global acyclicity constraint in undirected models. Recall (see theorem 18.5) that the acyclicity constraint couples decisions regarding the family of different variables, thereby making the structure selection problem much harder. The lack of such a global constraint in the undirected case eliminates these interactions, allowing us to choose the local structure locally in different parts of the network. In particular, it turns out that a particular variant of the structure learning task can be formulated as a continuous, convex optimization problem, a class of problems generally viewed as tractable. Thus, elimination of global acyclicity removes the main reason for the $\mathcal{NP}$-hardness of structure learning that we saw in Bayesian networks. However, this does *not* make structure learning of Markov networks efficient; the convex optimization process (as for parameter estimation) still requires multiple executions of inference over the network.

A final important issue that arises in the context of Markov networks is the overwhelmingly common use of these networks for settings, such as image segmentation and others, where we have a particular inference task in mind. In these settings, we often want to train a network *discriminatively* (see section 16.3.2), so as to provide good performance for our particular prediction task. Indeed, much of Markov network learning is currently performed for CRFs.

The remainder of this chapter is structured as follows. We begin with the analysis of the properties of the likelihood function, which, as always, forms the basis for all of our discussion of learning. We then discuss how the likelihood function can be optimized to find the maximum likelihood parameter estimates. The ensuing sections discuss various important extensions to these basic ideas: conditional training, parameter priors for MAP estimation, structure learning, learning with missing data, and approximate learning methods that avoid the computational bottleneck of multiple iterations of network inference. These extensions are usually described as building on top of standard maximum-likelihood parameter estimation. However, it is important to keep in mind that they are largely orthogonal to each other and can be combined. Thus, for example, we can also use the approximate learning methods in the case of structure learning or of learning with missing data. Similarly, all of the methods we described can be used with maximum conditional likelihood training. We return to this issue in section 20.8.

We note that, for convenience and consistency with standard usage, we use natural logarithms throughout this chapter, including in our definitions of entropy or KL-divergence.

20.2 The Likelihood Function

As we saw in earlier chapters, the key component in most learning tasks is the likelihood function. In this section, we discuss the form of the likelihood function for Markov networks, its properties, and their computational implications.

20.2.1 An Example

As we suggested, the existence of a global partition function couples the different parameters in a Markov network, greatly complicating our estimation problem. To understand this issue, consider the very simple network A—B—C, parameterized by two potentials $\phi_1(A, B)$ and $\phi_2(B, C)$. Recall that the log-likelihood of an instance $\langle a, b, c\rangle$ is

$$\ln P(a, b, c) = \ln \phi_1(a, b) + \ln \phi_2(b, c) - \ln Z,$$

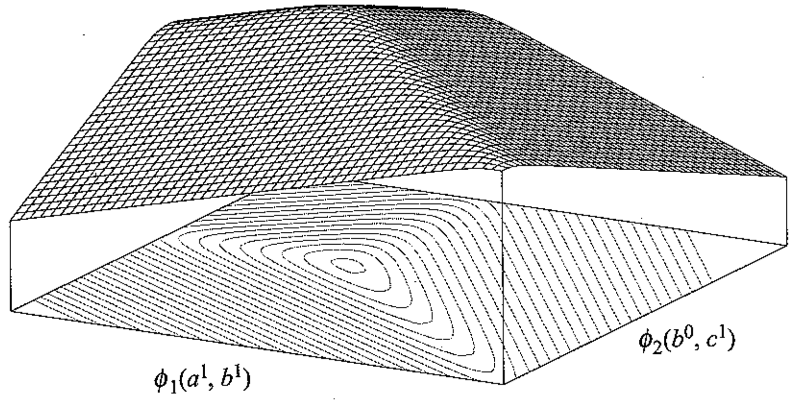

Figure 20.1 Log-likelihood surface for the Markov network A—B—C, as a function of $\ln \phi_1(a^1, b^1)$ **(x-axis) and** $\ln \phi_2(b^0, c^1)$ **(y-axis)**; all other parameters in both potentials are set to 1. The data set $\mathcal{D}$ has $M = 100$ instances, for which $M[a^1, b^1] = 40$ and $M[b^0, c^1] = 40$. (The other sufficient statistics are irrelevant, since all of the other log-parameters are 0.)

where Z is the partition function that ensures that the distribution sums up to one. Now, consider the log-likelihood function for a data set $\mathcal{D}$ containing M instances:

$$\begin{aligned}\ell(\boldsymbol{\theta} : \mathcal{D}) &= \sum_m (\ln \phi_1(a[m], b[m]) + \ln \phi_2(b[m], c[m]) - \ln Z) \\ &= \sum_{a,b} M[a,b] \ln \phi_1(a,b) + \sum_{b,c} M[b,c] \ln \phi_2(b,c) - M \ln Z(\boldsymbol{\theta}).\end{aligned}$$

Thus, we have sufficient statistics that summarize the data: the joint counts of variables that appear in each potential. This is analogous to the situation in learning Bayesian networks, where we needed the joint counts of variables that appear within the same family. This likelihood consists of three terms. The first term involves ϕ_1 alone, and the second term involves ϕ_2 alone. The third term, however, is the log-partition function $\ln Z$, where:

$$Z(\boldsymbol{\theta}) = \sum_{a,b,c} \phi_1(a,b)\phi_2(b,c).$$

Thus, $\ln Z(\boldsymbol{\theta})$ is a function of both ϕ_1 and ϕ_2. As a consequence, it *couples* the two potentials in the likelihood function.

Specifically, consider maximum likelihood estimation, where we aim to find parameters that maximize the log-likelihood function. In the case of Bayesian networks, we could estimate each conditional distribution independently of the other ones. Here, however, when we change one of the potentials, say ϕ_1, the partition function changes, possibly changing the value of ϕ_2 that maximizes $-\ln Z(\boldsymbol{\theta})$. Indeed, as illustrated in figure 20.1, the log-likelihood function in our simple example shows clear dependencies between the two potentials.

In this particular example, we can avoid this problem by noting that the network A—B—C is equivalent to a Bayesian network, say $A \rightarrow B \rightarrow C$. Therefore, we can learn the parameters of this BN, and then define $\phi_1(A, B) = P(A)P(B \mid A)$ and $\phi_2(B, C) = P(C \mid B)$. Because the two representations have equivalent expressive power, the same maximum likelihood is achievable in both, and so the resulting parameterization for the Markov network will also be a maximum-likelihood solution. In general, however, there are Markov networks that do not have an equivalent BN structure, for example, the diamond-structured network of figure 4.13 (see section 4.5.2). In such cases, we generally cannot convert a learned BN parameterization into an equivalent MN; indeed, the optimal likelihood achievable in the two representations is generally not the same.

20.2.2 Form of the Likelihood Function

log-linear model

To provide a more general description of the likelihood function, it first helps to provide a more convenient notational basis for the parameterization of these models. For this purpose, we use the framework of *log-linear models*, as defined in section 4.4.1.2. Given a set of features $\mathcal{F} = \{f_i(\boldsymbol{D}_i)\}_{i=1}^k$, where $f_i(\boldsymbol{D}_i)$ is a feature function defined over the variables in $\boldsymbol{D}_i$, we have:

$$P(X_1, \ldots, X_n : \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{\sum_{i=1}^{k} \theta_i f_i(\boldsymbol{D}_i)\right\}. \tag{20.1}$$

As usual, we use $f_i(\xi)$ as shorthand for $f_i(\xi\langle\boldsymbol{D}_i\rangle)$. The parameters of this distribution correspond to the weight we put on each feature. When $\theta_i = 0$, the feature is ignored, and it has no effect on the distribution.

As discussed in chapter 4, this representation is very generic and can capture Markov networks with global structure and local structure. A special case of particular interest is when $f_i(\boldsymbol{D}_i)$ is a binary indicator function that returns the value 0 or 1. With such features, we can encode a "standard" Markov network by simply having one feature per potential entry. In more general, however, we can consider arbitrary valued features.

Example 20.1

As a specific example, consider the simple diamond network of figure 3.10c, where we take all four variables to be binary-valued. The features that correspond to this network are sixteen indicator functions: four for each assignment of variables to each of our four clusters. For example, one such feature would be:

$$f_{a^0,b^0}(a, b) = \boldsymbol{1}\{a = a^0\}\boldsymbol{1}\{b = b^0\}.$$

With this representation, the weight of each indicator feature is simply the natural logarithm of the corresponding potential entry. For example, $\theta_{a^0,b^0} = \ln \phi_1(a^0, b^0)$. ■

Given a model in this form, the log-likelihood function has a simple form.

Proposition 20.1

Let $\mathcal{D}$ be a data set of M examples, and let $\mathcal{F} = \{f_i : i = 1, \ldots, k\}$ be a set of features that define a model. Then the log-likelihood is

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \left(\sum_m f_i(\xi[m])\right) - M \ln Z(\boldsymbol{\theta}). \tag{20.2}$$

sufficient statistics

The *sufficient statistics* of this likelihood function are the sums of the feature values in the instances in $\mathcal{D}$. We can derive a more elegant formulation if we divide the log-likelihood by the number of samples M.

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{d}_i)] - \ln Z(\boldsymbol{\theta}), \tag{20.3}$$

where $\boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{d}_i)]$ is the empirical expectation of f_i, that is, its average in the data set.

20.2.3 Properties of the Likelihood Function

The formulation of proposition 20.1 describes the likelihood function as a sum of two functions. The first function is linear in the parameters; increasing the parameters directly increases this linear term. Clearly, because the log-likelihood function (for a fixed data set) is upper-bounded (the probability of an event is at most 1), the second term $\ln Z(\boldsymbol{\theta})$ balances the first term.

Let us examine this second term in more detail. Recall that the partition function is defined as

$$\ln Z(\boldsymbol{\theta}) = \ln \sum_\xi \exp\left\{\sum_i \theta_i f_i(\xi)\right\}.$$

convex partition function

One important property of the partition function is that it is *convex* in the parameters $\boldsymbol{\theta}$. Recall that a function $f(\vec{x})$ is convex if for every $0 \geq \alpha \geq 1$,

$$f(\alpha\vec{x} + (1-\alpha)\vec{y}) \leq \alpha f(\vec{x}) + (1-\alpha) f(\vec{y}).$$

Hessian

In other words, the function is bowl-like, and every interpolation between the images of two points is larger than the image of their interpolation. One way to prove formally that the function f is convex is to show that the *Hessian* — the matrix of the function's second derivatives — is positive semidefinite. Therefore, we now compute the derivatives of $Z(\boldsymbol{\theta})$.

Proposition 20.2

Let $\mathcal{F}$ be a set of features. Then,

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i] \\ \frac{\partial^2}{\partial\theta_i\partial\theta_j}\ln Z(\boldsymbol{\theta}) &= \boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j],\end{aligned}$$

where $\boldsymbol{E}_{\boldsymbol{\theta}}[f_i]$ is a shorthand for $\boldsymbol{E}_{P(\mathcal{X}:\boldsymbol{\theta})}[f_i]$.

Proof The first derivatives are computed as:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})}\sum_\xi \frac{\partial}{\partial\theta_i}\exp\left\{\sum_j \theta_j f_j(\xi)\right\} \\ &= \frac{1}{Z(\boldsymbol{\theta})}\sum_\xi f_i(\xi)\exp\left\{\sum_j \theta_j f_j(\xi)\right\} \\ &= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i].\end{aligned}$$

We now consider the second derivative:

$$\begin{aligned}
\frac{\partial^2}{\partial\theta_j\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{\partial}{\partial\theta_j}\left[\frac{1}{Z(\boldsymbol{\theta})}\sum_\xi f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\}\right] \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}\left(\frac{\partial}{\partial\theta_j}Z(\boldsymbol{\theta})\right)\sum_\xi f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&\quad +\frac{1}{Z(\boldsymbol{\theta})}\sum_\xi f_i(\xi)f_j(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}Z(\boldsymbol{\theta})\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_\xi f_i(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&\quad +\frac{1}{Z(\boldsymbol{\theta})}\sum_\xi f_i(\xi)f_j(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&= -\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_\xi f_i(\xi)P(\xi:\boldsymbol{\theta}) \\
&\quad +\sum_\xi f_i(\xi)f_j(\xi)P(\xi:\boldsymbol{\theta}) \\
&= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i f_j] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]\boldsymbol{E}_{\boldsymbol{\theta}}[f_j] \\
&= \boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j].
\end{aligned}$$

Thus, the Hessian of $\ln Z(\boldsymbol{\theta})$ is the covariance matrix of the features, viewed as random variables distributed according to distribution defined by $\boldsymbol{\theta}$. Because a covariance matrix is always positive semidefinite, it follows that the Hessian is positive semidefinite, and hence that $\ln Z(\boldsymbol{\theta})$ is a convex function of $\boldsymbol{\theta}$. ■

Because $\ln Z(\boldsymbol{\theta})$ is convex, its complement $(-\ln Z(\boldsymbol{\theta}))$ is concave. The sum of a linear function and a concave function is concave, implying the following important result:

Corollary 20.1 *The log-likelihood function is concave.*

This result implies that the log-likelihood is unimodal and therefore has no local optima. It does not, however, imply the uniqueness of the global optimum: Recall that a parameterization of the Markov network can be *redundant*, giving rise to multiple representations of the same distribution. The standard parameterization of a set of table factors for a Markov network — a feature for every entry in the table — is always redundant. In our simple example, for instance, we have:

redundant parameterization

$$f_{a^0,b^0} = 1 - f_{a^0,b^1} - f_{a^1,b^0} - f_{a^1,b^1}.$$

We thus have a continuum of parameterizations that all encode the same distribution, and (necessarily) give rise to the same log-likelihood. Thus, there is a unique globally optimal value for the log-likelihood function, but not necessarily a unique solution. In general, because the function is concave, we are guaranteed that there is a convex region of continuous global optima.

It is possible to eliminate the redundancy by removing some of the features. However, as we discuss in section 20.4, that turns out to be unnecessary, and even harmful, in practice.

We note that we have defined the likelihood function in terms of a standard log-linear parameterization, but the exact same derivation also holds for networks that use shared parameters, as in section 6.5; see exercise 20.1 and exercise 20.2.

20.3 Maximum (Conditional) Likelihood Parameter Estimation

We now move to the question of estimating the parameters of a Markov network with a fixed structure, given a fully observable data set $\mathcal{D}$. We focus in this section on the simplest variant of this task — maximum-likelihood parameter estimation, where we select parameters that maximize the log-likelihood function of equation (20.2). In later sections, we discuss alternative objectives for the parameter estimation task.

20.3.1 Maximum Likelihood Estimation

As for any function, the gradient of the log-likelihood must be zero at its maximum points. For a concave function, the maxima are precisely the points at which the gradient is zero. Using proposition 20.2, we can compute the gradient of the average log-likelihood as follows:

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]. \tag{20.4}$$

This analysis provides us with a precise characterization of the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$:

Theorem 20.1 *Let $\mathcal{F}$ be a set of features. Then, $\boldsymbol{\theta}$ is a maximum-likelihood parameter assignment if and only if $\boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] = \boldsymbol{E}_{\hat{\boldsymbol{\theta}}}[f_i]$ for all i.*

expected sufficient statistics

moment matching

MLE consistency

In other words, at the maximal likelihood parameters $\hat{\boldsymbol{\theta}}$, the expected value of each feature relative to $P_{\hat{\boldsymbol{\theta}}}$ matches its empirical expectation in $\mathcal{D}$. In other words, we want the *expected sufficient statistics* in the learned distribution to match the empirical expectations. This type of equality constraint is also called *moment matching*. This theorem easily implies that maximum likelihood estimation is *consistent* in the same sense as definition 18.1: if the model is sufficiently expressive to capture the data-generating distribution, then, at the large sample limit, the optimum of the pseudolikelihood objective is the true model; see exercise 20.3.

By itself, this criterion does not provide a constructive definition of the maximum likelihood parameters. **Unfortunately, although the function is concave, there is no analytical form for its maximum. Thus, we must resort to iterative methods that search for the global optimum. Most commonly used are the gradient ascent methods reviewed in appendix A.5.2, which iteratively take steps in parameter space to improve the objective.** At each iteration, they compute the gradient, and possibly the Hessian, at the current point $\boldsymbol{\theta}$, and use those estimates to approximate the function at the current neighborhood. They then take a step in the right direction (as dictated by the approximation) and repeat the process. Due to the convexity of the problem, this process is guaranteed to converge to a global optimum, regardless of our starting point.

To apply these gradient-based methods, we need to compute the gradient. Fortunately, equation (20.4) provides us with an exact formula for the gradient: the difference between the feature's empirical count in the data and its expected count relative to our current parameterization $\boldsymbol{\theta}$. For example, consider again the fully parameterized network of example 20.1. Here, the features are simply indicator functions; the empirical count for a feature such as $f_{a^0,b^0}(a,b) = \boldsymbol{1}\{a = a^0\}\boldsymbol{1}\{b = b^0\}$ is simply the empirical frequency, in the data set $\mathcal{D}$, of the event a^0, b^0. At a particular parameterization $\boldsymbol{\theta}$, the expected count is simply $P_{\boldsymbol{\theta}}(a^0, b^0)$. Very naturally, the gradient for the parameter associated with this feature is the difference between these two numbers.

However, this discussion ignores one important aspect: the computation of the expected counts. In our example, for instance, we must compute the different probabilities of the form $P_{\boldsymbol{\theta}^t}(a, b)$. Clearly, this computation requires that we run inference over the network. As for the case of EM in Bayesian networks, a feature is necessarily part of a factor in the original network, and hence, due to family preservation, all of the variables involved in a feature must occur together in a cluster in a clique tree or cluster graph. Thus, a single inference pass that calibrates an entire cluster graph or tree suffices to compute all of the expected counts. Nevertheless, **a full inference step is required at every iteration of the gradient ascent procedure. Because inference is almost always costly in time and space, the computational cost of parameter estimation in Markov networks is usually high, sometimes prohibitively so.** In section 20.5 we return to this issue, considering the use of approximate methods that reduce the computational burden.

Our discussion does not make a specific choice of algorithm to use for the optimization. In practice, standard gradient ascent is not a particularly good algorithm, both because of its slow convergence rate and because of its sensitivity to the step size. Much faster convergence is obtained with second-order methods, which utilize the Hessian to provide a quadratic approximation to the function. However, from proposition 20.2 we can conclude that the *Hessian* of the log-likelihood function has the form:

log-likelihood Hessian

$$\frac{\partial}{\partial\theta_i\partial\theta_j}\ell(\boldsymbol{\theta} : \mathcal{D}) = -M\,\boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j]. \tag{20.5}$$

To compute the Hessian, we must compute the joint expectation of two features, a task that is often computationally infeasible. Currently, one commonly used solution is the *L-BFGS algorithm*, a gradient-based algorithm that uses line search to avoid computing the Hessian (see appendix A.5.2 for some background).

L-BFGS algorithm

20.3.2 Conditionally Trained Models

As we discussed in section 16.3.2, we often want to use a Markov network to perform a particular inference task, where we have a known set of observed variables, or features, $\boldsymbol{X}$, and a predetermined set of variables, $\boldsymbol{Y}$, that we want to query. In this case, we may prefer to use *discriminative training*, where we train the network as a *conditional random field* (CRF) that encodes a conditional distribution $P(\boldsymbol{Y} \mid \boldsymbol{X})$.

discriminative training

conditional random field

More formally, in this setting, our training set consists of pairs $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^{M}$, specifying assignments to $\boldsymbol{Y}, \boldsymbol{X}$. An appropriate objective function to use in this situation is the *conditional likelihood* or its logarithm, defined in equation (16.3). In our setting, the

conditional likelihood

log-conditional-likelihood has the form:

$$\ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) = \ln P(\boldsymbol{y}[1, \ldots, M] \mid \boldsymbol{x}[1, \ldots, M], \boldsymbol{\theta}) = \sum_{m=1}^{M} \ln P(\boldsymbol{y}[m] \mid \boldsymbol{x}[m], \boldsymbol{\theta}). \qquad (20.6)$$

In this objective, we are optimizing the likelihood of each observed assignment $\boldsymbol{y}[m]$ given the corresponding observed assignment $\boldsymbol{x}[m]$. Each of the terms $\ln P(\boldsymbol{y}[1, \ldots, M] \mid \boldsymbol{x}[1, \ldots, M], \boldsymbol{\theta})$ is a log-likelihood of a Markov network model with a different set of factors — the factors in the original network, reduced by the observation $\boldsymbol{x}[1, \ldots, M]$ — and its own partition function. Each term is thereby a concave function, and because the sum of concave functions is concave, we conclude:

Corollary 20.2 *The log conditional likelihood of equation (20.6) is a concave function.*

As for corollary 20.1, this result implies that the function has a global optimum and no local optima, but not that the global optimum is unique. Here also, redundancy in the parameterization may give rise to a convex region of contiguous global optima.

The approaches for optimizing this objective are similar to those used for optimizing the likelihood objective in the unconditional case. The objective function is a concave function, and so a gradient ascent process is guaranteed to give rise to the unique global optimum. The form of the gradient here can be derived directly from equation (20.4). We first observe that the gradient of a sum is the sum of the gradients of the individual terms. Here, each term is, in fact, a log-likelihood — the log-likelihood of a single data case $\boldsymbol{y}[m]$ in the Markov network obtained by reducing our original model to the context $\boldsymbol{x}[m]$. A reduced Markov network is itself a Markov network, and so we can apply equation (20.4) and conclude that:

$$\frac{\partial}{\partial \theta_i} \ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_{m=1}^{M} \left(f_i(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i \mid \boldsymbol{x}[m]] \right). \qquad (20.7)$$

This solution looks deceptively similar to equation (20.4). Indeed, if we aggregate the first component in each of the summands, we obtain precisely the empirical count of f_i in the data set $\mathcal{D}$. There is, however, one key difference. In the unreduced Markov network, the expected feature counts are computed relative to a single model; in the case of the conditional Markov network, these expected counts are computed as the summation of counts in an ensemble of models, defined by the different values of the conditioning variables $\boldsymbol{x}[m]$. This difference has significant computational consequences. Recall that computing these expectations involves running inference over the model. **Whereas in the unconditional case, each gradient step required only a single execution of inference, when training a CRF, we must (in general) execute inference *for every single data case*, conditioning on $\boldsymbol{x}[m]$. On the other hand, the inference is executed on a simpler model, since conditioning on evidence in a Markov network can only reduce the computational cost.** For example, the network of figure 20.2 is very densely connected, whereas the reduced network over $\boldsymbol{Y}$ alone (conditioned on $\boldsymbol{X}$) is a simple chain, allowing linear-time inference.

Discriminative training can be particularly beneficial in cases where the domain of $\boldsymbol{X}$ is very large or even infinite. For example, in our image classification task, the partition function in the

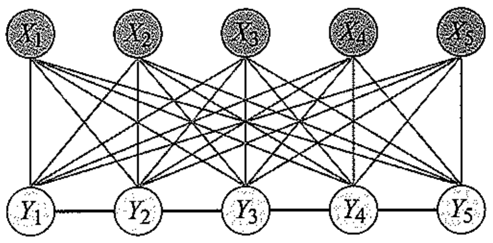

Figure 20.2 A highly connected CRF that allows simple inference when conditioned: The edges that disappear in the reduced Markov network after conditioning on $\boldsymbol{X}$ are marked in gray; the remaining edges form a simple linear chain.

generative setting involves summation (or integration) over the space of all possible images; if we have an $N \times N$ image where each pixel can take 256 values, the resulting space has 256^{N^2} values, giving rise to a highly intractable inference problem (even using approximate inference methods).

Box 20.A — Concept: Generative and Discriminative Models for Sequence Labeling. *One of the main tasks to which probabilistic graphical models have been applied is that of taking a set of interrelated instances and jointly labeling them, a process sometimes called* collective classification. *We have already seen examples of this task in box 4.B and in box 4.E; many other examples exist. Here, we discuss some of the trade-offs between different models that one can apply to this task. We focus on the context of labeling instances organized in a sequence, since it is simpler and allows us to illustrate another important point.*

collective classification

In the sequence labeling *task, we get as input a sequence of observations $\boldsymbol{X}$ and need to label each of them. For example, in text analysis (box 4.E), we might have a sequence of words each of which we want to label with some label. In a task of* activity recognition, *we might obtain a sequence of images and want to label each frame with the activity taking place in it (for example, running, jumping, walking). We assume that we want to construct a model for this task and to train it using fully labeled training data, where both $\boldsymbol{Y}$ and $\boldsymbol{X}$ are observed.*

sequence labeling

activity recognition

Figure 20.A.1 illustrates three different types of models that have been proposed and used for sequence labeling, all of which we have seen earlier in this book (see figure 6.2 and figure 4.14). The first model is a hidden Markov model *(or HMM), which is a purely generative model: the model generates both the labels $\boldsymbol{Y}$ and the observations $\boldsymbol{X}$. The second is called a* maximum entropy Markov model *(or MEMM). This model is also directed, but it represents a conditional distribution $P(\boldsymbol{Y} \mid \boldsymbol{X})$; hence, there is no attempt to model a distribution over the $\boldsymbol{X}$'s. The final model is the* conditional random field *(or CRF) of section 4.6.1. This model also encodes a conditional distribution; hence the arrows from $\boldsymbol{X}$ to $\boldsymbol{Y}$. However, here the interactions between the $\boldsymbol{Y}$ are modeled as undirected edges.*

hidden Markov model

maximum entropy Markov model

conditional random field

These different models present interesting trade-offs in terms of their expressive power and learnability. First, from a computational perspective, HMMs and MEMMs are much more easily learned. As purely directed models, their parameters can be computed in closed form using either maximum-likelihood or Bayesian estimation (see chapter 17); conversely, the CRF requires that we use an

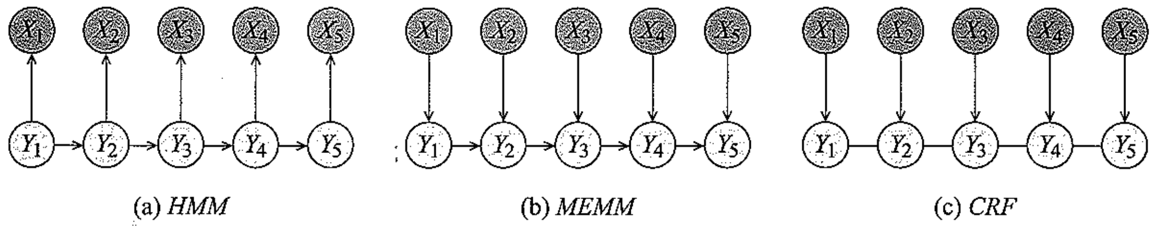

Figure 20.A.1 — Different models for sequence labeling: HMM, MEMM, and CRF

iterative gradient-based approach, which is considerably more expensive (particularly here, when inference must be run separately for every training sequence; see section 20.3.2).

A second important issue relates to our ability to use a rich feature set. As we discussed in example 16.3 and in box 4.E, our success in a classification task often depends strongly on the quality of our features. In an HMM, we must explicitly model the distribution over the features, including the interactions between them. This type of model is very hard, and often impossible, to construct correctly. The MEMM and the CRF are both discriminative models, and therefore they avoid this challenge entirely.

The third and perhaps subtler issue relates to the independence assumptions made by the model. As we discussed in section 4.6.1.2, the MEMM makes the independence assumption that $(Y_i \perp X_j \mid \boldsymbol{X}_{-j})$ *for any* $j > i$. *Thus, an observation from later in the sequence has absolutely no effect on the posterior probability of the current state; or, in other words, the model does not allow for any smoothing. The implications of this can be severe in many settings. For example, consider the task of activity recognition from a video sequence; here, we generally assume that activities are highly persistent: if a person is running in one frame, she is also extremely likely to be running in the next frame. Now, imagine that the person starts running, but our first observation in the sequence is ambiguous and consistent with both running and walking. The model will pick one — the one whose probability given that one frame is highest, a decision that may well be wrong. Assuming that activities are persistent, this choice of activity is likely to stay high for a large number of steps; the posterior of the initial activity will never change. In other words, the best we can expect is a prediction where the initial activity is running, and then (perhaps) transitions to walking. The model is incapable of going back and changing its prediction about the first few frames. This problem has been called the* label bias *problem.*

label bias problem

To summarize, the trade-offs between these different models are subtle and non-definitive. In cases where we have many correlated features, discriminative models are probably better; but, if only limited data are available, the stronger bias of the generative model may dominate and allow learning with fewer samples. Among the discriminative models, MEMMs should probably be avoided in cases where many transitions are close to deterministic. In many cases, CRFs are likely to be a safer choice, but the computational cost may be prohibitive for large data sets.

20 Learning Undirected Models

20.1 Overview

In previous chapters, we developed the theory and algorithms for learning Bayesian networks from data. In this chapter, we consider the task of learning Markov networks. Although many of the same concepts and principles arise, the issues and solutions turn out to be quite different.

Perhaps the most important reason for the differences is a key distinction between Markov networks and Bayesian networks: the use of a global normalization constant (the partition function) rather than local normalization within each CPD. This global factor couples all of the parameters across the network, preventing us from decomposing the problem and estimating local groups of parameters separately. This global parameter coupling has significant computational ramifications. As we will explain, in contrast to the situation for Bayesian networks, even simple (maximum-likelihood) parameter estimation with complete data cannot be solved in closed form (except for chordal Markov networks, which are therefore also Bayesian networks). Rather, we generally have to resort to iterative methods, such as gradient ascent, for optimizing over the parameter space. The good news is that the likelihood objective is concave, and so these methods are guaranteed to converge to the global optimum. The bad news is that each of the steps in the iterative algorithm requires that we run inference on the network, making even simple parameter estimation a fairly expensive, or even intractable, process. Bayesian estimation, which requires integration over the space of parameters, is even harder, since there is no closed-form expression for the parameter posterior. Thus, the integration associated with Bayesian estimation must be performed using approximate inference (such as variational methods or MCMC), a burden that is often infeasible in practice.

As a consequence of these computational issues, much of the work in this area has gone into the formulation of alternative, more tractable, objectives for this estimation problem. Other work has been focused on the use of approximate inference algorithms for this learning problem and on the development of new algorithms suited to this task.

The same issues have significant impact on structure learning. In particular, because a Bayesian parameter posterior is intractable to compute, the use of exact Bayesian scoring for model selection is generally infeasible. In fact, scoring any model (computing the likelihood) requires that we run inference to compute the partition function, greatly increasing the cost of search over model space. Thus, here also, the focus has been on approximations and heuristics that can reduce the computational cost of this task. Here, however, there is some good news, arising from another key distinction between Bayesian and Markov networks: the lack of a