1, c^0]}{\bar{M}_{\boldsymbol{\theta}}[c^0]}.$$

In our example, suppose the data consist of the two instances $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$ and $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then, using the calculated Q and Q' from above, we have that

$$\begin{aligned}
\bar{M}_{\boldsymbol{\theta}}[d^1, c^0] &= Q(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\
&= 0.1290 + 0.3423 = 0.4713 \\
\bar{M}_{\boldsymbol{\theta}}[c^0] &= Q(\langle b^1, c^0\rangle) + Q(\langle b^0, c^0\rangle) + Q'(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\
&= 0.8852 + 0.0492 + 0.1290 + 0.3423 = 1.4057.
\end{aligned}$$

Thus, in this example, using these particular parameters to compute expected sufficient statistics, we get

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{0.4713}{1.4057} = 0.3353.$$

Note that this estimate is quite different from the parameter $\theta_{d^1|c^0} = 0.1$ that we used in our estimate of the expected counts. The initial parameter and the estimate are different due to the incorporation of the observations in the data.

This intuition seems nice. However, it may require an unreasonable amount of computation. To compute the expected sufficient statistics, we must sum over all the completed data cases. The number of these completed data cases is much larger than the original data set. For each $\boldsymbol{o}[m]$, the number of completions is exponential in the number of missing values. Thus, if we have more than few missing values in an instances, an implementation of this approach will not be able to finish computing the expected sufficient statistics.

Fortunately, it turns out that there is a better approach to computing the expected sufficient statistic than simply summing over all possible completions. Let us reexamine the formula for an expected sufficient statistic, for example, $\bar{M}_{\boldsymbol{\theta}}[c^1]$. We have that

$$\bar{M}_{\boldsymbol{\theta}}[c^1] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \boldsymbol{I}\{\xi\langle C\rangle = c^1\}.$$

Let us consider the internal summation, say for a data case $o = \langle a^1, ?, ?, d^0 \rangle$. We have four possible completions, as before, but we are only summing over the two that are consistent with c^1, that is, $Q(b^1, c^1) + Q(b^0, c^1)$. This expression is equal to $Q(c^1) = P(c^1 \mid a^1, d^0, \theta) = P(c^1 \mid o[1], \theta)$. This idea clearly generalizes to our other data cases. Thus, we have that

$$\bar{M}_\theta[c^1] = \sum_{m=1}^{M} P(c^1 \mid o[m], \theta).$$

Now, recall our formula for sufficient statistics in the fully observable case:

$$M[c^1] = \sum_{m=1}^{M} \mathbf{I}\{\xi[m]\langle C \rangle = c^1\}.$$

Our new formula is identical, except that we have substituted our indicator variable — either 0 or 1 — with a probability that is somewhere between 0 and 1. Clearly, if in a certain data case we get to observe C, the indicator variable and the probability are the same. Thus, we can view the expected sufficient statistics as filling in soft estimates for hard data when the hard data are not available.

We stress that we use *posterior* probabilities in computing expected sufficient statistics. Thus, although our choice of θ clearly influences the result, the data also play a central role. This is in contrast to the probabilistic completion we discussed earlier that used a prior probability to fill in values, regardless of the evidence on the other variables in the same instances.

19.2.2.3 The EM Algorithm for Bayesian Networks

We now present the basic EM algorithm and describe the guarantees that it provides.

Networks with Table-CPDs Consider the application of the EM algorithm to a general Bayesian network with table-CPDs. Assume that the algorithm begins with some initial parameter assignment θ^0, which can be chosen either randomly or using some other approach. (The case where we begin with some assignment to the missing data is analogous.) The algorithm then repeatedly executes the following phases, for $t = 0, 1, \ldots$

expected sufficient statistics

Expectation (E-step): The algorithm uses the current parameters θ^t to compute the *expected sufficient statistics*.

- For each data case $o[m]$ and each family X, U, compute the joint distribution $P(X, U \mid o[m], \theta^t)$.
- Compute the expected sufficient statistics for each x, u as:

$$\bar{M}_{\theta^t}[x, u] = \sum_m P(x, u \mid o[m], \theta^t).$$

E-step

This phase is called the *E-step* (*expectation step*) because the counts used in the formula are the expected sufficient statistics, where the expectation is with respect to the current set of parameters.

Algorithm 19.2 Expectation-maximization algorithm for BN with table-CPDs

```
Procedure Compute-ESS (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ,    // Set of parameters for G
    D     // Partially observed data set
  )
    // Initialize data structures
  for each i = 1, ..., n
    for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
      M̄[x_i, u_i] ← 0
    // Collect probabilities from all instances
  for each m = 1 ... M
    Run inference on ⟨G, θ⟩ using evidence o[m]
    for each i = 1, ..., n
      for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
        M̄[x_i, u_i] ← M̄[x_i, u_i] + P(x_i, u_i | o[m])
  return {M̄[x_i, u_i] : ∀i = 1, ..., n, ∀x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)}

Procedure Expectation-Maximization (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ^0,  // Initial set of parameters for G
    D     // Partially observed data set
  )
  for each t = 0, 1 ..., until convergence
    // E-step
    {M̄_t[x_i, u_i]} ← Compute-ESS(G, θ^t, D)
    // M-step
    for each i = 1, ..., n
      for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
        θ^{t+1}_{x_i|u_i} ← M̄_t[x_i, u_i] / M̄_t[u_i]
  return θ^t
```

Maximization (M-step): Treat the expected sufficient statistics as observed, and perform maximum likelihood estimation, with respect to them, to derive a new set of parameters. In other words, set

$$\theta^{t+1}_{x|u} = \frac{\bar{M}_{\theta^t}[x, u]}{\bar{M}_{\theta^t}[u]}$$

M-step

This phase is called the *M-step* (*maximization step*), because we are maximizing the likelihood relative to the expected sufficient statistics.

A formal version of the algorithm is shown fully in algorithm 19.2.

The maximization step is straightforward. The more difficult step is the expectation step. How do we compute expected sufficient statistics? We must resort to Bayesian network inference over the network $\langle \mathcal{G}, \boldsymbol{\theta}^t \rangle$. Note that, as in the case of gradient ascent, the only expected sufficient statistics that we need involve a variable and its parents. Although one can use a variety of different inference methods to perform the inference task required for the E-step, we can, as in the case of gradient ascent, use the clique tree or cluster graph algorithm. Recall that the family-preservation property guarantees that X and its parents $\boldsymbol{U}$ will be together in some cluster in the tree or graph. **Thus, once again, we can do all of the required inference for each data case using one run of message-passing calibration.**

General Exponential Family ⋆ The same idea generalizes to other distributions where the likelihood has sufficient statistics, in particular, all models in the *exponential family*. Recall that, in this case, we have a sufficient statistic function $\tau(\xi)$ that maps a complete instance to a vector of sufficient statistics. When learning parameters of such a model, we can summarize the data using the sufficient statistic function τ. For a complete data set $\mathcal{D}^+$, we define

exponential family

$$\tau(\mathcal{D}^+) = \sum_m \tau(\boldsymbol{o}[m], \boldsymbol{h}[m]).$$

We can now define the same E and M-steps described earlier for this more general case.

E-step

Expectation (*E-step*): For each data case $\boldsymbol{o}[m]$, the algorithm uses the current parameters $\boldsymbol{\theta}^t$ to define a model, and a posterior distribution:

$$Q(\boldsymbol{H}[m]) = P(\boldsymbol{H}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t).$$

expected sufficient statistics

It then uses inference in this distribution to compute the *expected sufficient statistics*:

$$\boldsymbol{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_m \boldsymbol{E}_Q[\tau(\boldsymbol{o}[m], \boldsymbol{h}[m])]. \tag{19.4}$$

M-step

Maximization (*M-step*): As in the case of table-CPDs, once we have the expected sufficient statistics, the algorithm treats them as if they were real and uses them as the basis for maximum likelihood estimation, using the appropriate form of the ML estimator for this family.

Convergence Results Somewhat surprisingly, this simple algorithm can be shown to have several important properties. We now state somewhat simplified versions of the relevant results, deferring a more precise statement to the next section.

The first result states that each iteration is guaranteed to improve the log-likelihood of the current set of parameters.

Theorem 19.3

During iterations of the EM procedure of algorithm 19.2, we have

$$\ell(\boldsymbol{\theta}^t : \mathcal{D}) \leq \ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}).$$

Thus, the EM procedure is constantly increasing the log-likelihood objective function. Because the objective function can be shown to be bounded (under mild assumptions), this procedure is guaranteed to converge. By itself, this result does not imply that we converge to a maximum of the objective function. Indeed, this result is only "almost true":

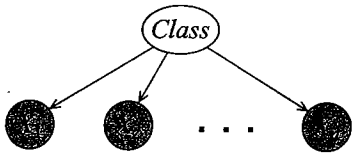


Figure 19.7 The naive Bayes clustering model. In this model each observed variables X_i is independent of the other observed variables given the value of the (unobserved) cluster variable C.

Theorem 19.4 *Suppose that θ^t is such that $\theta^{t+1} = \theta^t$ during EM, and θ^t is also an interior point of the allowed parameter space. Then θ^t is a stationary point of the log-likelihood function.*

This result shows that EM converges to a stationary point of the likelihood function. Recall that a stationary point can be a local maximum, local minimum, or a saddle point. Although it seems counterintuitive that by taking upward steps we reach a local minimum, it is possible to construct examples where EM converges to such a point. **However, nonmaximal convergence points can only be reached from very specific starting points, and are moreover not stable, since even small perturbations to the parameters are likely to move the algorithm away from this point. Thus, in practice, EM generally converges to a local maximum of the likelihood function.**

19.2.2.4 Bayesian Clustering Using EM

clustering

One important application of learning with incomplete data, and EM in particular, is to the problem of *clustering*. Here, we have a set of data points in some feature space $\boldsymbol{X}$. Let us even assume that they are fully observable. We want to classify these data points into coherent categories, that is, categories of points that seem to share similar statistical properties.

Bayesian clustering

The *Bayesian clustering* paradigm views this task as a learning problem with a single hidden variable C that denotes the category or class from which an instance comes. Each class is associated with a probability distribution over the features of the instances in the class. In most cases, we assume that the instances in each class c come from some coherent, fairly simple, distribution. In other words, we postulate a particular form for the *class-conditional distribution* $P(\boldsymbol{x} \mid c)$. For example, in the case of real-valued data, we typically assume that the class-conditional distribution is a multivariate Gaussian (see section 7.1). In discrete settings, we typically assume that the class-conditional distribution is a naive Bayes structure (section 3.1.3), where each feature is independent of the rest given the class variable. Overall, this approach views the data as coming from a *mixture distribution* and attempts to use the hidden variable to separate out the mixture into its components.

mixture distribution

naive Bayes

Suppose we consider the case of a *naive Bayes* model (figure 19.7) where the hidden class variable is the single parent of all the observed feature. In this particular learning scenario, the E-step involves computing the probability of different values of the class variables for each instance. Thus, we can think of EM as performing a soft classification of the instances, that is, each data instance belongs, to some degree, to multiple classes.

In the M-step we compute the parameters for the CPDs in the form $P(X \mid C)$ and the prior

$P(C)$ over the classes. These estimates depends on our expected sufficient statistics. These are:

$$\begin{aligned}\bar{M}_{\boldsymbol{\theta}}[c] &\leftarrow \sum_m P(c \mid x_1[m], \ldots, x_n[m], \boldsymbol{\theta}^t) \\ \bar{M}_{\boldsymbol{\theta}}[x_i, c] &\leftarrow \sum_m P(c, x_i \mid x_1[m], \ldots, x_n[m], \boldsymbol{\theta}^t).\end{aligned}$$

We see that an instance helps determine the parameters for all of the classes that it participates in (that is, ones where $P(c \mid \boldsymbol{x}[m])$ is bigger than 0). Stated a bit differently, each instance "votes" about the parameters of each cluster by contributing to the statistics of the conditional distribution given that value of the cluster variable. However, the weight of this vote depends on the probability with which we assign the instance to the particular cluster.

Once we have computed the expected sufficient statistics, the M-step is, as usual, simple. The parameters for the class variable CPD are

$$\theta_c^{t+1} \leftarrow \frac{\bar{M}_{\boldsymbol{\theta}}[c]}{M},$$

and for the conditional CPD are

$$\theta_{x_i|c}^{t+1} \leftarrow \frac{\bar{M}_{\boldsymbol{\theta}}[x_i, c]}{\bar{M}_{\boldsymbol{\theta}}[c]}.$$

We can develop similar formulas for the case where some of the observed variables are continuous, and we use a conditional Gaussian distribution (a special case of definition 5.15) to model the CPD $P(X_i \mid C)$. The application of EM to this specific model results in a simple and efficient algorithm.

We can think of the clustering problem with continuous observations from a geometrical perspective, where each observed variable X_i represents one coordinate, and instances correspond to points. The parameters in this case represent the distribution of coordinate values in each of the classes. Thus, each class corresponds to a *cloud* of points in the input data. In each iteration, we reestimate the location of these clouds. In general, depending on the particular starting point, EM will proceed to assign each class to a dense cloud.

The EM algorithm for clustering uses a "soft" cluster assignment, allowing each instance to contribute part of its weight to multiple clusters, proportionately to its probability of belonging to each of them. As another alternative, we can consider "hard clustering," where each instance contributes all of its weight to the cluster to which it is most likely to belong. This variant, called *hard-assigment EM* proceeds by performing the following steps.

hard-assignment EM

- Given parameters $\boldsymbol{\theta}^t$, we assign $c[m] = \arg\max_c P(c \mid \boldsymbol{x}[m], \boldsymbol{\theta}^t)$ for each instance m. If we let $\mathcal{H}^t$ comprise all of the assignments $c[m]$, this results in a complete data set $(\mathcal{D}^+)^t = \langle \mathcal{D}, \mathcal{H}^t \rangle$.
- Set $\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta} : (\mathcal{D}^+)^t)$. This step requires collecting sufficient statistics from $(\mathcal{D}^+)^t$, and then choosing MLE parameters based on these.

This approach is often used where the class-conditional distributions $P(\boldsymbol{X} \mid c)$ are all "round" Gaussian distributions with unit variance. Thus, each class c has its own mean vector $\boldsymbol{\mu}_c$, but a unit covariance matrix. In this case, the most likely class for an instance $\boldsymbol{x}$ is simply the

class c such that the Euclidean distance between x and μ_c is smallest. In other words, each point gravitates to the class to which it is "closest." The reestimation step is also simple. It simply selects the mean of the class to be at the center of the cloud of points that have aligned themselves with it. This process iterates until convergence. This algorithm is called *k-means*.

k-means

Although hard-assignment EM is often used for clustering, it can be defined more broadly; we return to it in greater detail in section 19.2.2.6.

---

collaborative filtering

Box 19.A — Case Study: Discovering User Clusters. *In box 18.C, we discussed the* collaborative filtering *problem, and the use of Bayesian network structure learning to address it. A different application of Bayesian network learning to the collaborative filtering data task, proposed by Breese et al. (1998), utilized a Bayesian clustering approach. Here, one can introduce a cluster variable C denoting subpopulations of customers. In a simple model, the individual purchases X_i of each user are taken to be conditionally independent given the user's cluster assignment C. Thus, we have a naive Bayes clustering model, to which we can apply the EM algorithm. (As in box 18.C, items i that the user did not purchase are assigned $X_i = x_i^0$.)*

This learned model can be used in several ways. Most obviously, we can use inference to compute the probability that the user will purchase item i, given a set of purchases S. Empirical studies show that this approach achieves lower performance than the structure learning approach of box 18.C, probably because the "user cluster" variable simply cannot capture the complex preference patterns over a large number of items. However, this model can provide significant insight into the types of users present in a population, allowing, for example, a more informed design of advertising campaigns.

As one example, Bayesian clustering was applied to a data set of people browsing the MSNBC website. Each article was associated with a binary random variable X_i, which took the value x_i^1 if the user followed the link to the article. Figure 19.A.1 shows the four largest clusters produced by Bayesian clustering applied to this data set. Cluster 1 appears to represent readers of commerce and technology news (a large segment of the reader population at that period, when Internet news was in its early stages). Cluster 2 are people who mostly read the top-promoted stories in the main page. Cluster 3 are readers of sports news. In all three of these cases, the user population was known in advance, and the website contained a page targeting these readers, from which the articles shown in the table were all linked. The fourth cluster was more surprising. It appears to contain readers interested in "softer" news. The articles read by this population were scattered all over the website, and users often browsed several pages to find them. Thus, the clustering algorithm revealed an unexpected pattern in the data, one that may be useful for redesigning the website.

19.2.2.5 Theoretical Foundations ⋆

So far, we used an intuitive argument to derive the details of the EM algorithm. We now formally analyze this algorithm and prove the results regarding its convergence properties.

At each iteration, EM maintains the "current" set of parameters. Thus, we can view it as a local learning algorithm. Each iteration amounts to taking a step in the parameter space from

Cluster 1 (36 percent)	**Cluster 2** (29 percent)
E-mail delivery isn't exactly guaranteed	757 Crashes at sea
Should you buy a DVD player?	Israel, Palestinians agree to direct talks
Price low, demand high for Nintendo	Fuhrman pleads innocent to perjury
Cluster 3 (19 percent)	**Cluster 4** (12 percent)
Umps refusing to work is the right thing	The truth about what things cost
Cowboys are reborn in win over eagles	Fuhrman pleads innocent to perjury
Did Orioles spend money wisely?	Real astrology

Figure 19.A.1 — Application of Bayesian clustering to collaborative filtering. Four largest clusters found by Bayesian clustering applied to MSNBC news browsing data. For each cluster, the table shows the three news articles whose probability of being browsed is highest.

θ^t to θ^{t+1}. This is similar to gradient-based algorithms, except that in those algorithms we have good understanding of the nature of the step, since each step attempts to go uphill in the steepest direction. Can we find a similar justification for the EM iterations?

The basic outline of the analysis proceeds as follows. We will show that each iteration can be viewed as maximizing an *auxiliary function*, rather than the actual likelihood function. The choice of auxiliary function depends on the current parameters at the beginning of the iteration. The auxiliary function is nice in the sense that it is similar to the likelihood function in complete data problems. The crucial part of the analysis is to show how the auxiliary function relates to the likelihood function we are trying to maximize. As we will show, the relation is such that we can show that the parameters that maximize the auxiliary function in an iteration also have better likelihood than the parameters with which we started the iteration.

The Expected Log-Likelihood Function Assume we are given a data set $\mathcal{D}$ that consists of partial observations. Recall that $\mathcal{H}$ denotes a possible assignment to all the missing values in our data set. The combination of $\mathcal{D}, \mathcal{H}$ defines a complete data set $\mathcal{D}^+ = \langle \mathcal{D}, \mathcal{H} \rangle = \{\boldsymbol{o}[m], \boldsymbol{h}[m]\}_m$, where in each instance we now have a full assignment to all the variables. We denote by $\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$ the log-likelihood of the parameters $\boldsymbol{\theta}$ with respect to this completed data set.

Suppose we are not sure about the true value of $\mathcal{H}$. Rather, we have a probabilistic estimate that we denote by a distribution Q that assigns a probability to each possible value of $\mathcal{H}$. Note that Q is a joint distribution over full assignments to all of the missing values in the entire data set. Thus, for example, in our earlier network, if $\mathcal{D}$ contains two instances $\boldsymbol{o}[1] = \langle a^1, ?, ?, d^0 \rangle$ and $\boldsymbol{o}[1] = \langle ?, b^1, ?, d^1 \rangle$, then Q is a joint distribution over $B[1], C[1], A[2], C[2]$.

In the fully observed case, our score for a set of parameters was the log-likelihood. In this case, given Q, we can use it to define an average score, which takes into account the different possible completions of the data and their probabilities. Specifically, we define the *expected log-likelihood* as:

expected log-likelihood

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{\mathcal{H}} Q(\mathcal{H}) \ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$$

This function has appealing characteristics that are important in the derivation of EM.

The first key property is a consequence of the linearity of expectation. Recall that when learning table-CPDs, we showed that

$$\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle) = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i] \log \theta_{x_i | \boldsymbol{u}_i}.$$

Because the only terms in this sum that depend on $\langle \mathcal{D}, \mathcal{H} \rangle$ are the counts $M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i]$, and these appear within a linear function, we can use linearity of expectations to show that

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} \boldsymbol{E}_Q \left[M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i] \right] \log \theta_{x_i | \boldsymbol{u}_i}.$$

If we now generalize our notation to define

$$\bar{M}_Q[x_i, \boldsymbol{u}_i] = \boldsymbol{E}_{\mathcal{H} \sim Q} \left[M_{(\mathcal{D}, \mathcal{H})}[x_i, \boldsymbol{u}_i] \right] \tag{19.5}$$

we obtain

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} \bar{M}_Q[x_i, \boldsymbol{u}_i] \log \theta_{x_i | \boldsymbol{u}_i}.$$

This expression has precisely the same form as the log-likelihood function in the complete data case, but using the expected counts rather than the exact full-data counts. The implication is that instead of summing over all possible completions of the data, we can evaluate the expected log-likelihood based on the expected counts.

The crucial point here is that the log-likelihood function of complete data is *linear* in the counts. This allows us to use linearity of expectations to write the expected likelihood as a function of the expected counts.

The same idea generalizes to any model in the exponential family, which we defined in chapter 8. Recall that a model is in the exponential family if we can write:

$$P(\xi \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} A(\xi) \exp \{ \langle \mathrm{t}(\boldsymbol{\theta}), \tau(\xi) \rangle \},$$

where $\langle \cdot, \cdot \rangle$ is the inner product, $A(\xi)$, $\mathrm{t}(\boldsymbol{\theta})$, and $Z(\boldsymbol{\theta})$ are functions that define the family, and $\tau(\xi)$ is the sufficient statistics function that maps a complete instance to a vector of sufficient statistics.

As discussed in section 17.2.5, when learning parameters of such a model, we can summarize the data using the sufficient statistic function τ. We define

$$\tau(\langle \mathcal{D}, \mathcal{H} \rangle) = \sum_{m} \tau(\boldsymbol{o}[m], \boldsymbol{h}[m]).$$

Because the model is in the exponential family, we can write the log-likelihood $\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$ as a *linear function* of $\tau(\langle \mathcal{D}, \mathcal{H} \rangle)$

$$\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle) = \langle \mathrm{t}(\boldsymbol{\theta}), \tau(\langle \mathcal{D}, \mathcal{H} \rangle) \rangle + \sum_{m} A(\boldsymbol{o}[m], \boldsymbol{h}[m]) - \log Z(\boldsymbol{\theta}).$$

Using the linearity of expectation, we see that

$$\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \langle \mathrm{t}(\boldsymbol{\theta}), \mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] \rangle + \sum_m \mathbf{E}_Q[A(\boldsymbol{o}[m], \boldsymbol{h}[m])] - M \log Z(\boldsymbol{\theta}).$$

Because $A(\boldsymbol{o}[m], \boldsymbol{h}[m])$ does not depend on the choice of $\boldsymbol{\theta}$, we can ignore it. We are left with maximizing the function:

$$\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \langle \mathrm{t}(\boldsymbol{\theta}), \mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] \rangle - M \log Z(\boldsymbol{\theta}) + \text{const.} \tag{19.6}$$

In summary, the derivation here is directly analogous to the one for table-CPDs. The expected log-likelihood is a linear function of the expected sufficient statistics $\mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)]$. We can compute these as in equation (19.4), by aggregating their expectation in each instance in the training data. Now, maximizing the right-hand side of equation (19.6) is equivalent to maximum likelihood estimation in a *complete* data set where the sum of the sufficient statistics coincides with the expected sufficient statistics $\mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)]$. These two steps are exactly the E-step and M-step we take in each iteration of the EM procedure shown in algorithm 19.2. In the procedure, the distribution Q that we are using is $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. Because instances are assumed to be independent given the parameters, it follows that

$$P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t) = \prod_m P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t),$$

where $\boldsymbol{h}[m]$ are the missing variables in the m'th data instance, and $\boldsymbol{o}[m]$ are the observations in the m'th instance. Thus, we see that in the t'th iteration of the EM procedure, we choose $\boldsymbol{\theta}^{t+1}$ to be the ones that maximize $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ with $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. This discussion allows us to understand a single iteration as an (implicit) optimization step of a well-defined target function.

Choosing Q The discussion so far has showed that we can use properties of exponential models to efficiently maximize the expected log-likelihood function. Moreover, we have seen that the t'th EM iteration can be viewed as maximizing $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ where Q is the conditional probability $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. This discussion, however, does not provide us with guidance as to why we choose this particular auxiliary distribution Q. Note that each iteration uses a different Q distribution, and thus we cannot relate the optimization taken in one iteration to the ones made in the subsequent one. We now show why the choice $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ allows us to prove that each EM iteration improves the likelihood function.

To do this, we will define a new function that will be the target of our optimization. Recall that our ultimate goal is to maximize the log-likelihood function. The log-likelihood is a function only of $\boldsymbol{\theta}$; however, in intermediate steps, we also have the current choice of Q. Therefore, we will define a new function that accounts for both $\boldsymbol{\theta}$ and Q, and view each step in the algorithm as maximizing this function.

We already encountered a similar problem in our discussion of approximate inference in chapter 11. Recall that in that setting we had a known distribution P and attempted to find an approximating distribution Q. This problem is similar to the one we face, except that in learning we also change the parameters of target distribution P to maximize the probability of the data.

Let us briefly summarize the main idea that we used in chapter 11. Suppose that $P = \tilde{P}/Z$ is some distribution, where $\tilde{P}$ is an unnormalized part of the distribution, specified by a product

of factors, and Z is the partition function that ensures that P sums up to one. We defined the energy functional *energy functional* as

$$F[P, Q] = \mathbf{E}_Q\left[\log \tilde{P}\right] + \mathbf{H}_Q(\mathcal{X}).$$

We then showed that the logarithm of the partition function can be rewritten as:

$$\log Z = F[P, Q] + \mathbf{D}(Q \| P).$$

How does this apply to the case of learning from missing data? We can choose

$$P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}) = P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta}) / P(\mathcal{D} \mid \boldsymbol{\theta})$$

as our distribution over $\mathcal{H}$ (we hold $\mathcal{D}$ and $\boldsymbol{\theta}$ fixed for now). With this choice, the partition function $Z(\boldsymbol{\theta})$ is the data likelihood $P(\mathcal{D} \mid \boldsymbol{\theta})$ and $\tilde{P}$ is the joint probability $P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})$, so that $\log \tilde{P} = \ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$. Rewriting the energy functional for this new setting, we obtain:

$$F_{\mathcal{D}}[\boldsymbol{\theta}, Q] = \mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \mathbf{H}_Q(\mathcal{H}).$$

expected log-likelihood Note that the first term is precisely the *expected log-likelihood* relative to Q. Applying our earlier analysis, we now can prove

Corollary 19.1

For any Q,

$$\begin{aligned} \ell(\boldsymbol{\theta} : \mathcal{D}) &= F_{\mathcal{D}}[\boldsymbol{\theta}, Q] + \mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})) \\ &= \mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \mathbf{H}_Q(\mathcal{H}) + \mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})). \end{aligned}$$

Both equalities have important ramifications. Starting from the second equality, since both the entropy $\mathbf{H}_Q(\mathcal{H})$ and the relative entropy $\mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}))$ are nonnegative, we conclude that the expected log-likelihood $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ is a lower bound on $\ell(\boldsymbol{\theta} : \mathcal{D})$. This result is data completion true for any choice of distribution Q. If we select $Q(\mathcal{H})$ to be the *data completion distribution* $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, the relative entropy term becomes zero. In this case, the remaining term $\mathbf{H}_Q(\mathcal{H})$ captures to a certain extent the difference between the expected log-likelihood and the real log-likelihood. Intuitively, when Q is close to being deterministic, the expected value is close to the actual value.

The first equality, for the same reasons, shows that, for any distribution Q, the F function is a lower bound on the log-likelihood. Moreover, this lower bound is tight for every choice of $\boldsymbol{\theta}$: if we choose $Q = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, the two functions have the same value. Thus, if we maximize the F function, we are bound to maximize the log-likelihood.

There many possible ways to optimize this target function. We now show that the EM procedure we described can be viewed as *implicitly* optimizing the EM functional F using a coordinate ascent particular optimization strategy. The strategy we are going to utilize is a *coordinate ascent* optimization. We start with some choice $\boldsymbol{\theta}$ of parameters. We then search for Q that maximizes $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$ while keeping $\boldsymbol{\theta}$ fixed. Next, we fix Q and search for parameters that maximize $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$. We continue in this manner until convergence.

We now consider each of these steps.

- **Optimizing** Q. Suppose that $\boldsymbol{\theta}$ are fixed, and we are searching for $\arg\max_Q F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$. Using corollary 19.1, we know that, if $Q^* = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, then

$$F_{\mathcal{D}}[\boldsymbol{\theta}, Q^*] = \ell(\boldsymbol{\theta} : \mathcal{D}) \geq F_{\mathcal{D}}[\boldsymbol{\theta}, Q].$$

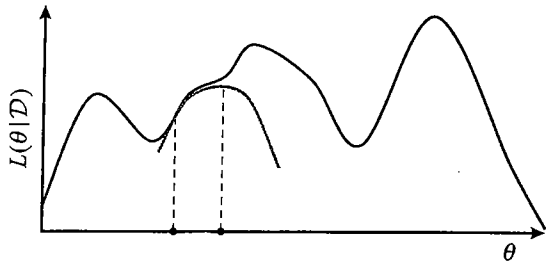


Figure 19.8 An illustration of the hill-climbing process performed by the EM algorithm. The black line represents the log-likelihood function; the point on the left represents θ^t; the gray line represents the expected log-likelihood derived from θ^t; and the point on the right represents the parameters θ^{t+1} that maximize this expected log-likelihood.

Thus, we maximize the EM functional by choosing the auxiliary distribution Q^*. In other words, we can view the E-step as implicitly optimizing Q by using $P(\mathcal{H} \mid \mathcal{D}, \theta^t)$ in computing the expected sufficient statistics.

- **Optimizing θ.** Suppose Q is fixed, and that we wish to find $\arg\max_\theta F_\mathcal{D}[\theta, Q]$. Because the only term in F that involves θ is $\mathbf{E}_Q[\ell(\theta : \langle \mathcal{D}, \mathcal{H} \rangle)]$, the maximization is equivalent to maximizing the expected log-likelihood. As we saw, we can find the maximum by computing expected sufficient statistics and then solving the MLE given these expected sufficient statistics.

Convergence of EM The discussion so far shows that the EM procedure can be viewed as maximizing an objective function; because the objective function can be shown to be bounded, this procedure is guaranteed to converge. However, it is not clear what can be said about the convergence points of this procedure. We now analyze the convergence points of this procedure in terms of our true objective: the log-likelihood function. Intuitively, as our procedure is optimizing the energy functional, which is a tight lower bound of the log-likelihood function, each step of this optimization also improves the log-likelihood. This intuition is illustrated in figure 19.8. In more detail, the E-step is selecting, at the current set of parameters, the distribution Q^t for which the energy functional is a tight lower bound to $\ell(\theta : \mathcal{D})$. The energy functional, which is a well-behaved concave function in θ, can be maximized effectively via the M-step, taking us to the parameters θ^{t+1}. Since the energy functional is guaranteed to remain below the log-likelihood function, this step is guaranteed to improve the log-likelihood. Moreover, the improvement is guaranteed to be at least as large as the improvement in the energy functional. More formally, using corollary 19.1, we can now prove the following generalization of theorem 19.5:

Theorem 19.5

During iterations of the EM procedure of algorithm 19.2, we have that

$$\ell(\theta^{t+1} : \mathcal{D}) - \ell(\theta^t : \mathcal{D}) \geq \mathbf{E}_{P(\mathcal{H} \mid \mathcal{D}, \theta^t)}\left[\ell(\theta^{t+1} : \mathcal{D}, \mathcal{H})\right] - \mathbf{E}_{P(\mathcal{H} \mid \mathcal{D}, \theta^t)}\left[\ell(\theta^t : \mathcal{D}, \mathcal{H})\right].$$

As a consequence, we obtain that:

$$\ell(\boldsymbol{\theta}^t : \mathcal{D}) \leq \ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}).$$

PROOF We begin with the first statement. Using corollary 19.1, with the distribution $Q^t(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ we have that

$$\begin{aligned}
\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) &= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t+1} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}) + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t+1})) \\
\ell(\boldsymbol{\theta}^{t} : \mathcal{D}) &= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}) + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t})) \\
&= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}).
\end{aligned}$$

The last step is justified by our choice of $Q^t(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. Subtracting these two terms, we have that

$$\begin{aligned}
&\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) - \ell(\boldsymbol{\theta}^{t} : \mathcal{D}) = \\
&\quad \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}, \mathcal{H})\right] - \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \mathcal{D}, \mathcal{H})\right] + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t+1})).
\end{aligned}$$

Because the last term is nonnegative, we get the desired inequality.

To prove the second statement of the theorem, we note that $\boldsymbol{\theta}^{t+1}$ is the value of $\boldsymbol{\theta}$ that maximizes $\boldsymbol{E}_{P(\mathcal{H}|\mathcal{D},\boldsymbol{\theta}^t)}[\ell(\boldsymbol{\theta} : \mathcal{D}, \mathcal{H})]$. Hence the value obtained for this expression for $\boldsymbol{\theta}^{t+1}$ is at least at large as the value obtained for any other set of parameters, including $\boldsymbol{\theta}^t$. It follows that the right-hand side of the inequality is nonnegative, which implies the first statement. ■

☞ **We conclude that EM performs a variant of hill climbing, in the sense that it improves the log-likelihood at each step. Moreover, the M-step can be understood as maximizing a lower-bound on the improvement in the likelihood. Thus, in a sense we can view the algorithm as searching for the largest possible improvement, when using the expected log-likelihood as a proxy for the actual log-likelihood.**

For most learning problems, we know that the log-likelihood is upper bounded. For example, if we have discrete data, then the maximal likelihood we can assign to the data is 1. Thus, the log-likelihood is bounded by 0. If we have a continuous model, we can construct examples where the likelihood can grow unboundedly; however, we can often introduce constraints on the parameters that guarantee a bound on the likelihood (see exercise 19.10). If the log-likelihood is bounded, and the EM iterations are nondecreasing in the log-likelihood, then the sequence of log-likelihoods at successive iterations must converge.

The question is what can be said about this convergence point. Ideally, we would like to guarantee convergence to the maximum value of our log-likelihood function. Unfortunately, as we mentioned earlier, we cannot provide this guarantee; however, we can now prove theorem 19.4, which shows convergence to a fixed point of the log-likelihood function, that is, one where the gradient is zero. We restate the theorem for convenience:

Theorem 19.6

Suppose that $\boldsymbol{\theta}^t$ is such that $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$ during EM, and $\boldsymbol{\theta}^t$ is also an interior point of the allowed parameter space. Then $\boldsymbol{\theta}^t$ is a stationary point of the log-likelihood function.

PROOF We start by rewriting the log-likelihood function using corollary 19.1.

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H}\rangle)] + \boldsymbol{H}_Q(\mathcal{H}) + \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})).$$

We now consider the gradient of $\ell(\theta : \mathcal{D})$ with respect to θ. Since the term $\boldsymbol{H}_Q(\mathcal{H})$ does not depend on θ, we get that

$$\nabla_\theta \ell(\theta : \mathcal{D}) = \nabla_\theta \boldsymbol{E}_Q[\ell(\theta : \langle \mathcal{D}, \mathcal{H} \rangle)] + \nabla_\theta \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \theta)).$$

This observation is true for any choice of Q. Now suppose we are in an EM iteration. In this case, we set $Q = P(\mathcal{H} \mid \mathcal{D}, \theta^t)$ and evaluate the gradient at θ^t.

A somewhat simplified proof runs as follows. Because $\theta = \theta^t$ is a minimum of the KL-divergence term, we know that $\nabla_\theta \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \theta^t))$ is 0. This implies that

$$\nabla_\theta \ell(\theta^t : \mathcal{D}) = \nabla_\theta \boldsymbol{E}_Q\left[\ell(\theta^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right].$$

Or, in other words, $\nabla_\theta \ell(\theta^t : \mathcal{D}) = 0$ if and only if $\nabla_\theta \boldsymbol{E}_Q\left[\ell(\theta^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right] = 0$.

Recall that $\theta^{t+1} = \arg\max_\theta \boldsymbol{E}_Q\left[\ell(\theta^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right]$. Hence the gradient of the expected likelihood at θ^{t+1} is 0. Thus, we conclude that $\theta^{t+1} = \theta^t$ only if $\nabla_\theta \boldsymbol{E}_Q\left[\ell(\theta^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right] = 0$. And so, at this point, $\nabla_\theta \ell(\theta^t : \mathcal{D}) = 0$. This implies that this set of parameters is a stationary point of the log-likelihood function.

The actual argument has to be somewhat more careful. Recall that the parameters must lie within some allowable set. For example, the parameters of a discrete random variable must sum up to one. Thus, we are searching within a constrained space of parameters. When we have constraints, we often do not have zero gradient. Instead, we get to a stationary point when the gradient is orthogonal to the constraints (that is, local changes within the allowed space do not improve the likelihood). The arguments we have stated apply equally well when we replace statements about equality to 0 with orthogonality to the constraints on the parameter space. ■

19.2.2.6 Hard-Assignment EM

In section 19.2.2.4, we briefly mentioned the idea of using a hard assignment to the hidden variables, in the context of applying EM to Bayesian clustering. We now generalize this simple idea to the case of arbitrary Bayesian networks.

hard-assignment EM

This algorithm, called *hard-assignment EM*, also iterates over two steps: one in which it completes the data given the current parameters θ^t, and the other in which it uses the completion to estimate new parameters θ^{t+1}. However, rather than using a soft completion of the data, as in standard EM, it selects for each data instance $\boldsymbol{o}[m]$ the single assignment $\boldsymbol{h}[m]$ that maximizes $P(\boldsymbol{h} \mid \boldsymbol{o}[m], \theta^t)$.

Although hard-assignment EM is similar in outline to EM, there are important differences. In fact, hard-assignment EM can be described as optimizing a different objective function, one that involves both the learned parameters and the learned assignment to the hidden variables. This objective is to maximize the likelihood of the complete data $\langle \mathcal{D}, \mathcal{H} \rangle$, given the parameters:

$$\max_{\theta, \mathcal{H}} \ell(\theta : \mathcal{H}, \mathcal{D}).$$

See exercise 19.14. Compare this objective to the EM objective, which attempts to maximize $\ell(\theta : \mathcal{D})$, averaging over all possible completions of the data.

Does this observation provide us insight on these two learning procedures? The intuition is that these two objectives are similar if $P(\mathcal{H} \mid \mathcal{D}, \theta)$ assigns most of the probability mass to

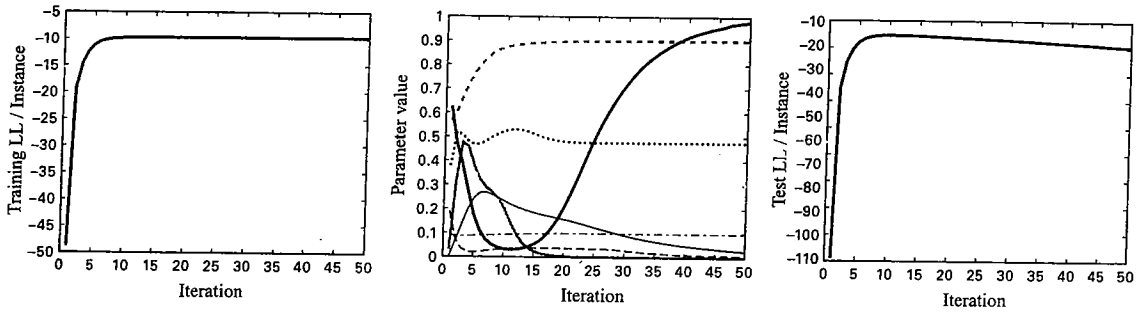


Figure 19.B.1 — Convergence of EM run on the ICU Alarm network. (a) Training likelihood. (b) Progress of several sample parameters. (c) Test data log-likelihood.

one completion of the data. In such a case, EM will effectively perform hard assignment during the E-step. However, if $P(\mathcal{H} \mid \mathcal{D}, \theta)$ is diffuse, the two algorithms will lead to very different solutions. In clustering, the hard-assignment version tends to increase the contrast between different classes, since assignments have to choose between them. In contrast, EM can learn classes that are overlapping, by having many instances contributing to two or more classes.

Another difference between the two EM variants is in the way they progress during the learning. Note that for a given data set, at the end of an iteration, the hard-assignment EM can be in one of a finite number of parameter values. Namely, there is only one parameter assignment for each possible assignment to $\mathcal{H}$. Thus, hard-assignment EM traverses a path in the combinatorial space of assignments to $\mathcal{H}$. The soft-assignment EM, on the other hand, traverses the continuous space of parameter assignments. The intuition is that hard-assignment EM converges faster, since it makes discrete steps. In contrast, soft-assignment EM can converge very slowly to a local maximum, since close to the maximum, each iteration makes only small changes to the parameters. The flip side of this argument is that soft-assignment EM can traverse paths that are infeasible to the hard-assignment EM. For example, if two clusters need to shift their means in a coordinated fashion, soft-assignment EM can progressively change their means. On the other hand, hard-assignment EM needs to make a "jump," since it cannot simultaneously reassign multiple instances and change the class means.

Box 19.B — Case Study: EM in Practice. *The EM algorithm is guaranteed to monotonically improve the training log-likelihood at each iteration. However, there are no guarantees as to the speed of convergence or the quality of the local maxima attained. To gain a better perspective of how the algorithm behaves in practice, we consider here the application of the method to the* ICU-Alarm *network discussed in earlier learning chapters.*

We start by considering the progress of the training data likelihood during the algorithm's iterations. In this example, 1,000 *samples were generated from the* ICU-Alarm *network. For each instance, we then independently and randomly hid 50 percent of the variables. As can be seen in figure 19.B.1a, much of the improvement over the performance of the random starting point is in the*

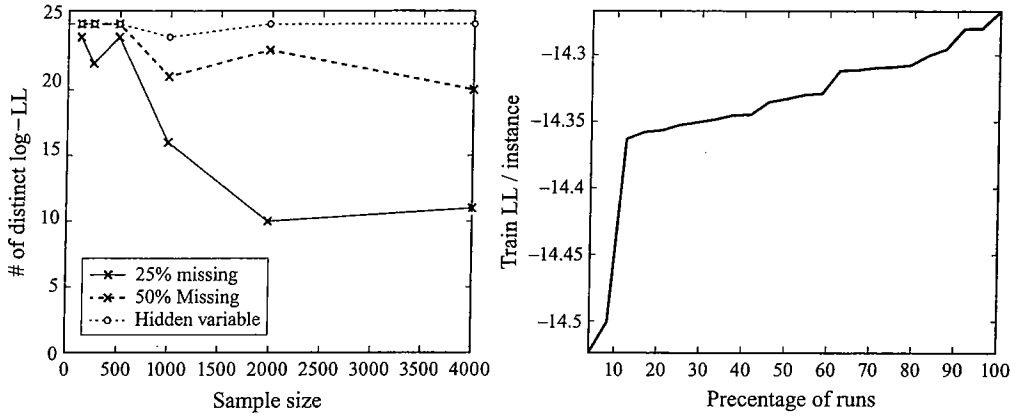


Figure 19.B.2 — Local maxima in likelihood surface. (a) Number of local maxima for different sample sizes and missing value configurations. (b) Distribution of training likelihood of local maxima attained for 25 random starting points with 1,000 samples and one hidden variable ('VENTTUBE').

first few iterations. However, examining the convergence of different parameters in (b), we see that some parameters change significantly after the fifth iteration, even though changes to the likelihood are relatively small. In practice, any nontrivial model will display a wide range of sensitivity to the network parameters. Given more training data, the sensitivity will, typically, overall decrease. Owing to these changes in parameters, the training likelihood continues to improve after the initial iterations, but very slowly. This behavior of fast initial improvement, followed by slow convergence, is typical of EM.

We next consider the behavior of the learned model on unseen test data. As we can see in (c), early in the process, test-data improvement correlates with training-data performance. However, after the 10*th iterations, training performance continues to improve, but test performance decreases. This phenomenon is an instance of* overfitting *to the training data. With more data or fewer unobserved values, this phenomenon will be less pronounced. With less data or hidden variables, on the other hand, explicit techniques for coping with the problem may be needed (see box 19.C).*

overfitting

A second key issue any type of optimization of the likelihood in the case of missing data is that of local maxima. To study this phenomenon, we consider the number of local maxima for 25 *random starting points under different settings. As the sample size (x-axis) grows, the number of local maxima diminishes. In addition, the number of local maxima when more values are missing (dashed line) is consistently greater than the number of local maxima in the setting where more data is available (solid line). Importantly, in the case where just a single variable is hidden, the number of local maxima is large, and remains large even when the amount of training data is quite large. To see that this is not just an artifact of possible permutations of the values of the hidden variable, and to demonstrate the importance of achieving a superior local maxima, in (b) we show the training set log-likelihood of the* 25 *different local maxima attained. The difference between the*

best and worst local maxima is over 0.2 *bit-per-instance. While this may not seem significant, for a training set of* $1,000$ *instances, this corresponds to a factor of* $2^{0.2*1,000} \approx 10^{60}$ *in the training set likelihood. We also note that the spread of the quality in different local maxima is quite uniform, so that it is not easy to attain a good local maximum with a small number of random trials.*

19.2.3 Comparison: Gradient Ascent versus EM

So far we have discussed two algorithms for parameter learning with incomplete data: gradient ascent and EM. As we will discuss (see box 19.C), there are many issues involved in the actual implementation of these algorithms: the choice of initial parameters, the stopping criteria, and so forth. However, before discussing these general points, it is worth comparing the two algorithms.

There are several points of similarity in the overall strategy of both algorithms. Both algorithms are *local* in nature. At each iteration they maintain a "current" set of parameters, and use these to find the next set. Moreover, both perform some version of greedy optimization based on the current point. Gradient ascent attempts to progress in the steepest direction from the current point. EM performs a greedy step in improving its target function given the local parameters. Finally, both algorithms provide a guarantee to converge to local maxima (or, more precisely, to stationary points where the gradient is 0). On one hand, this is an important guarantee, in the sense that both are at least locally maximal. On the other hand, this is a weak guarantee, since many real-world problems have multimodal likelihood functions, and thus we do not know how far the learned parameters are from the global maximum (or maxima).

In terms of the actual computational steps, the two algorithms are also quite similar. For table-CPDs, the main component of either an EM iteration or a gradient step is computing the expected sufficient statistics of the data given the current parameters. This involves performing inference on each instance. Thus, both algorithms can exploit dynamic programming procedures (for example, clique tree inference) to compute all the expected sufficient statistics in an instance efficiently.

In term of implementation details, the algorithms provide different benefits. On one hand, gradient ascent allows to use "black box" nonlinear optimization techniques, such as conjugate gradient ascent (see appendix A.5.2). This allows the implementation to build on a rich set of existing tools. Moreover, gradient ascent can be easily applied to various CPDs by using the chain rule of derivatives. On the other hand, EM relies on maximization from complete data. Thus, it allows for a fairly straightforward use of learning procedure for complete data in the case of incomplete data. The only change is replacing the part that accumulates sufficient statistics by a procedure that computes expected sufficient statistics. As such, most people find EM easier to implement.

A final aspect for consideration is the convergence rate of the algorithm. Although we cannot predict in advance how many iterations are needed to learn parameters, analysis can show the general behavior of the algorithm in terms of how fast it approaches the convergence point.

Suppose we denote by $\ell_t = \ell(\theta^t : \mathcal{D})$ the likelihood of the solution found in the t'th iteration (of either EM or gradient ascent). The algorithm converges toward $\ell^* = \lim_{t \to \infty} \ell_t$. The error at the t'th iteration is

$$\epsilon_t = \ell^* - \ell_t.$$

convergence rate

Although we do not go through the proof, one can show that EM has *linear convergence rate*. This means that for each domain there exists a t_0 and $\alpha < 1$ such that for all $t \geq t_0$

$$\epsilon_{t+1} \leq \alpha \epsilon_t.$$

On the face of it, this is good news, since it shows that the error decreases at each iteration. Such a convergence rate means that $\ell_{t+1} - \ell_t = \epsilon_t - \epsilon_{t+1} \geq \epsilon_t(1 - \alpha)$. In other words, if we know α, we can bound the error

$$\epsilon_t \leq \frac{\ell_{t+1} - \ell_t}{1 - \alpha}.$$

While this result provides a bound on the error (and also suggests a way of estimating it), it is not always a useful one. In particular, if α is relatively close to 1, then even when the difference is likelihood between successive iterations is small, the error can be much larger. Moreover, the number of iterations to convergence can be very large. In practice we see this behavior quite often. **The first iterations of EM show huge improvement in the likelihood. These are then followed by many iterations that slowly increase the likelihood; see box 19.B. Conjugate gradient often has opposite behavior. The initial iterations (which are far away from the local maxima) often take longer to improve the likelihood. However, once the algorithm is in the vicinity of maximum, where the log-likelihood function is approximately quadratic, this method is much more efficient in zooming on the maximum.** Finally, it is important to keep in mind that these arguments are asymptotic in the number of iterations; the actual number of iterations required for convergence may not be in the asymptotic regime. Thus, the rate of convergence of different algorithms may not be the best indicator as to which of them is likely to work most efficiently in practice.

Box 19.C — Skill: Practical Considerations in Parameter Learning. *There are a few practical considerations in implementing both gradient-based methods and EM for learning parameters with missing data. We now consider a few of these. We present these points mostly in the context of the EM algorithm, but most of our points apply equally to both classes of algorithms.*

In a practical implementation of EM, there are two key issues that one needs to address. The first is the presence of local maxima. As demonstrated in box 19.B, the likelihood of even relatively simple networks can have a large number of local maxima that significantly differ in terms of their quality. There are several adaptations of these local search algorithms that aim to consistently reach beneficial local maxima. These adaptations include a judicious selection of initialization, and methods for modifying the search so as to achieve a better local maximum. The second key issue involves the convergence of the algorithm: determining convergence, and improving the rate of convergence.

Local Maxima *One of the main limitations of both the EM and the gradient ascent procedures is that they are only guaranteed to reach a stationary point, which is usually a local maximum. How do we improve the odds of finding a global — or at least a good local — maximum?*

The first place where we can try to address the issue of local maxima is in the initialization of the algorithm. EM and gradient ascent, as well as most other "local" algorithms, require a starting point — a set of initial parameters that the algorithm proceeds to improve. Since both

algorithms are deterministic, this starting point (implicitly) determines which local maximum is found. **In practice, different initializations can result in radically different convergence points, sometimes with very different likelihood values. Even when the likelihood values are similar, different convergence points may represent semantically different conclusions about the data.** *This issue is particularly severe when hidden variables are involved, where we can easily obtain very different clusterings of the data. For example, when clustering text documents by similarity (for example, using a version of the model in box 17.E where the document cluster variable is hidden), we can learn one model where the clusters correspond to document topics, or another where they correspond to the style of the publication in which the document appeared (for example, newspaper, webpage, or blog). Thus, initialization should generally be considered very seriously in these situations, especially when the amount of missing data is large or hidden variables are involved.*

In general, we can initialize the algorithm either in the E-step, by picking an initial set of parameters, or in the M-step, by picking an initial assignment to the unobserved variables. In the first type of approach, the simplest choices for starting points are either a set of parameters fixed in advance or randomly chosen parameters. If we use predetermined initial parameters, we should exercise care in choosing them, since a misguided choice can lead to very poor outcomes. In particular, for some learning problems, the seemingly natural choice of uniform parameters can lead to disastrous results; see exercise 19.11. Another easy choice is applicable for parts of the network where we have only a moderate amount of missing data. Here, we can sometimes estimate parameters using only the observed data, and then use those to initialize the E-step. Of course, this approach is not always feasible, and it is inapplicable when we have a hidden variable. A different natural choice is to use the mean of our prior over parameters. On one hand, if we have good prior information, this might serve as a good starting position. Note that, although this choice does bias the learning algorithm to prefer the prior's view of the data, the learned parameters can be drastically different in the end. On the other hand, if the prior is not too informative, this choice suffers from the same drawbacks we mentioned earlier. Finally, a common choice is to use a randomized starting point, an approach that avoids any intrinsic bias. However, there is also no reason to expect that a random choice will give rise to a good solution. For this reason, often one tries multiple random starting points, and the convergence point of highest likelihood is chosen.

The second class of methods initializes the procedure at the M-step by completing the missing data. Again, there are many choices for completing the data. For example, we can use a uniform or a random imputation method to assign values to missing observations. This procedure is particularly useful when we have different patterns of missing observations in each sample. Then, the counts from the imputed data consist of actual counts combined with imputed ones. The real data thus bias the estimated parameters to be reasonable. Another alternative is to use a simplified learning procedure to learn initial assignment to missing values. This procedure can be, for example, hard-assignment EM. As we discussed, such a procedure usually converges faster and therefore can serve as a good initialization. However, hard-assignment EM also requires a starting point, or a selection among multiple random starting points.

When learning with hidden variables, such procedures can be more problematic. For example, if we consider a naive Bayes clustering model and use random imputation, the result would be that we randomly assign instances to clusters. With a sufficiently large data set, these clusters will be very similar (since they all sample from the same population). In a smaller data set the sampling noise might distinguish the initial clusters, but nonetheless, this is not a very informed starting point. We

discuss some methods for initializing a hidden variable in section 19.5.3.

Other than initialization, we can also consider modifying our search so as to reduce the risk of getting stuck at a poor local optimum. The problem of avoiding local maxima is a standard one, and we describe some of the more common solutions in appendix A.4.2. Many of these solutions are applicable in this setting as well. As we mentioned, the approach of using multiple random restarts is commonly used, often with a beam search *modification to quickly prune poor starting points. In particular, in this beam search variant, K EM runs are carried out in parallel and every few iterations only the most promising ones are retained. A variant of this approach is to generate K EM threads at each step by slightly perturbing the most beneficial $k < K$ threads from the previous iteration. While such adaptations have no formal guarantees, they are extremely useful in practice in terms trading off quality of solution and computational requirements.*

beam search

Annealing methods (appendix A.4.2) have also been used successfully in the context of the EM algorithm. In such methods, we gradually transform from an easy objective with a single local maximum to the desired EM objective, and thereby we potentially avoid many local maxima that are far away from the central basin of attraction. Such an approach can be carried out by directly smoothing the log-likelihood function and gradually reducing the level to which it is smoothed, or implicitly by gradually altering the weights of training instances.

Finally, we note that we can never determine with certainty whether the EM convergence point is truly the global maximum. In some applications this limitation is acceptable — for example, if we care only about fitting the probability distribution over the training examples (say for detecting instances from a particular subpopulation). In this case, if we manage to learn parameters that assign high probability for samples in the target population, then we might be content even if these parameters are not the best ones possible. On the other hand, if we want to use the learned parameters to reveal insight about the domain, then we might care about whether the parameters are truly the optimal ones or not. In addition, if the learning procedure does not perform well, we have to decide whether the problem stems from getting trapped in a poor local maximum, or from the fact that the model is not well suited to the distribution in our particular domain.

Stopping Criteria *Both algorithms we discussed have the property that they will reach a fixed point once they converged on a stationary point of the likelihood surface. In practice, we never really reach the stationary point, although we can get quite close to it. This raises the question of when we stop the procedure.*

The basic idea is that when solutions at successive iterations are similar to each other, additional iterations will not change the solution by much. The question is how to measure similarity of solutions. There are two main approaches. The first is to compare the parameters from successive iterations. The second is to compare the likelihood of these choices of parameters. Somewhat surprisingly, these two criteria are quite different. In some situations small changes in parameters lead to dramatic changes in likelihood, and in others large changes in parameters lead to small changes in the likelihood.

To understand how there can be a discrepancy between changes in parameters and changes in likelihood, consider the properties of the gradient as shown in theorem 19.2. Using a Taylor expansion of the likelihood, this gradient provides us with an estimate how the likelihood will change when we change the parameters. We see that if $P(x, \mathbf{u} \mid \mathbf{o}[m], \boldsymbol{\theta})$ is small in most data instances, then the gradient $\frac{\partial \ell(\boldsymbol{\theta}:\mathcal{D})}{\partial P(x|\mathbf{u})}$ will be small. This implies that relatively large changes in

$P(x \mid \boldsymbol{u})$ will not change the likelihood by much. This can happen for example if the event $\boldsymbol{u}$ is uncommon in the training data, and the value of $P(x \mid \boldsymbol{u})$ is involved in the likelihood only in a few instances. On the flip side, if the event $x, \boldsymbol{u}$ has a large posterior in all samples, then the gradient $\frac{\partial \ell(\theta:\mathcal{D})}{\partial P(x|\boldsymbol{u})}$ will be of size proportional to M. In such a situation a small change in the parameter can result in a large change in the likelihood.

In general, since we are aiming to maximize the likelihood, large changes in the parameters that have negligible effect on the likelihood are of less interest. Moreover, measuring the magnitude of changes in parameters is highly dependent on our parameterization. For example, if we use the reparameterization of equation (19.3), the difference (say in Euclidean distance) between two sets of parameters can change dramatically. Using the likelihood for tracking convergence is thus less sensitive to these choices and more directly related to the goal of the optimization.

Even once we decide what to measure, we still need to determine when we should stop the process. Some gradient-based methods, such as conjugate gradient ascent, build an estimate of the second-order derivative of the function. Using these derivatives, they estimate the improvement we expect to have. We can then decide to stop when the expected improvement is not more than a fixed amount of log-likelihood units. We can apply similar stoping criteria to EM, where again, if the change in likelihood in the last iteration is smaller than a predetermined threshold we stop the iterations.

Importantly, although the training set log-likelihood is guaranteed to increase monotonically until convergence, there is no guarantee that the generalization performance of the model — the expected log-likelihood relative to the underlying distribution — also increases monotonically. (See section 16.3.1.) Indeed, it is often the case that, as we approach convergence, the generalization performance starts to decrease, due to *overfitting* of the parameters to the specifics of the training data.

overfitting

Thus, an alternative approach is to measure directly when additional improvement to the training set likelihood does not contribute to generalization. To do so we need to separate the available data into a training set and a validation set (see box 16.A). We run learning on the training set, but at the end of each iteration we evaluate the log-likelihood of the validation set *(which is not seen during learning). We stop the procedure when the likelihood of the validation set does not improve. (As usual, the actual performance of the model would then need to be evaluated on a separate test set.) This method allows us to judge when the procedure stops learning the interesting phenomena and begins to overfit the training data. On the flip side, such a procedure is both slower (since we need to evaluate likelihood on an additional data set at the end of iteration) and forces us to train on a smaller subset of data, increasing the risk of overfitting. Moreover, if the validation set is small, then the estimate of the generalization ability by the likelihood on this set is noisy. This noise can influence the stopping time.*

validation set

Finally, we note that in EM much of the improvement is typically observed in the first few iterations, but the final convergence can be quite slow. Thus, in practice, it is often useful to limit the number of EM iterations or use a lenient convergence threshold. This is particularly important when EM is used as part of a higher-level algorithm (for example, structure learning) and where, in the intermediate stages of the overall learning algorithm, approximate parameter estimates are often sufficient. Moreover, early stopping can help reduce overfitting, as we discussed.

Accelerating Convergence *There are also several strategies that can help improve the rate of convergence of EM to its local optimum. We briefly list a few of them.*

The first idea is to use hybrid algorithms *that mix EM and gradient methods. The basic intuition is that EM is good at rapidly moving to the general neighborhood of a local maximum in few iterations but bad at pinpointing the actual maximum. Advanced gradient methods, on the other hand, quickly converge once we are close to a maximum. This observation suggests that we should run EM for few iterations and then switch over to using a method such as conjugate gradient. Such hybrid algorithms are often much more efficient. Another alternative is to use* accelerated EM *methods that take even larger steps in the search than standard EM (see section 19.7).*

accelerated EM

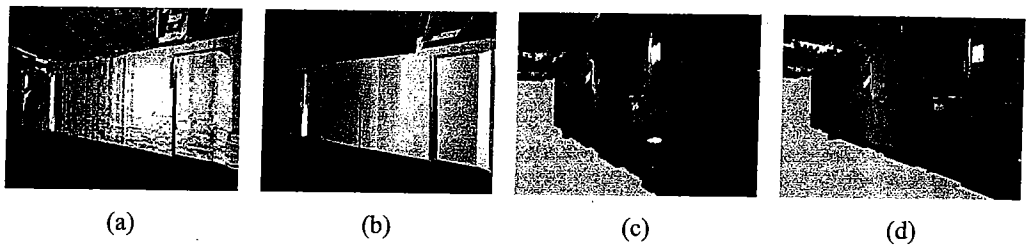
incremental EM

Another class of variations comprises incremental methods. *In these methods we do not perform a full E-step or a full M-step. Again, the high-level intuition is that, since we view our procedure as maximizing the energy functional $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$, we can consider steps that increase this functional but do not necessarily find the maximum value parameters or Q. For example, recall that $\boldsymbol{\theta}$ consists of several subcomponents, one per CPD. Rather than maximizing all the parameters at once, we can consider a partial update where we maximize the energy functional with respect to one of the subcomponents while freezing the others; see exercise 19.16. Another type of partial update is based on writing Q as a product of independent distributions — each one over the missing values in a particular instance. Again, we can optimize the energy functional with respect to one of these while freezing the other; see exercise 19.17. These partial updates can provide two types of benefit: they can require less computation than a full EM update, and they can propagate changes between the statistics and the parameters much faster, reducing the total number of iterations.*

Box 19.D — Case Study: EM for Robot Mapping. *One interesting application of the EM algorithm is to robotic mapping. Many variants of this applications have been proposed; we focus on one by Thrun et al. (2004) that explicitly tries to use the probabilistic model to capture the structure in the environment.*

The data in this application are a point cloud representation of an indoor environment. The point cloud can be obtained by collecting a sequence of point clouds, measured along a robot's motion trajectory. One can use a robot localization procedure to (approximately) assess the robot's pose (position and heading) along each point in the trajectory, which allows the different measurements to be put on a common frame of reference. Although the localization process is not fully accurate, the estimates are usually reasonable for short trajectories. One can then take the points obtained over the trajectory, and fit the points using polygons, to derive a 3D map of the surfaces in the robot's environment. However, the noise in the laser measurements, combined with the errors in localization, leads adjacent polygons to have slightly different surface normals, giving rise to a very jagged representation of the environment.

The EM algorithm can be used to fit a more compact representation of the environment to the data, reducing the noise and providing a smoother, more realistic output. In particular, in this example, the model consists of a set of 3D planes $p_1, \ldots, p_K$, each characterized by two parameters α_k, β_k, where α_k is a unit-length vector in $\mathbb{R}^3$ that encodes the plane's surface normal vector, and β_k is a scalar that denotes its distance to the origin of the global coordinate system. Thus, the distance of any point x to the plane is $d(x, p_k) = |\alpha_k x - \beta_k|$.



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

Figure 19.D.1 — Sample results from EM-based 3D plane mapping (a) Raw data map obtained from range finder. (b) Planes extracted from map using EM. (c) Fragment of reconstructed surface using raw data. (d) The same fragment reconstructed using planes.

correspondence variable

data association

The probabilistic model also needs to specify, for each point $\boldsymbol{x}_m$ in the point cloud, to which plane $\boldsymbol{x}_m$ belongs. This assignment can be modeled via a set of correspondence variables C_m *such that $C_m = k$ if the measurement point $\boldsymbol{x}_m$ was generated by the kth plane. Each assignment to the correspondence variables, which are unobserved, encodes a possible solution to the* data association *problem. (See box 12.D for more details.) We define $P(\boldsymbol{X}_m \mid C_m = k : \boldsymbol{\theta}_k)$ to be $\propto \mathcal{N}\left(d(\boldsymbol{x}, p_k) \mid 0; \sigma^2\right)$. In addition, we also allow an additional value $C_m = 0$ that encodes points that are not generated by any of the planes; the distribution $P(\boldsymbol{X}_m \mid C_m = 0)$ is taken to be uniform over the (finite) space.*

Given a probabilistic model, the EM algorithm can be applied to find the assignment of points to planes — the correspondence variables, which are taken to be hidden; and the parameters α_k, β_k that characterize the planes. Intuitively, the E-step computes the assignment to the correspondence variables by assigning the weight of each point proportionately to its distance to each of them. The M-step then recomputes the parameters of each plane to fit the points assigned to it. See exercise 19.18 and exercise 19.19. The algorithm also contains an additional outer loop that heuristically suggests new surfaces to be added to the model, and removes surfaces that do not have enough support in the data (for example, one possible criterion can depend on the total weight that different data points assign to the surface).

The results of this algorithm are shown in figure 19.D.1. One can see that the resulting map is considerably smoother and more realistic than the results derived directly from the raw data.

19.2.4 Approximate Inference ⋆

The main computational cost in both gradient ascent and EM is in the computation of expected sufficient statistics. This step requires running probabilistic inference on each instance in the training data. These inference steps are needed both for computing the likelihood and for computing the posterior probability over events of the form x, pa_X for each variable and its parents. For some models, such as the naive Bayes clustering model, this inference step is almost trivial. For other models, this inference step can be extremely costly. In practice, we

often want to learn parameters for models where exact inference is impractical. Formally, this happens when the tree-width of the unobserved parts of the model is large. (Note the contrast to learning from complete data, where the cost of learning the model did not depend on the complexity of inference.) In such situations the cost of inference becomes the limiting factor in our ability to learn from data.

Example 19.9

Recall the network discussed in example 6.11 and example 19.9, where we have n students taking m classes, and the grade for each student in each class depends on both the difficulty of the class and his or her intelligence. In the ground network for this example, we have a set of variables $\boldsymbol{I} = \{I(s)\}$ for the n students (denoting the intelligence level of each student s), $\boldsymbol{D} = \{D(c)\}$ for the m courses (denoting the difficulty level of each course c), and $\boldsymbol{G} = \{G(s,c)\}$ for the grades, where each variable $G(s,c)$ has as parents $I(s)$ and $D(c)$. Since this network is derived from a plate model, the CPDs are all shared, so we only have three CPDs that must be learned: $P(I(S))$, $P(D(C))$ $P(G(S,C) \mid I(S), D(C))$.

Suppose we only observe the grades of the students but not their intelligence or the difficulty of courses, and we want to learn this model. First, we note that there is no way to force the model to respect our desired semantics for the (hidden) variables I and D; for example, a model in which we flip the two values for I is equally good. Nevertheless, we can hope that some value for I will correspond to "high intelligence" and the other to "low intelligence," and similarly for D.

To perform EM in this model, we need to infer the expected counts of assignments to triplets of variables of the form $I(s), D(c), G(s,c)$. Since we have parameter sharing, we will aggregate these counts and then estimate the CPD $P(G(S,C) \mid I(S), D(C))$ from the aggregate counts. The problem is that observing a variable $G(s,c)$ couples its two parents. Thus, this network induces a Markov network that has a pairwise potential between any pair of $I(s)$ and $D(c)$ variables that share an observed child. If enough grade variables are observed, this network will be close to a full bipartite graph, and exact inference about the posterior probability becomes intractable. This creates a serious problem in applying either EM or gradient ascent for learning this seemingly simple model from data. ■

An obvious solution to this problem is to use approximate inference procedures. A simple approach is to view inference as a "black box." Rather than invoking exact inference in the learning procedures shown in algorithm 19.1 and algorithm 19.2, we can simply invoke one of the approximate inference procedures we discussed in earlier chapters. This view is elegant because it decouples the choices made in the design of the learning procedure from the choices made in the approximate inference procedures.

However, this decoupling can obscure important effects of the approximation on our learning procedure. For example, suppose we use approximate inference for computing the gradient in a gradient ascent approach. In this case, our estimate of the gradient is generally somewhat wrong, and the errors in successive iterations are generally not consistent with each other. Such inaccuracies can confuse the gradient ascent procedure, a problem that is particularly significant when the procedure is closer to the convergence point and the gradient is close to 0, so that the errors can easily dominate. **A key question is whether learning with approximate inference results in an approximate learning procedure; that is, whether we are guaranteed to find a local maximum of an approximation of the likelihood function. In general, there are very few cases where we can provide any types of guarantees on the interaction between approximate inference and learning. Nevertheless, in practice, the use of approximate**

inference is often unavoidable, and so many applications use some form of approximate inference despite the lack of theoretical guarantees.

One class of approximation algorithms for which a unifying perspective is useful is in the combination of EM with the global approximate inference methods of chapter 11. Let us consider first the *structured variational* methods of section 11.5, where the integration is easiest to understand. In these methods, we are attempting to find an approximate distribution Q that is close to an unnormalized distribution $\tilde{P}$ in which we are interested. We saw that algorithms in this class can be viewed as finding a distribution Q in a suitable family of distributions that maximizes the energy functional:

structured variational

$$F[\tilde{P}, Q] = \boldsymbol{E}_Q\left[\log \tilde{P}\right] + \boldsymbol{H}_Q(\mathcal{X}).$$

Thus, in these approximate inference procedures, we search for a distribution Q that maximizes

$$\max_{Q \in \mathcal{Q}} F[\tilde{P}, Q].$$

We saw that we can view EM as an attempt to maximize the same energy functional, with the difference that we are also optimizing over the parameterization $\boldsymbol{\theta}$ of $\tilde{P}$. We can combine both goals into a single objective by requiring that the distribution Q used in the EM functional come from a particular family $\mathcal{Q}$. Thus, we obtain the following *variational EM* problem:

variational EM

$$\max_{\boldsymbol{\theta}} \max_{Q \in \mathcal{Q}} F_{\mathcal{D}}[\boldsymbol{\theta}, Q], \tag{19.7}$$

where $\mathcal{Q}$ is a family of approximate distributions we are considering for representing the distribution over the unobserved variables.

To apply the variational EM framework, we need to choose the family of distributions $\mathcal{Q}$ that will be used to approximate the distribution $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$. Importantly, because this posterior distribution is a product of the posteriors for the different training instance, our approximation Q can take the same form without incurring any error. Thus, we need only to decide how to represent the posterior $P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$ for each instance m. We therefore define a class $\mathcal{Q}$ that we will use to approximate $P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$. Importantly, since the evidence $\boldsymbol{o}[m]$ is different for each data instance m, the posterior distribution for each instance is also different, and hence we need to use a different distribution $Q[m] \in \mathcal{Q}$ to approximate the posterior for each data instance. In principle, using the techniques of section 11.5, we can use any class $\mathcal{Q}$ that allows tractable inference. In practice, a common solution is to use the mean field approximation, where we assume that Q is a product of marginal distributions (one per each unobserved value).

Example 19.10

Consider using the mean field approximation (see section 11.5.1) for the learning problem of example 19.9. Recall that in the mean field approximation we approximate the target posterior distribution by a product of marginals. More precisely, we approximate $P(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m) \mid \boldsymbol{o}, \boldsymbol{\theta})$ *by a distribution*

$$Q(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m)) = Q(I(s_1)) \cdots Q(I(s_n)) Q(D(c_1)) \cdots Q(D(c_m)).$$

Importantly, although the prior over the variables $I(s_1), \ldots, I(s_n)$ *is identical, their posterior is generally different. Thus, the marginal of each of the variable has different parameters in* Q *(and similarly for the* $D(c)$ *variables).*

In our approximate E-step, given a set of parameters θ for the model, we need to compute approximate expected sufficient statistics. We do so in two steps. First, we use iterations of the mean field update equation equation (11.54) to find the best choice of marginals in Q to approximate $P(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m) \mid \boldsymbol{o}, \boldsymbol{\theta})$. We then use the distribution Q to compute approximate expected sufficient statistics by finding:

$$\begin{aligned} \bar{M}_Q[G_{(i,j)}, I(s_i), D(c_j)] &= Q(I(s_i), D(c_j))\mathbf{I}\{G(s_i, c_j) = g_{(i,j)}\} \\ &= Q(I(s_i))Q(D(c_j))\mathbf{I}\{G(s_i, c_j) = g_{(i,j)}\}. \end{aligned}$$ ■

Given our choice of $\mathcal{Q}$, we can optimize the variational EM objective very similarly to the optimization of the exact EM objective, by iterating over two steps:

variational E-step

- **Variational E-step** For each m, find

$$Q^t[m] = \arg\max_{Q \in \mathcal{Q}} F_{\boldsymbol{o}[m]}[\boldsymbol{\theta}, Q].$$

  This step is identical to our definition of variational inference in chapter 11, and it can be implemented using the algorithms we discussed there, usually involving iterations of local updates until convergence.

  At the end of this step, we have an approximate distribution $Q^t = \prod_m Q^t[m]$ and can collect the expected sufficient statistics. To compute the expected sufficient statistics, we combine the observed values in the data with expected counts from the distribution Q. This process requires answering queries about events in the distribution Q. For some approximations, such as the mean field approximation, we can answer such queries efficiently (that is, by multiplying the marginal probabilities over each variables); see example 19.10. If we use a richer class of approximate distributions, we must perform a more elaborate inference process. Note that, because the approximation Q is simpler than the original distribution P, we have no guarantee that a clique tree for Q will respect the family-preservation property relative to families in P. Thus, in some cases, we may need to perform queries that are outside the clique tree used to perform the E-step (see section 10.3.3.2).

- **M-step** We find a new set of parameters

$$\boldsymbol{\theta}^{t+1} = \arg\max_{\theta} F_{\mathcal{D}}[\boldsymbol{\theta}, Q^t];$$

  this step is identical to the M-step in standard EM.

The preceding algorithm is essentially performing coordinate-wise ascent alternating between optimization of Q and $\boldsymbol{\theta}$. It opens up the way to alternative ways of maximizing the same objective function. For example, we can limit the number of iterations in the variational E-step. Since each such iteration improves the energy functional, we do not need to reach a maximum in the Q dimension before making an improvement to the parameters.

Importantly, regardless of the method used to optimize the variational EM functional of equation (19.7), we can provide some guarantee regarding the properties of our optimum. Recall that we showed that

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \max_{Q} F_{\mathcal{D}}[\boldsymbol{\theta}, Q] \geq \max_{Q \in \mathcal{Q}} F_{\mathcal{D}}[\boldsymbol{\theta}, Q].$$

lower bound

Thus, maximizing the objective of equation (19.7) maximizes a *lower bound* of the likelihood. When we limit the choice of Q to be in a particular family, we cannot necessarily get a tight bound on the likelihood. However, since we are maximizing a lower bound, we know that we do not overestimate the likelihood of parameters we are considering. If the lower bound is relatively good, this property implies that we distinguish high-likelihood regions in the parameter space from very low ones. Of course, if the lower bound is loose, this guarantee is not very meaningful.

We can try to extend these ideas to other approximation methods. For example, generalized belief propagation section 11.3 is an attractive algorithm in this context, since it can be fairly efficient. Moreover, because the cluster graph satisfies the family preservation property, computation of an expected sufficient statistic can be done locally within a single cluster in the graph. The question is whether such an approximation can be understood as maximizing a clear objective. Recall that cluster-graph belief propagation can be viewed as attempting to maximize an approximation of the energy functional where we replace the term $\boldsymbol{H}_Q(\mathcal{X})$ by approximate entropy terms. Using exactly the same arguments as before, we can then show that, if we use generalized belief propagation for computing expected sufficient statistics in the E-step, then we are effectively attempting to maximize the approximate version of the energy functional. In this case, we cannot prove that this approximation is a lower bound to the correct likelihood. Moreover, if we use a standard message passing algorithm to compute the fixed points of the energy functional, we have no guarantees of convergence, and we may get oscillations both within an E-step and over several steps, which can cause significant problems in practice. Of course, we can use other approximations of the energy functional, including ones that are guaranteed to be lower bounds of the likelihood, and algorithms that are guaranteed to be convergent. These approaches, although less commonly used at the moment, share the same benefits of the structured variational approximation.

More broadly, the ability to characterize the approximate algorithm as attempting to optimize a clear objective function is important. For example, an immediate consequence is that, to monitor the progress of the algorithm, we should evaluate the approximate energy functional, since we know that, at least when all goes well, this quantity should increase until the convergence point.

19.3 Bayesian Learning with Incomplete Data ⋆

19.3.1 Overview

In our discussion of parameter learning from complete data, we discussed the limitations of maximum likelihood estimation, many of which can be addressed by the Bayesian approach. In the Bayesian approach, we view the parameters as unobserved variables that influence the probability of all training instances. Learning then amounts to computing the probability of new examples based on the observation, which can be performed by computing the posterior probability over the parameters, and using it for prediction.

More precisely, in Bayesian reasoning, we introduce a prior $P(\boldsymbol{\theta})$ over the parameters, and are interested in computing the posterior $P(\boldsymbol{\theta} \mid \mathcal{D})$ given the data. In the case of complete data, we saw that if the prior has some properties (for example, the priors over the parameters of different CPDs are independent, and the prior is conjugate), then the posterior has a nice form

and is representable in a compact manner. Because the posterior is a product of the prior and the likelihood, it follows from our discussion in section 19.1.3 that these useful properties are lost in the case of incomplete data. In particular, as we can see from figure 19.4 and figure 19.5, the parameter variables are generally correlated in the posterior. Thus, we can no longer represent the posterior as a product of posteriors over each set of parameters. Moreover, the posterior will generally be highly complex and even multimodal. Bayesian inference over this posterior would generally require a complex integration procedure, which generally has no analytic solution.

MAP estimation

One approach, once we realize that incomplete data makes the prospects of exact Bayesian reasoning unlikely, is to focus on the more modest goal of *MAP estimation*, which we first discussed in section 17.4.4. In this approach, rather than integrating over the entire posterior $P(\mathcal{D}, \boldsymbol{\theta})$, we search for a maximum of this distribution:

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \frac{P(\boldsymbol{\theta})P(\mathcal{D} \mid \boldsymbol{\theta})}{P(\mathcal{D})}.$$

Ideally, the neighborhood of the MAP parameters is the center of mass of the posterior, and therefore, using them might be a reasonable approximation for averaging over parameters in their neighborhood. Using the same transformations as in equation (17.14), the problem reduces to one of computing the optimum:

$$\text{score}_{\text{MAP}}(\boldsymbol{\theta} : \mathcal{D}) = \ell(\boldsymbol{\theta} : \mathcal{D}) + \log P(\boldsymbol{\theta}).$$

MAP-EM

This function is simply the log-likelihood function with an additional prior term. Because this prior term is usually well behaved, we can generally easily extend both gradient-based methods and the EM algorithm to this case; see, for example, exercise 19.20 and exercise 19.21. Thus, finding MAP parameters is essentially as hard or as easy as finding MLE parameters. As such, it is often applicable in practice. Of course, the same caveats that we discussed in section 17.4.4 — the sensitivity to parameterization, and the insensitivity to the form of the posterior — also apply here.

A second approach is to try to address the task of full Bayesian learning using an approximate method. Recall from section 17.3 that we can cast Bayesian learning as inference in the meta-network that includes all the variables in all the instances as well as the parameters. Computing the probability of future events amounts to performing queries about the posterior probability of the $(M + 1)$st instance given the observations about the first M instances. In the case of complete data, we could derive closed-form solutions to this inference problem. In the case of incomplete data, these solutions do not exist, and so we need to resort to approximate inference procedure.

In theory, we can apply any approximate inference procedure for Bayesian network to this problem. Thus, all the procedures we discussed in the inference chapter can potentially be used for performing Bayesian inference with incomplete data. Of course, some are more suitable than others.

For example, we can conceivably perform likelihood weighting, as described in section 12.2: we first sample parameters from the prior, and then the unobserved variables. Each such sample will be weighted by the probability of the observed data given the sampled parameter and hidden variables. Such a procedure is relatively easy to implement and does not require running complex inference procedure. However, since the parameter space is a high-dimensional continuous region, the chance of sampling high-posterior parameters is exceedingly small. As a

result, virtually all the samples will be assigned negligible weight. Unless the learning problem is relatively easy and the prior is quite informative, this inference procedure would provide a poor approximation and would require a huge number of samples.

In the next two sections, we consider two approximate inference procedures that can be applied to this problem with some degree of success.

19.3.2 MCMC Sampling

A common strategy for dealing with hard Bayesian learning problems is to perform MCMC simulation (see section 12.3). Recall that, in these methods, we construct a Markov chain whose state is the assignment to all unobserved variables, such that the stationary distribution of the chain is posterior probability over these variables. In our case, the state of the chain consists of $\boldsymbol{\theta}$ and $\mathcal{H}$, and we need to ensure that the stationary distribution of the chain is the desired posterior distribution.

19.3.2.1 Gibbs Sampling

Gibbs sampling

One of the simplest MCMC strategies for complex multivariable chains is *Gibbs sampling*. In Gibbs sampling, we choose one of the variables and sample its value given the value of all the other variables. In our setting, there are two types of variables: those in $\mathcal{H}$ and those in $\boldsymbol{\theta}$; we deal with each separately.

Suppose $X[m]$ is one of the variables in $\mathcal{H}$. The current state of the MCMC sampler has a value for all other variables in $\mathcal{H}$ and for $\boldsymbol{\theta}$. Since the parameters are known, selecting a value for $X[m]$ requires a sampling step that is essentially the same as the one we did when we performed Gibbs sampling for inference in the m'th instance. This step can be performed using the same sampling procedure as in section 12.3.

meta-network

Now suppose that $\boldsymbol{\theta}_{X|\boldsymbol{U}}$ are the parameters for a particular CPD. Again, the current state of the sampler assigns value for all of the variables in $\mathcal{H}$. Since the structure of the *meta-network* is such that $\boldsymbol{\theta}_{X|\boldsymbol{U}}$ are independent of the parameters of all other CPDs given $\mathcal{D}$ and $\mathcal{H}$, then we need to sample from $P(\boldsymbol{\theta}_{X|\boldsymbol{U}} \mid \mathcal{D}, \mathcal{H})$ — the posterior distribution over the parameters given the complete data $\mathcal{D}, \mathcal{H}$. In section 17.4 we showed that, if the prior is of a particular form (for example, a product of Dirichlet priors), then the posterior based on complete data also has a compact form. Now, we can use these properties to sample from the posterior. To be concrete, if we consider table-CPDs with Dirichlet priors, then the posterior is a product of Dirichlet distributions, one for each assignment of values for $\boldsymbol{U}$. Thus, if we know how to sample from a Dirichlet distribution, then we can sample from this posterior. It turns out that sampling from a Dirichlet distribution can be done using a reasonably efficient procedure; see box 19.E.

Thus, we can apply Gibbs sampling to the meta-network. If we simulate a sufficiently long run, the samples we generate will be from the joint posterior probability of the parameters and the hidden variables. We can then use these samples to make predictions about new samples and to estimate the marginal posterior of parameters or hidden variables. The BUGS system (see box 12.C) provides a simple, general-purpose tool for Gibbs-sampling-based Bayesian learning.

Box 19.E — Skill: Sampling from a Dirichlet distribution. *Suppose we have a parameter vector random variable $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_k \rangle$ that is distributed according to a Dirichlet distribution $\boldsymbol{\theta} \sim Dirichlet(\alpha_1, \ldots, \alpha_K)$. How do we sample a parameter vector from this distribution?*

A useful way to sampling such a vector relies on an alternative definition of the Dirichlet distribution. We need to start with some definitions.

Definition 19.6 Gamma distribution

A continuous random variable X has a Gamma distribution $\text{Gamma}(\alpha, \beta)$ *if it has the density*

$$p(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

■

We can see that the $x^{\alpha-1}$ term is reminiscent of components of the Dirichlet distribution. Thus, it might not be too surprising that there is a connection between the two distributions.

Theorem 19.7

Let $X_1, \ldots, X_k$ be independent continuous random variables such that $X_i \sim \text{Gamma}(\alpha_i, 1)$. Define the random vector

$$\boldsymbol{\theta} = \left\langle \frac{X_1}{X_1 + \cdots + X_k}, \ldots, \frac{X_k}{X_1 + \cdots + X_k} \right\rangle.$$

Then, $\boldsymbol{\theta} \sim Dirichlet(\alpha_1, \ldots, \alpha_k)$.

Thus, we can think of a Dirichlet distribution as a two-step process. First, we sample k independent values, each from a separate Gamma distribution. Then we normalize these values to get a distribution. The normalization creates the dependency between the components of the vector $\boldsymbol{\theta}$.

This theorem suggests a natural way to sample from a Dirichlet distribution. If we can sample from Gamma distributions, we can sample values $X_1, \ldots, X_k$ from the appropriate Gamma distribution and then normalize these values.

The only remaining question is how to sample from a Gamma distribution. We start with a special case. If we consider a variable $X \sim \text{Gamma}(1, 1)$, then the density function is

$$p(X = x) = e^{-x}.$$

In this case, we can solve the cumulative distribution using simple integration and get that

$$P(X < x) = 1 - e^{-x}.$$

From this, it is not hard to show the following result:

Lemma 19.2

If $U \sim \text{Unif}([0 : 1])$, then $-\ln U \sim \text{Gamma}(1, 1)$.

In particular, if we want to sample parameter vectors from $Dirichlet(1, \ldots, 1)$, which is the uniform distribution over parameter vectors, we need to sample k values from the uniform distribution, take their negative logarithm, and normalize. Since a sample from a uniform distribution can be readily obtained from a pseudo-random number generator, we get a simple procedure for sampling from the uniform distribution over multinomial distributions. Note that this procedure is not the one we intuitively would consider for this problem. When $\alpha \neq 1$ the sampling problem is harder, and requires more sophisticated methods, often based on the rejection sampling *approach described in section 14.5.1; these methods are outside the scope of this book.*

19.3.2.2 Collapsed MCMC

Recall that, in many situations, we can make MCMC sampling more efficient by using collapsed particles that represent a partial state of the system. If we can perform exact inference over the remaining state, then we can use MCMC sampling over the smaller state space and thereby get more efficient sampling.

We can apply this idea in the context of Bayesian inference in two different ways. In one approach, we have *parameter collapsed particles*, where each particle is an assignment to the parameters $\boldsymbol{\theta}$, associated with a distribution over $\mathcal{H}$; in the other, we have *data completion distribution particles*, where each particle is an assignment to the unobserved variables $\mathcal{H}$, associated with a distribution over $\boldsymbol{\theta}$. We now discuss each of these approaches in turn.

Parameter Collapsed Particles Suppose we choose the collapsed particles to contain assignments to the parameters $\boldsymbol{\theta}$, accompanied by distributions over the hidden variables. Thus, we need to be able to deal with queries about $P(\mathcal{H} \mid \boldsymbol{\theta}, \mathcal{D})$ and $P(\mathcal{D}, \boldsymbol{\theta})$. First note that given $\boldsymbol{\theta}$, the different training instances are conditionally independent. Thus, we can perform inference in each instance separately. The question now is whether we can perform this instance-level inference efficiently. This depends on the structure of the network we are learning.

Consider, for example, the task of learning the naive Bayes clustering model of section 19.2.2.4. In this case, each instance has a single hidden variable, denoting the cluster of the instance. Given the value of the parameters, inference over the hidden variable involves summing over all the values of the hidden variable, and computing the probability of the observation variables given each value. These operations are linear in the size of the network, and thus can be done efficiently. This means that evaluating the likelihood of a proposed particle is quite fast. On the other hand, if we are learning parameters for the network of example 19.9, then the network structure is such that we cannot perform efficient exact inference. In this case the cost of evaluating the likelihood of a proposed particle is nontrivial and requires additional approximations. Thus, the ease of operations with this type of collapsed particle depends on the network structure.

In addition to evaluating the previous queries, we need to be able to perform the sampling steps. In particular, for Gibbs sampling, we need to be able to sample from the distribution:

$$P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_i} \mid \{\boldsymbol{\theta}_{X_j|\mathrm{Pa}_j}\}_{j \neq i}, \mathcal{D}).$$

Unfortunately, sampling from this conditional distribution is unwieldy. Even though we assume the value of all other parameters, since we do not have complete data, we are not guaranteed to have a simple form for this conditional distribution (see example 19.11). As an alternative to Gibbs sampling, we can use Metropolis-Hastings. Here, in each proposal step, we suggest new parameter values and evaluate the likelihood of these new parameters relative to the old ones. This step requires that we evaluate $P(\mathcal{D}, \boldsymbol{\theta})$, which is as costly as computing the likelihood. This fact makes it critical that we construct a good proposal distribution, since a poor one can lead to many (expensive) rejected proposals.

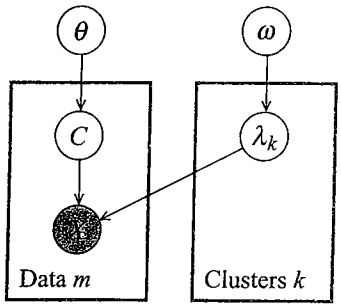


Figure 19.9 Plate model for Bayesian clustering

Example 19.11 Bayesian clustering

Let us return to the setting of Bayesian clustering *described in section 19.2.2.4. In this model, which is illustrated in figure 19.9, we have a set of data points* $\mathcal{D} = \{\boldsymbol{x}[1], \ldots, \boldsymbol{x}[M]\}$, *which are taken from one of a set of* K *clusters. We assume that each cluster is characterized by a distribution* $Q(\boldsymbol{X} \mid \boldsymbol{\theta})$, *which has the same form for each cluster, but different parameters. As we discussed, the form of the class-conditional distribution depends on the data; typical models include naive Bayes for discrete data, or Gaussian distributions for continuous data. This decision is orthogonal to our discussion here. Thus, we have a set of* K *parameter vectors* $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$, *each sampled from a distribution* $P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega})$. *We use a hidden variable* $C[m]$ *to represent the cluster from which the* m*'th data point was sampled. Thus, the class-conditional distribution* $P(\boldsymbol{X} \mid C = c^k, \boldsymbol{\lambda}_{1,\ldots,K}) = Q(\boldsymbol{X} \mid \boldsymbol{\lambda}_k)$. *We assume that the cluster variable* C *is sampled from a multinomial with parameters* $\boldsymbol{\theta}$, *sampled from a Dirichlet distribution* $\boldsymbol{\theta} \sim Dirichlet(\alpha_0/K, \ldots, \alpha_0/K)$. *The symmetry of the model relative to clusters reflects the fact that cluster identifiers are meaningless placeholders; it is only the partition of instances to clusters that is significant.*

To consider the use of parameter collapsed particles, let us begin by writing down the data likelihood given the parameters.

$$P(\mathcal{D} \mid \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K, \boldsymbol{\theta}) = \prod_{m=1}^{M} \left(\sum_{k=1}^{K} P(C[m] = c^k \mid \boldsymbol{\theta}) P(\boldsymbol{x}[m] \mid C[m] = c^k, \boldsymbol{\lambda}_k) \right). \quad (19.8)$$

In this case, because of the simple structure of the graphical model, this expression is easy to evaluate for any fixed set of parameters.

Now, let us consider the task of sampling $\boldsymbol{\lambda}_k$ *given* $\boldsymbol{\theta}$ *and* $\boldsymbol{\lambda}_{-k}$, *where we use a subscript of* $-k$ *to denote the set consisting of all of the values* $k' \in \{1, \ldots, K\} - \{k\}$. *The distribution with which we wish to sample* $\boldsymbol{\lambda}_k$ *is*

$$P(\boldsymbol{\lambda}_k \mid \boldsymbol{\theta}, \boldsymbol{\lambda}_{-k}, \mathcal{D}) \propto P(\mathcal{D} \mid \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K, \boldsymbol{\theta}).$$

Examining equation (19.8), we see that, given the data and the parameters other than $\boldsymbol{\lambda}_k$, *all of the terms* $P(\boldsymbol{x}[m] \mid C[m] = c^{k'}, \boldsymbol{\lambda}_{k'})$ *for* $k' \neq k$ *can now be treated as a constant, and aggregated into a single number; similarly, the terms* $P(C[m] = c^k \mid \boldsymbol{\theta})$ *are also constant. Hence, each of the terms inside the outermost product can be written as a linear function* $a_m P(\boldsymbol{x}[m] \mid C[m] =$

$c^k) + b_m$. *Unfortunately, the entire expression is a product of these linear functions, making the sampling distribution for* $\boldsymbol{\lambda}_k$ *proportional to a degree* M *polynomial in its likelihood function (multiplied by* $P(\boldsymbol{\lambda}_k)$*). This distribution is rarely one from which we can easily sample.*

random-walk chain

The Metropolis-Hastings approach is more feasible in this case. Here, as discussed in section 14.5.3, we can use a random-walk chain *as a proposal distribution and use the data likelihood to compute the acceptance probabilities. In this case, the computation is fairly straightforward, since it involves the ratio of two expressions of the form of equation (19.8), which are the same except for the values of* $\boldsymbol{\lambda}_k$*. Unfortunately, although most of the terms in the numerator and denominator are identical, they appear within the scope of a summation over* k *and therefore do not cancel. Thus, to compute the acceptance probability, we need to compute the full data likelihood for both the current and proposed parameter choices; in this particular network, this computation can be performed fairly efficiently.* ■

Data Completion Collapsed Particles An alternative choice is to use collapsed particles that assign a value to $\mathcal{H}$. In this case, each particle represents a complete data set. Recall that if the prior distribution satisfies certain properties, then we can use closed-form formulas to compute parameter posteriors from $P(\boldsymbol{\lambda} \mid \mathcal{D}, \mathcal{H})$ and to evaluate the marginal likelihood $P(\mathcal{D}, \mathcal{H})$. This implies that if we are using a well-behaved prior, we can evaluate the likelihood of particles in time that does not depend on the network structure.

For concreteness, consider the case where we are learning a Bayesian network with table-CPDs where we have an independent Dirichlet prior over each distribution $P(X_i \mid \mathrm{pa}_i)$. In this case, if we have a particle that represents a complete data instance, we can summarize it by the sufficient statistics $M[x_i, \mathrm{pa}_i]$, and using these we can compute both the posterior over parameters and the marginal likelihood using closed-form formulas; see section 17.4 and section 18.3.4.

Example 19.12

Let us return to the Bayesian clustering task, but now consider the setting where each particle $\mathbf{c}$ *is an assignment to the hidden variables* $C[1], \ldots, C[M]$*. Given an assignment to these variables, we are now in the regime of complete data, in which the different parameters are independent in the posterior. In particular, let* $I_k(\mathbf{c})$ *be the set of indexes* $\{m : c[m] = k\}$*. We can now compute the distribution associated with our particle* $\mathbf{c}$ *as a Dirichlet posterior*

$$P(\boldsymbol{\theta} \mid \mathbf{c}) = Dirichlet(\alpha_0/K + |I_1(\mathbf{c})|, \ldots, \alpha_0/K + |I_K(\mathbf{c})|).$$

If we now further assume that $P(\boldsymbol{\lambda} \mid \boldsymbol{\omega})$ *is a conjugate prior to* $Q(\boldsymbol{X} \mid \boldsymbol{\lambda})$*, we also obtain a set of closed-form posteriors:*

$$P(\boldsymbol{\lambda}_k \mid \mathbf{c}, \mathcal{D}, \boldsymbol{\omega}) = Q(\boldsymbol{\lambda}_k \mid \mathcal{D}_{I_k(\mathbf{c})}, \boldsymbol{\omega}) \propto P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega}) \prod_{m \in I_k(\mathbf{c})} P(\boldsymbol{x}[m] \mid \boldsymbol{\lambda}_k),$$

that is, the posterior over $\boldsymbol{\lambda}_k$ *starting from the prior defined by* $\boldsymbol{\omega}$ *and conditioning on the data instances in* $I_k(\mathbf{c})$*.*

To apply Gibbs sampling, we also need to specify a distribution for sampling a new value for $C[m']$ *given* $\mathbf{c}_{-m'}$*, where again, we use the notation* $-m'$ *to indicate all values* $\{1, \ldots, M\} - \{m'\}$*. Similarly, let* $I_k(\mathbf{c}_{-m'})$ *denote the set of indexes* $\{m \neq m' : c[m] = k\}$*. Due to the*

independencies represented by the model structure, we have:

$$P(C[m'] = k \mid \mathbf{c}_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto \\ P(C[m'] = k \mid \mathbf{c}_{-m'}) P(\boldsymbol{x}[m'] \mid C[m'] = k, \boldsymbol{x}[I_k(\mathbf{c}_{-m'})], \boldsymbol{\omega}). \tag{19.9}$$

The second term on the right-hand side is simply a Bayesian prediction over $\boldsymbol{X}$ from the parameter posterior $Q(\boldsymbol{\lambda}_k \mid \mathcal{D}_{I_k(\mathbf{c}_{-m'})})$, as defined. Because of the symmetry of the parameters for the different clusters, the term does not depend on m' or on k, but only on the data set on which we condition. We can rewrite this term as $Q(\boldsymbol{X} \mid \mathcal{D}_{I_k(\mathbf{c}_{-m'})}, \boldsymbol{\omega})$. The first term on the right-hand side is the prior on cluster assignment for the instance m', as determined by the Dirichlet prior and the assignments of the other instances. Some algebra allows us to simplify this expression, resulting in:

$$P(C[m'] = k \mid \mathbf{c}_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto (|I_k(\mathbf{c}_{-m'})| + \alpha_0/K) Q(\boldsymbol{X} \mid \mathcal{D}_{I_k(\mathbf{c}_{-m'})}, \boldsymbol{\omega}). \tag{19.10}$$

Assuming we have a conjugate prior, this expression can be easily computed. Overall, for most conjugate priors, the cost of computing the sampling distribution for $C[m]$ in this model is $O(MK)$.■

It turns out that efficient sampling is also possible in more general models; see exercise 19.22. Alternatively we can use a Metropolis-Hastings approach where the proposal distribution can propose to modify several hidden values at once; see exercise 19.23.

Comparison Both types of collapsed particles can be useful for learning in practice, but they have quite different characteristics.

As we discussed, when we use parameter collapsed particles, the cost of evaluating a particle (for example, in a Metropolis-Hastings iteration) is determined by cost of inference in the network. In the worst case, this cost can be exponential, but in many examples it can be efficient. In contrast, the cost of evaluating data collapsed particles depends on properties of the prior. If the prior is properly chosen, the cost is linear in the size of the network.

Another aspect is the space in which we perform MCMC. In the case of parameter collapsed particles, the MCMC procedure is performing integration over a high-dimensional continuous space. The simple Metropolis-Hastings procedures we discussed in this book are usually quite poor for addressing this type of task. However, there is an extensive literature of more efficient MCMC procedures for this task (these are beyond the scope of this book). In the case of data collapsed particles, we perform the integration over parameters in closed form and use MCMC to explore the discrete (but exponential) space of assignments to the unobserved variables. In this problem, relatively simple MCMC methods, such as Gibbs sampling, can be fairly efficient.

To summarize, there is no clear choice between these two options. Both types of collapsed particles can speed up the convergence of the sampling procedure and the accuracy of the estimates of the parameters.

19.3.3 Variational Bayesian Learning

Another class of approximate inference procedures that we can apply to perform Bayesian inference in the case of incomplete data are variational approximations. Here, we can use the methods we developed in chapter 11 to the inference problem posed by Bayesian learning paradigm, resulting in an approach called *variational Bayes*. Recall that, in a variational approximation, we

variational Bayes

aim to find a distribution Q, from a predetermined family of distributions $\mathcal{Q}$, that is close to the real posterior distribution. In our case, we attempt to approximate $P(\mathcal{H}, \boldsymbol{\theta} \mid \mathcal{D})$; thus, the unnormalized measure $\tilde{P}$ in equation (11.3) is $P(\mathcal{H}, \boldsymbol{\theta}, \mathcal{D})$, and our approximating distribution Q is over the parameters and the hidden variables.

Plugging in the variational principle for our problem, and using the fact that $P(\mathcal{H}, \boldsymbol{\theta}, \mathcal{D}) = P(\boldsymbol{\theta})P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})$, we have that the energy functional takes the form:

$$F[P, Q] = \boldsymbol{E}_Q[\log P(\boldsymbol{\theta})] + \boldsymbol{E}_Q[\log P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})] + \boldsymbol{H}_Q(\boldsymbol{\theta}, \mathcal{H}).$$

The development of such an approximation requires that we decide on the class of approximate distributions we want to consider. While there are many choices here, a natural one is to decouple the posterior over the parameters from the posterior over the missing data. That is, assume that

$$Q(\boldsymbol{\theta}, \mathcal{H}) = Q(\boldsymbol{\theta})Q(\mathcal{H}). \tag{19.11}$$

This is clearly a nontrivial assumption, since our previous discussion shows that these two posteriors are coupled by the data. Nonetheless, we can hope that an approximation that decouples the two distributions will be more tractable.

Recall that, in our discussion of structured variational methods, we saw that the interactions between the structure of the approximation Q and the true distribution P can lead to further structural simplifications in Q (see section 11.5.2.4). Using these tools, we can find the following simplification.

Theorem 19.8

Let $P(\boldsymbol{\theta})$ be a parameter prior satisfying global parameter independence, $P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_{X_i|U_i})$. Let $\mathcal{D}$ be a partially observable IID data set. If we consider a variational approximation with distributions satisfying $Q(\boldsymbol{\theta}, \mathcal{H}) = Q(\boldsymbol{\theta})Q(\mathcal{H})$, then Q can be decomposed as

$$Q(\boldsymbol{\theta}, \mathcal{H}) = \prod_i Q(\boldsymbol{\theta}_{X_i|U_i}) \prod_m Q(\boldsymbol{h}[m]).$$

The proof is by direct application of proposition 11.7 and is left as an exercise (exercise 19.24).

This theorem shows that, once we decouple the posteriors over the parameters and missing data, we also lose the coupling between components of the two distributions (that is, different parameters or different instances). Thus, we can further decompose each of the two posteriors into a product of independent terms. This result matches our intuition, since the coupling between the parameters and missing data was the source of dependence between components of the two distributions. That is, the posteriors of two parameters were dependent due to incomplete data, and the posterior of missing data in two instances were dependent due to uncertainty about the parameters.

This theorem does not necessarily justify the (strong) assumption of equation (19.11), but it does suggest that it provides significant computational gains. In this case, we see that we can assume that the approximate posterior also satisfies global parameter independence, and similarly the approximate distribution over $\mathcal{H}$ consists of independent posteriors, one per instance. This simplification already makes the representation of Q much more tractable. Other simplifications, following the same logic, are also possible.

The variational Bayes approach often gives rise to very natural update rules.

Example 19.13

Consider again the Bayesian clustering model of section 19.2.2.4. In this case, we aim to represent the posterior over the parameters $\boldsymbol{\theta}_H, \boldsymbol{\theta}_{X_1|H}, \ldots, \boldsymbol{\theta}_{X_n|H}$ and over the hidden variables $H[1], \ldots, H[M]$. The decomposition of theorem 19.8 allows us write Q as a product distribution, with a term for each of these variables. Thus, we have that

$$Q = Q(\boldsymbol{\theta}_H) \left[\prod_i Q(\boldsymbol{\theta}_{X_i|H}) \right] \left[\prod_m Q(H[m]) \right].$$

mean field

This factorization is essentially a mean field *approximation. Using the results of section 11.5.1, we see that the fixed-point equations for this approximation are of the form*

$$\begin{aligned}
Q(\boldsymbol{\theta}_H) &\propto \exp\left\{ \ln P(\boldsymbol{\theta}_H) + \sum_m \boldsymbol{E}_{Q(H[m])}[\ln P(H[m] \mid \boldsymbol{\theta}_H)] \right\} \\
Q(\boldsymbol{\theta}_{X_i|H}) &\propto \exp\left\{ \ln P(\boldsymbol{\theta}_{X_i|H}) + \sum_m \boldsymbol{E}_{Q(H[m])}[\ln P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H})] \right\} \\
Q(H[m]) &\propto \exp\left\{ \boldsymbol{E}_{Q(\boldsymbol{\theta}_H)}[\ln P(H[m] \mid \boldsymbol{\theta}_H)] \right. \\
&\qquad \left. + \sum_i \boldsymbol{E}_{Q(\boldsymbol{\theta}_{X_i|H})}[\ln P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H})] \right\}.
\end{aligned}$$

The application of the mean-field theory allows us to identify the structure of the update equation. To provide a constructive solution, we also need to determine how to evaluate the expectations in these update equations. We now examine these expectations in the case where all the variables are binary and the priors over parameters are simple Dirichlet distributions (Beta distributions, in fact).

We start with the first fixed-point equation. A value for $\boldsymbol{\theta}_H$ is a pair $\langle \theta_{h^0}, \theta_{h^1} \rangle$. Using the definition of the Dirichlet prior, we have that

$$\ln P(\boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle) = \ln c + (\alpha_{h^0} - 1) \ln \theta_{h^0} + (\alpha_{h^1} - 1) \ln \theta_{h^1},$$

where α_{h^0} and α_{h^1} are the hyperparameters of the prior $P(\boldsymbol{\theta}_H)$, and c is the normalizing constant of the prior (which we can ignore). Similarly, we can see that

$$\boldsymbol{E}_{Q(H[m])}[\ln P(H[m] \mid \boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle)] = Q(H[m] = h^0) \ln \theta_{h^0} + Q(H[m] = h^1) \ln \theta_{h^1}.$$

Combining these results, we get that

$$\begin{aligned}
Q(\boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle) &\propto \exp\left\{ \left(\alpha_{h^0} + \sum_m Q(H[m] = h^0) - 1 \right) \ln \theta_{h^0} + \right. \\
&\qquad \left. \left(\alpha_{h^1} + \sum_m Q(H[m] = h^1) - 1 \right) \ln \theta_{h^1} \right\} \\
&= \theta_{h^0}^{\alpha_{h^0} + \sum_m Q(H[m]=h^0) - 1} \theta_{h^1}^{\alpha_{h^1} + \sum_m Q(H[m]=h^1) - 1}.
\end{aligned}$$

In other words, $Q(\boldsymbol{\theta}_H)$ is a Beta distribution with hyperparameters

$$\begin{aligned}\alpha'_{h^0} &= \alpha_{h^0} + \sum_m Q(H[m] = h^0)\\ \alpha'_{h^1} &= \alpha_{h^1} + \sum_m Q(H[m] = h^1).\end{aligned}$$

Note that this is exactly the Bayesian update for θ_H with the expected sufficient statistics given $Q(\mathcal{H})$.

A similar derivation shows that $Q(\boldsymbol{\theta}_{X_i|H})$ is also a pair of independent Beta distributions (one for each value of H) that are updated with the expected sufficient statistics given $Q(\mathcal{H})$.

M-step

These updates are reminiscent of the EM-update (M-step)*, since we use expected sufficient statistics to update the posterior. In the EM M-step, we update the MLE using the expected sufficient statistics. If we carry the analogy further, the last fixed-point equation, which updates $Q(H[m])$, corresponds*

E-step

to the E-step, since it updates the expectations over the missing values. Recall that, in the E-step *of EM, we use the current parameters to compute*

$$Q(H[m]) = P(H[m] \mid x_1[m], \ldots x_n[m]) \propto P(H[m] \mid \boldsymbol{\theta}_H) \prod_i P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H}).$$

If we were doing a Bayesian approach, we would not simply take our current values for the parameters $\boldsymbol{\theta}_H, \boldsymbol{\theta}_{X_i|H}$; rather, we would average over their posteriors. Examining this last fixed-point equation, we see that we indeed average over the (approximate) posteriors $Q(\boldsymbol{\theta}_H)$ and $Q(\boldsymbol{\theta}_{X_i|H})$. However, unlike standard Bayesian averaging, where we compute the average value of the parameter itself, here we average its logarithm; that is, we evaluate terms of the form

$$\boldsymbol{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\left[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\right] = \int_0^1 Q(\theta_{x_i|H[m]}) \ln \theta_{x_i|H[m]} d\theta_{x_i|H[m]}.$$

Using methods that are beyond the scope of this book, one can show that this integral has a closed-form solution:

$$\boldsymbol{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\left[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\right] = \varphi(\alpha'_{x_i|h}) - \varphi(\sum_{x'_i} \alpha'_{x'_i|h}),$$

where α' are the hyperparameters of the posterior approximation in $Q(\boldsymbol{\theta}_{X_i|H})$ and $\varphi(z) = (\ln \Gamma(z))' = \frac{\Gamma'(z)}{\Gamma(z)}$ is the digamma function*, which is equal to $\ln(z)$ plus a polynomial function of $\frac{1}{z}$. And so, for $z \gg 1$, $\varphi(z) \approx \ln(z)$. Using this approximation, we see that*

digamma function

$$\boldsymbol{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\left[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\right] \approx \ln \frac{\alpha'_{x_i|h}}{\sum_{x'_i} \alpha'_{x'_i|h}},$$

that is, the logarithm of the expected conditional probability according to the posterior $Q(\boldsymbol{\theta}_{X_i|H})$. This shows that if the posterior hyperparameters are large the variational update is almost identical to EM's E-step.

To wrap up, we applied the structured variational approximation to the Bayesian learning problem. Using the tools we developed in previous chapters, we defined tractable fixed-point equations.

As with the mean field approximation we discussed in section 11.5.1, we can find a fixed-point solution for Q by iteratively applying these equations.

The resulting algorithm is very similar to applications of EM. Applications of the update equations for the parameters are almost identical to standard EM of section 19.2.2.4 in the sense that we use expected sufficient statistics. However, instead of finding the MLE parameters given these expected sufficient statistics, we compute the posterior assuming these were observed. The update for $Q(H[m])$ is reminiscent to the computation of $P(H[m])$ when we know the parameters. However, instead of using parameter values we use expectations of their logarithm and then take the exponent. ■

This example shows that we can find a variational approximation to the Bayesian posterior using an EM-like algorithm in which we iterate between updates to the parameter posteriors and updates to the missing data posterior. These ideas generalize to other network structures in a fairly straightforward way. The update for the posterior over parameter is similar to Bayesian update with expected sufficient statistics, and the update of the posterior over hidden variable is similar to a computation with the expected parameters (with the differences discussed earlier). In more complex examples we might need to make further assumptions about the distribution Q in order to get a tractable approximation. For example, if there are multiple missing values per instance, then we might not be able to afford to represent their distribution by the joint distribution and would instead need to introduce structure into Q. The basic ideas are similar to ones we explored before, and so we do not elaborate them. See exercise 15.6 for one example.

Of course, this method has some clear drawbacks. Because we are representing the parameter posterior by a factored distribution, we cannot expect to represent a multimodal posterior. Unfortunately, we know that the posterior is often multimodal. For example, in the clustering problem, we know that change in names of values of H would not change the prediction. Thus, the posterior in this example should be symmetric under such renaming. This implies that a unimodal distribution can only be a partial approximation to the true posterior. In multimodal cases, the effect of the variational approximation cannot be predicted. It may select one of the peaks and try to approximate it using Q, or it may choose a "broad" distribution that averages over some or all of the peaks.

19.4 Structure Learning

We now move to discuss the more complex task of learning the network structure as well as the parameters, again in the presence of incomplete data. Recall that in the case of complete data, we started by defining a score for evaluating different network structures and then examined search procedures that can maximize this score. As we will see, **both components of structure learning — the scoring function and the search procedure — are considerably more complicated in the case of incomplete data. Moreover, in the presence of hidden variables, even our search space becomes significantly more complex, since we now have to select the value space for the hidden variables, and even the number of hidden variables that the model contains.**