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

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

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

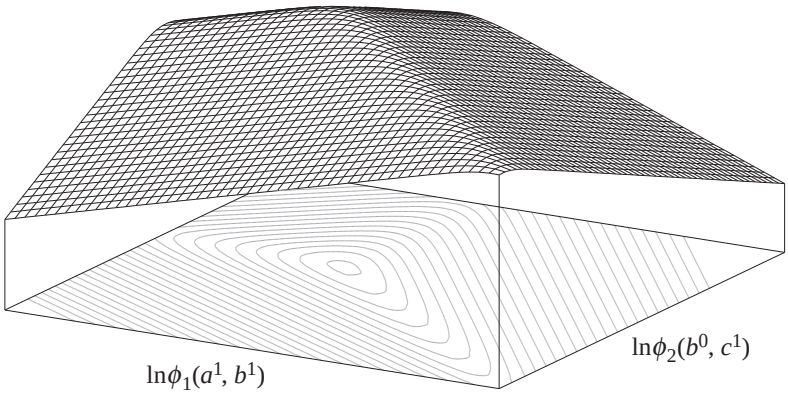


Figure 20.1 Log-likelihood surface for the Markov network A—B—C, as a function of $\ln \phi_1(a^1, b^1)$ (x-axis) and $\ln \phi_2(b^0, c^1)$ (y-axis); all other parameters in both potentials are set to 1. Surface is viewed from the $(+\infty, +\infty)$ point toward the $(-, -)$ quadrant. The data set $\mathcal{D}$ 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:

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

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 $A \to B \to C$. 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

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:

log-linear model

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

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 3.10a, 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:*

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

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 $0 \leq \alpha \leq 1$,

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

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 $Z(\boldsymbol{\theta})$.

Hessian

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.*

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 likelihood objective is the true model; see exercise 20.3.

expected sufficient statistics

moment matching

MLE consistency

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

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

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:

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

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:

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

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

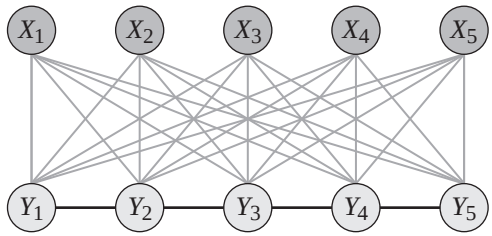


Figure 20.2 A highly connected CRF that allows simple inference when conditioned: The edges that disappear in the reduced Markov network after conditioning on $\boldsymbol{X}$ are marked in gray; the remaining edges form a simple linear chain.

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

sequence labeling

activity recognition

In the sequence labeling *task, we get as input a sequence of observations $\boldsymbol{X}$ and need to label them with some joint label $\boldsymbol{Y}$. 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.*

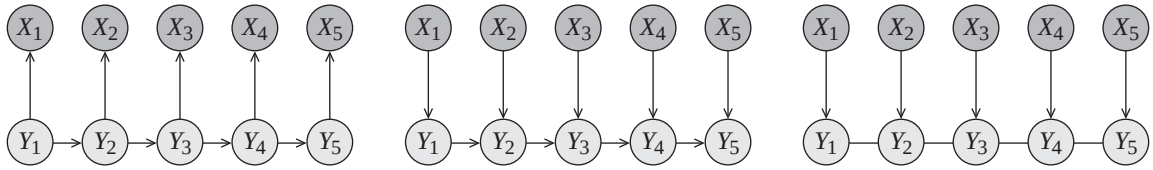
hidden Markov model

maximum entropy Markov model

conditional random field

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.*

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



(a) *HMM* (b) *MEMM* (c) *CRF*

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 walking in one frame, she is also extremely likely to be walking in the next frame. Now, imagine that the person starts running, but our first few observations in the sequence are ambiguous and consistent with both running and walking. The model will pick one — the one whose probability given that one frame is highest — which may well be walking. 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 walking, and then (perhaps) transitions to running. 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.

20.3.3 Learning with Missing Data

We now turn to the problem of parameter estimation in the context of missing data. As we saw in section 19.1, the introduction of missing data introduces both conceptual and technical difficulties. In certain settings, we may need to model explicitly the process by which data are observed. Parameters may not be identifiable from the data. And the likelihood function becomes significantly more complex: there is coupling between the likelihood's dependence on different parameters; worse, the function is no longer concave and generally has multiple local maxima.

The same issues regarding observation processes (ones that are not missing at random) and identifiability arise equally in the context of Markov network learning. The issue regarding the complexity of the likelihood function is analogous, although not quite the same. In the case of Markov networks, of course, we have coupling between the parameters even in the likelihood function for complete data. However, as we discuss, in the complete data case, the log-likelihood function is concave and easily optimized using gradient methods. Once we have missing data, we lose the concavity of the function and can have multiple local maxima. Indeed, the example we used was in the context of a Bayesian network of the form $X \rightarrow Y$, which can also be represented as a Markov network. Of course, the parameterization of the two models is not the same, and so the form of the function may differ. However, one can verify that a function that is multimodal in one parameterization will also be multimodal in the other.

20.3.3.1 Gradient Ascent

As in the case of Bayesian networks, if we assume our data is missing at random, we can perform maximum-likelihood parameter estimation by using some form of gradient ascent process to optimize the likelihood function. Let us therefore begin by analyzing the form of the gradient in the case of missing data. Let $\mathcal{D}$ be a data set where some entries are missing; let $\boldsymbol{o}[m]$ be the observed entries in the mth data instance and $\mathcal{H}[m]$ be the random variables that are the missing entries in that instance, so that for any $\boldsymbol{h}[m] \in Val(\mathcal{H}[m])$, $(\boldsymbol{o}[m], \boldsymbol{h}[m])$ is a complete assignment to $\mathcal{X}$.

As usual, the average log-likelihood function has the form:

$$\begin{aligned} \frac{1}{M} \ln P(\mathcal{D} \mid \boldsymbol{\theta}) &= \frac{1}{M} \sum_{m=1}^{M} \ln \left(\sum_{\boldsymbol{h}[m]} P(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) \right) \\ &= \frac{1}{M} \sum_{m=1}^{M} \ln \left(\sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) \right) - \ln Z. \end{aligned} \tag{20.8}$$

Now, consider a single term within the summation, $\sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta})$. This expression has the same form as a partition function; indeed, it is precisely the partition function for the Markov network that we would obtain by reducing our original Markov network with the observation $\boldsymbol{o}[m]$, to obtain a Markov network representing the conditional distribution $\tilde{P}(\mathcal{H}[m] \mid \boldsymbol{o}[m])$. Therefore, we can apply proposition 20.2 and conclude that:

$$\frac{\partial}{\partial \theta_i} \ln \sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) = \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})}[f_i],$$

that is, the gradient of this term is simply the *conditional* expectation of the feature, given the observations in this instance.

Putting this together with previous computations, we obtain the following:

Proposition 20.3 *For a data set $\mathcal{D}$*

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \left[\sum_{m=1}^{M} \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] | \boldsymbol{o}[m], \boldsymbol{\theta})} [f_i] \right] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]. \tag{20.9}$$

In other words, the gradient for feature f_i in the case of missing data is the difference between two expectations — the feature expectation *over the data and the hidden variables* minus the feature expectation over all of the variables.

It is instructive to compare the cost of this computation to that of computing the gradient in equation (20.4). For the latter, to compute the second term in the derivative, we need to run inference once, to compute the expected feature counts relative to our current distribution $P(\mathcal{X} \mid \boldsymbol{\theta})$. The first term is computed by simply aggregating the feature over the data. By comparison, to compute the derivative here, we actually need to run inference separately for every instance m, conditioning on $\boldsymbol{o}[m]$. Although inference in the reduced network may be simpler (since reduced factors are simpler), the cost of this computation is still much higher than learning without missing data. Indeed, not surprisingly, the cost here is comparable to the cost of a single iteration of gradient descent or EM in Bayesian network learning.

20.3.3.2 Expectation Maximization

As for any other probabilistic model, an alternative method for parameter estimation in context of missing data is via the expectation maximization algorithm. In the case of Bayesian network learning, EM seemed to have significant advantages. Can we define a variant of EM for Markov networks? And does it have the same benefits?

The answer to the first question is clearly yes. We can perform an E-step by using our current parameters $\boldsymbol{\theta}^{(t)}$ to compute the expected sufficient statistics, in this case, the expected feature counts. That is, at iteration t of the EM algorithm, we compute, for each feature f_i, the expected sufficient statistic:

$$\bar{M}_{\boldsymbol{\theta}^{(t)}}[f_i] = \frac{1}{M} \left[\sum_{m=1}^{M} \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] | \boldsymbol{o}[m], \boldsymbol{\theta})} [f_i] \right].$$

With these expected feature counts, we can perform an M-step by doing maximum likelihood parameter estimation. The proofs of convergence and other properties of the algorithm go through unchanged.

Here, however, there is one critical difference. Recall that, in the case of directed models, given the expected sufficient statistics, we can perform the M-step efficiently, in closed form. By contrast, the M-step for Markov networks requires that we run inference multiple times, once for each iteration of whatever gradient ascent procedure we are using. At step k of this "inner-loop" optimization, we now have a gradient of the form:

$$\bar{M}_{\boldsymbol{\theta}^{(t)}}[f_i] - \boldsymbol{E}_{\boldsymbol{\theta}^{(t,k)}}[f_i].$$

The trade-offs between the two algorithms are now more subtle than in the case of Bayesian networks. For the joint gradient ascent procedure of the previous section, we need to run inference $M + 1$ times in each gradient step: once without evidence, and once for each data case. If we use EM, we run inference M times to compute the expected sufficient statistics in the E-step, and then once for each gradient step, to compute the second term in the gradient. Clearly, there is a computational savings here. However, each of these gradient steps now uses an "out-of-date" set of expected sufficient statistics, making it increasingly less relevant as our optimization proceeds.

In fact, we can view the EM algorithm, in this case, as a form of caching of the first term in the derivative: Rather than compute the expected counts in each iteration, we compute them every few iterations, take a number of gradient steps, and then recompute the expected counts. There is no need to run the "inner-loop" optimization until convergence; indeed, that strategy is often not optimal in practice.

20.3.4 Maximum Entropy and Maximum Likelihood ⋆

We now return to the case of basic maximum likelihood estimation, in order to derive an alternative formulation that provides significant insight. In particular, we now use theorem 20.1 to relate maximum likelihood estimation in log-linear models to another important class or problems examined in statistics: the problem of finding the distribution of maximum entropy subject to a set of constraints.

To motivate this alternative formulation, consider a situation where we are given some summary statistics of an empirical distribution, such as those that may be published in a census report. These statistics may include the marginal distributions of single variables, of certain pairs, and perhaps of other events that the researcher summarizing the data happened to consider of interest. As another example, we might know the average final grade of students in the class and the correlation of their final grade with their homework scores. However, we do not have access to the full data set. While these two numbers constrain the space of possible distributions over the domain, they do not specify it uniquely. Nevertheless, we might want to construct a "typical" distribution that satisfies the constraints and use it to answer other queries.

One compelling intuition is that we should select a distribution that satisfies the given constraints but has no additional "structure" or "information." There are many ways of making this intuition precise. One that has received quite a bit of attention is based on the intuition that entropy is the inverse of information, so that we should search for the distribution of highest entropy. (There are more formal justifications for this intuition, but these are beyond the scope of this book.) More formally, in *maximum entropy* estimation, we solve the following problem:

maximum entropy

Maximum-Entropy:

Find	$Q(\mathcal{X})$
maximizing	$\boldsymbol{H}_Q(\mathcal{X})$
subject to	

$$\boldsymbol{E}_Q[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k. \tag{20.10}$$

expectation constraints

The constraints of equation (20.10) are called *expectation constraints*, since they constrain us to the set of distributions that have a particular set of expectations. We know that this set is

non-empty, since we have one example of a distribution that satisfies these constraints — the empirical distribution.

Somewhat surprisingly, the solution to this problem is a Gibbs distribution over the features $\mathcal{F}$ that matches the given expectations.

Theorem 20.2 *The distribution Q^* is the maximum entropy distribution satisfying equation (20.10) if and only if $Q^* = P_{\hat{\boldsymbol{\theta}}}$, where*

$$P_{\hat{\boldsymbol{\theta}}}(\mathcal{X}) = \frac{1}{Z(\hat{\boldsymbol{\theta}})} \exp\left\{\sum_i \hat{\theta}_i f_i(\mathcal{X})\right\}$$

and $\hat{\boldsymbol{\theta}}$ is the maximum likelihood parameterization relative to $\mathcal{D}$.

Proof For notational simplicity, let $P = P_{\hat{\boldsymbol{\theta}}}$. From theorem 20.1, it follows that $\boldsymbol{E}_P[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})]$ for $i = 1, \ldots, k$, and hence that P satisfies the constraints of equation (20.10). Therefore, to prove that $P = Q^*$, we need only show that $\boldsymbol{H}_P(\mathcal{X}) \geq \boldsymbol{H}_Q(\mathcal{X})$ for all other distributions Q that satisfy these constraints. Consider any such distribution Q.

From proposition 8.1, it follows that:

$$\boldsymbol{H}_P(\mathcal{X}) = -\sum_i \hat{\theta}_i \boldsymbol{E}_P[f_i] + \ln Z(\boldsymbol{\theta}). \tag{20.11}$$

Thus,

$$\begin{aligned}
\boldsymbol{H}_P(\mathcal{X}) - \boldsymbol{H}_Q(\mathcal{X}) &= -\left[\sum_i \hat{\theta}_i \boldsymbol{E}_P[f_i(\mathcal{X})]\right] + \ln Z_P - \boldsymbol{E}_Q[-\ln Q(\mathcal{X})] \\
(i) &= -\left[\sum_i \hat{\theta}_i \boldsymbol{E}_Q[f_i(\mathcal{X})]\right] + \ln Z_P + \boldsymbol{E}_Q[\ln Q(\mathcal{X})] \\
&= \boldsymbol{E}_Q[-\ln P(\mathcal{X})] + \boldsymbol{E}_Q[\ln Q(\mathcal{X})] \\
&= \boldsymbol{D}(Q \| P) \geq 0,
\end{aligned}$$

where (i) follows from the fact that both $P_{\hat{\boldsymbol{\theta}}}$ and Q satisfy the constraints, so that $\boldsymbol{E}_{P_{\hat{\boldsymbol{\theta}}}}[f_i] = \boldsymbol{E}_Q[f_i]$ for all i.

We conclude that $\boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) \geq \boldsymbol{H}_Q(\mathcal{X})$ with equality if and only if $P_{\hat{\boldsymbol{\theta}}} = Q$. Thus, the maximum entropy distribution Q^* is necessarily equal to $P_{\hat{\boldsymbol{\theta}}}$, proving the result. ■

duality

One can also provide an alternative proof of this result based on the concept of *duality* discussed in appendix A.5.4. Using this alternative derivation, one can show that the two problems, maximizing the entropy given expectation constraints and maximizing the likelihood given structural constraints on the distribution, are *convex duals* of each other. (See exercise 20.5.)

Both derivations show that these objective functions provide bounds on each other, and are identical at their convergence point. That is, for the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$,

$$\boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) = -\frac{1}{M}\ell(\hat{\boldsymbol{\theta}} : \mathcal{D}).$$

As a consequence, we see that for any set of parameters $\boldsymbol{\theta}$ and for any distribution Q that satisfy the expectation constraints equation (20.10), we have that

$$\boldsymbol{H}_Q(\mathcal{X}) \leq \boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) = -\frac{1}{M}\ell(\hat{\boldsymbol{\theta}} : \mathcal{D}) \leq -\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D})$$

with equality if and only if $Q = \mathcal{P}_{\boldsymbol{\theta}}$. We note that, while we provided a proof for this result from first principles, it also follows directly from the theory of convex duality.

Our discussion has shown an entropy dual only for likelihood. A similar connection can be shown between conditional likelihood and conditional entropy; see exercise 20.6.

20.4 Parameter Priors and Regularization

So far, we have focused on maximum likelihood estimation for selecting parameters in a Markov network. However, as we discussed in chapter 17, maximum likelihood estimation (MLE) is prone to overfitting to the training data. Although the effects are not as transparent in this case (due to the lack of direct correspondence between empirical counts and parameters), overfitting of the maximum likelihood estimator is as much of a problem here.

As for Bayesian networks, we can reduce the effect of overfitting by introducing a prior distribution $P(\boldsymbol{\theta})$ over the model parameters. Note that, because we do not have a decomposable closed form for the likelihood function, we do not obtain a decomposable closed form for the posterior in this case. Thus, a fully Bayesian approach, where we integrate out the parameters to compute the next prediction, is not generally feasible in Markov networks. However, we can aim to perform *MAP estimation* — to find the parameters that maximize $P(\boldsymbol{\theta})P(\mathcal{D} \mid \boldsymbol{\theta})$.

MAP estimation

Given that we have no constraints on the conjugacy of the prior and the likelihood, we can consider virtually any reasonable distribution as a possible prior. However, only a few priors have been applied in practice.

20.4.1 Local Priors

Most commonly used is a Gaussian prior on the log-linear parameters $\boldsymbol{\theta}$. The most standard form of this prior is simply a zero-mean diagonal Gaussian, usually with equal variances for each of the weights:

$$P(\boldsymbol{\theta} \mid \sigma^2) = \prod_{i=1}^{k} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{\theta_i^2}{2\sigma^2}\right\},$$

hyperparameter

for some choice of the variance σ^2. This variance is a *hyperparameter*, as were the α_i's in the Dirichlet distribution (section 17.3.2). Converting to log-space (in which the optimization is typically done), this prior gives rise to a term of the form:

$$-\frac{1}{2\sigma^2}\sum_{i=1}^{k}\theta_i^2,$$

This term places a quadratic penalty on the magnitude of the weights, where the penalty is measured in Euclidean, or L_2-norm, generally called an *L_2-regularization* term. This term is

L_2-regularization

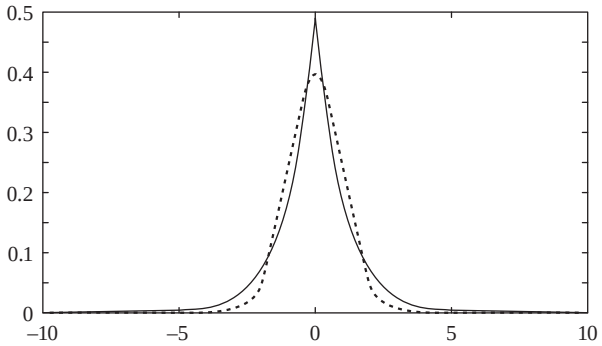


Figure 20.3 Laplacian distribution ($\beta = 1$) and Gaussian distribution ($\sigma^2 = 1$)

concave, and therefore it gives rise to a concave objective, which can be optimized using the same set of methods as standard MLE.

Laplacian distribution

A different prior that has been used in practice uses the zero-mean *Laplacian distribution*, which, for a single parameter, has the form

$$P_{Laplacian}(\theta \mid \beta) = \frac{1}{2\beta} \exp\left\{-\frac{|\theta|}{\beta}\right\}. \tag{20.12}$$

One example of the Laplacian distribution is shown in figure 20.3; it has a nondifferentiable point at $\theta = 0$, arising from the use of the absolute value in the exponent. As for the Gaussian case, one generally assumes that the different parameters θ_i are independent, and often (but not always) that they are identically distributed with the same hyperparameter β. Taking the logarithm, we obtain a term

$$-\frac{1}{\beta}\sum_{i=1}^{k}|\theta_i|$$

L_1-regularization

that also penalizes weights of high magnitude, measured using the L_1-norm. Thus, this approach is generally called *L_1-regularization*.

Both forms of regularization penalize parameters whose magnitude (positive or negative) is large. Why is a bias in favor of parameters of low magnitude a reasonable one? Recall from our discussion in section 17.3 that a prior often serves to pull the distribution toward an "uninformed" one, smoothing out fluctuations in the data. Intuitively, a distribution is "smooth" if the probabilities assigned to different assignments are not radically different. Consider two assignments ξ and ξ'; their relative probability is

$$\frac{P(\xi)}{P(\xi')} = \frac{\tilde{P}(\xi)/Z_{\boldsymbol{\theta}}}{\tilde{P}(\xi')/Z_{\boldsymbol{\theta}}} = \frac{\tilde{P}(\xi)}{\tilde{P}(\xi')}.$$

Moving to log-space and expanding the unnormalized measure $\tilde{P}$, we obtain:

$$\begin{aligned}\ln \frac{P(\xi)}{P(\xi')} &= \sum_{i=1}^{k} \theta_i f_i(\xi) - \sum_{i=1}^{k} \theta_i f_i(\xi') \\ &= \sum_{i=1}^{k} \theta_i (f_i(\xi) - f_i(\xi')).\end{aligned}$$

When all of the θ_i's have small magnitude, this log-ratio is also bounded, resulting in a smooth distribution. Conversely, when the parameters can be large, we can obtain a "spiky" distribution with arbitrarily large differences between the probabilities of different assignments.

In both the L_2 and the L_1 case, we penalize the magnitude of the parameters. In the Gaussian case, the penalty grows quadratically with the parameter magnitude, implying that an increase in magnitude in a large parameter is penalized more than a similar increase in a small parameter. For example, an increase in θ_i from 0 to 0.1 is penalized less than an increase from 3 to 3.1. In the Laplacian case, the penalty is linear in the parameter magnitude, so that the penalty growth is invariant over the entire range of parameter values. This property has important ramifications. In the quadratic case, as the parameters get close to 0, the effect of the penalty diminishes. Hence, the models that optimize the penalized likelihood tend to have many small weights. Although the resulting models are smooth, as desired, they are structurally quite dense. By comparison, in the L_1 case, the penalty is linear all the way until the parameter value is 0. This penalty provides a continued incentive for parameters to shrink until they actually hit 0. As a consequence, **the models learned with an L_1 penalty tend to be much sparser than those learned with an L_2 penalty, with many parameter weights achieving a value of 0. From a structural perspective, this effect gives rise to models with fewer edges and sparser potentials, which are potentially much more tractable.** We return to this issue in section 20.7.

Importantly, both the L_1 and L_2 regularization terms are concave. Because the log-likelihood is also concave, the resulting posterior is concave, and can therefore be optimized efficiently using the gradient-based methods we described for the likelihood case. Moreover, the introduction of these penalty terms serves to reduce or even eliminate multiple (equivalent) optima that arise when the parameterization of the network is redundant. For example, consider the trivial example where we have no data. In this case, the maximum likelihood solution is (as desired) the uniform distribution. However, due to redundancy, there is a continuum of parameterizations that give rise to the uniform distribution. However, when we introduce either of the earlier prior distributions, the penalty term drives the parameters toward zero, giving rise to the unique optimum $\boldsymbol{\theta} = \mathbf{0}$. Although one can still construct examples where multiple optima occur, they are very rare in practice. Conversely, methods that eliminate redundancies by reexpressing some of the parameters in terms of others can produce undesirable interactions with the regularization terms, giving rise to priors where some parameters are penalized more than others.

The regularization hyperparameters — σ^2 in the L_2 case, and β in the L_1 case — encode the strength in our belief that the model weights should be close to 0. The larger these parameters (both in the denominator), the broader our parameter prior, and the less strong our bias toward 0. In principle, any choice of hyperparameter is legitimate, since a prior is simply a reflection of our beliefs. **In practice, however, the choice of prior can have a significant effect on the quality of our learned model. A standard method for selecting this parameter is via a**

cross-validation procedure, as described in box 16.A: We repeatedly partition the training set, learn a model over one part with some choice of hyperparameter, and measure the performance of the learned model (for example, log-likelihood) on the held-out fragment.

20.4.2 Global Priors

conjugate prior

An alternative approach for defining priors is to search for a *conjugate prior*. Examining the likelihood function, we see that the posterior over parameters has the following general form:

$$\begin{aligned} P(\boldsymbol{\theta} \mid \mathcal{D}) &\propto P(\boldsymbol{\theta}) P(\mathcal{D} \mid \boldsymbol{\theta}) \\ &= P(\boldsymbol{\theta}) \exp\left\{ \sum_i M \boldsymbol{E}_{\mathcal{D}}[f_i] \theta_i - M \ln Z(\boldsymbol{\theta}) \right\}. \end{aligned}$$

This expression suggests that we use a family of prior distributions of the form:

$$P(\boldsymbol{\theta}) \propto \exp\left\{ \sum_i M_0 \alpha_i \theta_i - M_0 \ln Z(\boldsymbol{\theta}) \right\}.$$

This form defines a family of priors with hyperparameters $\{\alpha_i\}$. It is easy to see that the posterior is from the same family with $\alpha'_i = \alpha_i + \boldsymbol{E}_{\mathcal{D}}[f_i]$ and $M'_0 = M_0 + M$, so that this prior is conjugate to the log-linear model likelihood function.

We can think of the hyperparameters $\{\alpha_i\}$ as specifying the sufficient statistics from prior observations and of M_0 as specifying the number of these prior observations. This formulation is quite similar to the use of pseudocounts in the BDe priors for directed models (see section 17.4.3). The main difference from directed models is that this conjugate family (both the prior and the likelihood) does not decompose into independent priors for the different features.

20.5 Learning with Approximate Inference

The methods we have discussed here assume that we are able to compute the partition function $Z(\boldsymbol{\theta})$ and expectations such as $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_i]$. In many real-life applications the structure of the network does not allow for exact computation of these terms. For example, in applications to image segmentation (box 4.B), we generally use a grid-structured network, which requires exponential size clusters for exact inference.

The simplest approach for learning in intractable networks is to apply the learning procedure (say, conjugate gradient ascent) using an approximate inference procedure to compute the required queries about the distribution $P_{\boldsymbol{\theta}}$. This view decouples the question of inference from learning and treats the inference procedure as a black box during learning. The success of such an approach depends on whether the approximation method interferes with the learning. In particular, **nonconvergence of the inference method, or convergence to approximate answers, can lead to inaccurate and even oscillating estimates of the gradient, potentially harming convergence of the overall learning algorithm.** This type of situation can arise both in particle-based methods (say MCMC sampling) and in global algorithms such as belief propagation. In this section, we describe several methods that better integrate the inference into the learning outer loop in order to reduce problems such as this.

A second approach for dealing with inference-induced costs is to come up with alternative (possibly approximate) objective functions whose optimization does not require (as much) inference. Some of these techniques are reviewed in the next section. However, one of the main messages of this section is that the boundary between these two classes of methods is surprisingly ambiguous. **Approximately optimizing the likelihood objective by using an approximate inference algorithm to compute the gradient can often be reformulated as exactly optimizing an approximate objective.** When applicable, this view is often more insightful and also more usable. First, it provides more insight about the outcome of the optimization. Second, it may allow us to bound the error in the optimum in terms of the distance between the two functions being optimized. Finally, by formulating a clear objective to be optimized, we can apply any applicable optimization algorithm, such as conjugate gradient or Newton's method.

Importantly, while we describe the methods in this section relative to the plain likelihood objective, they apply almost without change to the generalizations and extensions we describe in this chapter: conditional Markov networks; parameter priors and regularization; structure learning; and learning with missing data.

20.5.1 Belief Propagation

belief propagation

A fairly popular approach for approximate inference is the *belief propagation* algorithm and its variants. Indeed, in many cases, an algorithm in this family would be used for inference in the model resulting from the learning procedure. In this case, it can be shown that **we should learn the model using the same inference algorithm that will be used for querying it. Indeed, it can be shown that using a model trained with the same approximate inference algorithm is better than using a model trained with exact inference.**

At first glance, the use of belief propgation for learning appears straightforward. We can simply run BP within every iteration of gradient ascent to compute the expected feature counts used in the gradient computation. Due to the family preservation property, each feature f_i must be a subset of a cluster $\boldsymbol{C}_i$ in the cluster graph. Hence, to compute the expected feature count $\boldsymbol{E}_{\boldsymbol{\theta}}[f_i]$, we can compute the BP marginals over $\boldsymbol{C}_i$, and then compute the expectation. In practice, however, this approach can be highly problematic. As we have seen, BP often does not converge. The marginals that we derive from the algorithm therefore oscillate, and the final results depend on the point at which we choose to stop the algorithm. As a result, the gradient computed from these expected counts is also *unstable*. This instability can be a significant problem in a gradient-based procedure, since it can gravely hurt the convergence properties of the algorithm. This problem is even more severe in the context of line-search methods, where the function evaluations can be inconsistent at different points in the line search.

unstable gradient

There are several solutions to this problem: One can use one of the convergent alternatives to the BP algorithm that still optimizes the same Bethe energy objective; one can use a convex energy approximation, such as those of section 11.3.7.2; or, as we now show, one can reformulate the task of learning with approximate inference as optimizing an alternative objective, allowing the use of a range of optimization methods with better convergence properties.

20.5.1.1 Pseudo-moment Matching

Let us begin by a simple analysis of the fixed points of the learning algorithm. At convergence, the approximate expectations must satisfy the condition of theorem 20.1; in particular, the converged BP beliefs for $\boldsymbol{C}_i$ must satisfy

$$\boldsymbol{E}_{\beta_i(\boldsymbol{C}_i)}[f_{\boldsymbol{C}_i}] = \boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{C}_i)].$$

Now, let us consider the special case where our feature model defines a set of fully parameterized potentials that precisely match the clusters used in the BP cluster graph. That is, for every cluster $\boldsymbol{C}_i$ in the cluster graph, and every assignment $\boldsymbol{c}_i^j$ to $\boldsymbol{C}_i$, we have a feature which is an indicator function $\boldsymbol{I}\{\boldsymbol{c}_i^j\}$, that is, it is 1 when $\boldsymbol{C}_i = \boldsymbol{c}_i^j$ and 0 otherwise. In this case, the preceding set of equalities imply that, for every assignment $\boldsymbol{c}_i^j$ to $\boldsymbol{C}_i$, we have that

$$\beta_i(\boldsymbol{c}_i^j) = \hat{P}(\boldsymbol{c}_i^j). \tag{20.13}$$

That is, at convergence of the gradient ascent algorithm, the convergence point of the underlying belief propagation must be to a set of beliefs that exactly matches the empirical marginals in the data. But if we already know the outcome of our convergence, there is no point to running the algorithm!

This derivation gives us a closed form for the BP potentials at the point when both algorithms — BP inference and parameter gradient ascent — have converged. As we have already discussed, the full-table parameterization of Markov network potentials is redundant, and therefore there are multiple solutions that can give rise to this set of beliefs. One of these solutions can be obtained by dividing each sepset in the calibrated cluster graph into one of the adjacent clique potentials. More precisely, for each sepset $\boldsymbol{S}_{i,j}$ between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, we select the endpoint for which $i < j$ (in some arbitrary ordering), and we then define:

$$\phi_i \leftarrow \frac{\beta_i}{\mu_{i,j}}.$$

We perform this transformation for each sepset. We use the final set of potentials as the parameterization for our Markov network. We can show that a single pass of message passing in a particular order gives rise to a calibrated cluster graph whose potentials are precisely the ones in equation (20.13). Thus, in this particular special case, we can provide a closed-form solution to both the inference and learning problem. This approach is called *pseudo-moment matching*.

pseudo-moment matching

While it is satisfying that we can find a solution so effectively, the form of the solution should be considered with care. In particular, we note that the clique potentials are simply empirical cluster marginals divided by empirical sepset marginals. These quantities depend only on the local structure of the factor and not on any global aspect of the cluster graph, including its structure. For example, the BC factor is estimated in exactly the same way within the diamond network of figure 11.1a and within the chain network A—B—C—D. Of course, potentials are also estimated locally in a Bayesian network, but there the local calibration ensures that the distribution can be factorized using purely local computations. As we have already seen, this is not the case for Markov networks, and so we expect different potentials to adjust to fit each other; however, the estimation using loopy BP does not accommodate that. In a sense, this observation is not surprising, since the BP approach also ignores the more global information.

We note, however, that this purely local estimation of the parameters only holds under the very restrictive conditions described earlier. It does not hold when we have parameter priors (regularization), general features rather than table factors, any type of shared parameters (as in section 6.5), or conditional random fields. We discuss this more general case in the next section.

20.5.1.2 Belief Propagation and Entropy Approximations ⋆

We now provide a more general derivation that allows us to reformulate maximum-likelihood learning with belief propagation as a unified optimization problem with an approximate objective. This perspective opens the door to the use of better approximation algorithms.

maximum entropy

Our analysis starts from the *maximum-entropy* dual of the maximum-likelihood problem.

Maximum-Entropy:

Find $Q(\mathcal{X})$
maximizing $\boldsymbol{H}_Q(\mathcal{X})$
subject to

$$\boldsymbol{E}_Q[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k.$$

We can obtain a tractable approximation to this problem by applying the same sequence of transformations that we used in section 11.3.6 to derive belief propagation from the energy optimization problem. More precisely, assume we have a cluster graph $\mathcal{U}$ consisting of a set of clusters $\{\boldsymbol{C}_i\}$ connected by sepsets $\boldsymbol{S}_{i,j}$. Now, rather than optimize Maximum-Entropy over the space of distributions Q, we optimize over the set of possible pseudo-marginals in the *local consistency polytope Local*$[\mathcal{U}]$, as defined in equation (11.16). Continuing as in the BP derivation, we also approximate the entropy as in its *factored* form (definition 11.1):

local consistency polytope

factored entropy

$$\boldsymbol{H}_Q(\mathcal{X}) \approx \sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{U}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}). \tag{20.14}$$

As before, this reformulation is exact when the cluster graph is a tree but is approximate otherwise.

Putting these approximations together, we obtain the following approximation to the maximum-entropy optimization problem:

Approx-Maximum-Entropy:

Find $\boldsymbol{Q}$
maximizing $\sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{U}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j})$
subject to

$$\begin{aligned} \boldsymbol{E}_{\beta_i}[f_i] &= \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k \\ \boldsymbol{Q} &\in \mathit{Local}[\mathcal{U}]. \end{aligned} \tag{20.15}$$

CAMEL

This approach is called *CAMEL*, for constrained approximate maximum enropy learning.

Example 20.2

To illustrate this reformulation, consider a simple pairwise Markov network over the binary variables A, B, C, with three clusters: $\boldsymbol{C}_1 = \{A, B\}, \boldsymbol{C}_2 = \{B, C\}, \boldsymbol{C}_3 = \{A, C\}$. We assume that the log-linear model is defined by the following two features, both of which are shared over all clusters: $f_{00}(x, y) = 1$ if $x = 0$ and $y = 0$, and 0 otherwise; and $f_{11}(x, y) = 1$ if $x = 1$ and $y = 1$. Assume we have 3 data instances $[0, 0, 0]$, $[0, 1, 0]$, $[1, 0, 0]$. The unnormalized empirical counts of each feature, pooled over all clusters, is then $\boldsymbol{E}_{\hat{P}}[f_{00}] = (3 + 1 + 1)/3 = 5/3$, $\boldsymbol{E}_{\hat{P}}[f_{11}] = 0$. In this case, the optimization of equation (20.15) would take the following form:

$$
\begin{aligned}
\textbf{Find} \quad & \boldsymbol{Q} = \{\beta_1, \beta_2, \beta_3, \mu_{1,2}, \mu_{2,3}, \mu_{1,3}\} \\
\textbf{maximizing} \quad & \boldsymbol{H}_{\beta_1}(A, B) + \boldsymbol{H}_{\beta_2}(B, C) + \boldsymbol{H}_{\beta_3}(A, C) \\
& -\boldsymbol{H}_{\mu_{1,2}}(B) - \boldsymbol{H}_{\mu_{2,3}}(C) - \boldsymbol{H}_{\mu_{1,3}}(A) \\
\textbf{subject to} \quad & \\
& \sum_i \boldsymbol{E}_{\beta_i}[f_{00}] = 5/3 \\
& \sum_i \boldsymbol{E}_{\beta_i}[f_{11}] = 0 \\
& \sum_a \beta_1(a, b) - \sum_c \beta_2(b, c) = 0 \quad \forall b \\
& \sum_b \beta_2(b, c) - \sum_a \beta_3(a, c) = 0 \quad \forall c \\
& \sum_c \beta_3(a, c) - \sum_b \beta_1(a, b) = 0 \quad \forall a \\
& \sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) = 1 \quad i = 1, 2, 3 \\
& \beta_i \geq 0 \quad i = 1, 2, 3.
\end{aligned}
$$

The CAMEL optimization problem of equation (20.15) is a constrained maximization problem with linear constraints and a nonconcave objective. The problem actually has two distinct sets of constraints: the first set encodes the moment-matching constraints and comes from the learning problem; and the second set encodes the constraint that $\boldsymbol{Q}$ be in the marginal polytope and arises from the cluster-graph approximation. It thus forms a unified optimization problem that encompasses both the learning task — moment matching — and the inference task — obtaining a set of consistent pseudo-marginals over a cluster graph. Analogously, if we introduce Lagrange multipliers for these constraints (as in appendix A.5.3), they would have very different interpretations. The multipliers for the first set of constraints would correspond to weights $\boldsymbol{\theta}$ in the log-linear model, as in the max-likelihood / max-entropy duality ; those in the second set would correspond to messages $\delta_{i \to j}$ in the cluster graph, as in the BP algorithm.

This observation leads to several solution algorithms for this problem. In one class of methods, we could introduce Lagrange multipliers for all of the constraints and then optimize the resulting problem over these new variables. If we perform the optimization by a double-loop algorithm where the outer loop optimizes over $\boldsymbol{\theta}$ (say using gradient ascent) and the inner loops "optimizes" the $\delta_{i \to j}$ by iterating their fixed point equations, the result would be precisely gradient ascent over parameters with BP in the inner loop for inference.

20.5.1.3 Sampling-Based Learning ⋆

The partition function $Z(\theta)$ is a summation over an exponentially large space. One approach to approximating this summation is to reformulate it as an expectation with respect to some distribution $Q(\mathcal{X})$:

$$\begin{aligned} Z(\boldsymbol{\theta}) &= \sum_{\xi} \exp\left\{\sum_i \theta_i f_i(\xi)\right\} \\ &= \sum_{\xi} \frac{Q(\xi)}{Q(\xi)} \exp\left\{\sum_i \theta_i f_i(\xi)\right\} \\ &= \boldsymbol{E}_Q\left[\frac{1}{Q(\mathcal{X})} \exp\left\{\sum_i \theta_i f_i(\mathcal{X})\right\}\right]. \end{aligned}$$

importance sampling

This is precisely the form of the *importance sampling* estimator described in section 12.2.2. Thus, we can approximate it by generating samples from Q, and correcting appropriately via weights. We can simplify this expression if we choose Q to be $P_{\boldsymbol{\theta}^0}$ for some set of parameters $\boldsymbol{\theta}^0$:

$$\begin{aligned} Z(\boldsymbol{\theta}) &= \boldsymbol{E}_{P_{\boldsymbol{\theta}^0}}\left[\frac{Z(\boldsymbol{\theta}^0)\exp\{\sum_i \theta_i f_i(\mathcal{X})\}}{\exp\{\sum_i \theta_i^0 f_i(\mathcal{X})\}}\right] \\ &= Z(\boldsymbol{\theta}^0)\boldsymbol{E}_{P_{\boldsymbol{\theta}^0}}\left[\exp\left\{\sum_i (\theta_i - \theta_i^0) f_i(\mathcal{X})\right\}\right]. \end{aligned}$$

If we can sample instances $\xi^1, \ldots, \xi^K$ from $P_{\boldsymbol{\theta}_0}$, we can approximate the log-partition function as:

$$\ln Z(\boldsymbol{\theta}) \approx \ln\left(\frac{1}{K}\sum_{k=1}^{K} \exp\left\{\sum_i (\theta_i - \theta_i^0) f_i(\xi^k)\right\}\right) + \ln Z(\boldsymbol{\theta}^0). \tag{20.16}$$

We can plug this approximation of $\ln Z(\boldsymbol{\theta})$ into the log-likelihood of equation (20.3) and optimize it. Note that $\ln Z(\boldsymbol{\theta}^0)$ is a constant that we can ignore in the optimization, and the resulting expression is therefore a simple function of $\boldsymbol{\theta}$, which can be optimized using methods such as gradient ascent or one of its extensions. Interestingly, gradient ascent over $\boldsymbol{\theta}$ relative to equation (20.16) is equivalent to utilizing an importance sampling estimator directly to approximate the expected counts in the gradient of equation (20.4) (see exercise 20.12). However, as we discussed, it is generally more instructive and useful to view such methods as exactly optimizing an approximate objective rather than approximately optimizing the exact likelihood.

Of course, as we discussed in section 12.2.2, the quality of an importance sampling estimator depends on the difference between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^0$: the greater the difference, the larger the variance of the importance weights. Thus, this type of approximation is reasonable only in a neighborhood surrounding $\boldsymbol{\theta}^0$.

MCMC

How do we use this approximation? One possible strategy is to iterate between two steps. In one we run a sampling procedure, such as *MCMC*, to generate samples from the current parameter set $\boldsymbol{\theta}^t$. Then in the second iteration we use some gradient procedure to find $\boldsymbol{\theta}^{t+1}$

that improve the approximate log-likelihood based on these samples. We can then regenerate samples and repeat the process. As the samples are regenerated from a new distribution, we can hope that they are generated from a distribution not too far from the one we are currently optimizing, maintaining a reasonable approximation.

20.5.2 MAP-Based Learning ⋆

MAP assignment

As another approximation to the inference step in the learning algorithm, we can consider approximating the expected feature counts with their counts in the single *MAP assignment* to the current Markov network. As we discussed in chapter 13, in many classes of models, computing a single MAP assignment is a much easier computational task, making this a very appealing approach in many settings.

More precisely, to approximate the gradient at a given parameter assignment $\boldsymbol{\theta}$, we compute

$$\boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] - f_i(\xi^{\text{MAP}}(\boldsymbol{\theta})), \tag{20.17}$$

Viterbi training

where $\xi^{\text{MAP}}(\boldsymbol{\theta}) = \arg\max_{\xi} P(\xi \mid \boldsymbol{\theta})$ is the MAP assignment given the current set of parameters $\boldsymbol{\theta}$. This approach is also called *Viterbi training*.

Once again, we can gain considerable intuition by reformulating this approximate inference step as an exact optimization of an approximate objective. Some straightforward algebra shows that this gradient corresponds exactly to the approximate objective

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}), \tag{20.18}$$

or, due to the cancellation of the partition function:

$$\frac{1}{M}\sum_{m=1}^{M} \ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}). \tag{20.19}$$

To see this, consider a single data instance $\xi[m]$:

$$\begin{aligned}
\ln P(\xi[m] \mid \boldsymbol{\theta}) - \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) \\
&= [\ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta})] - [\ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta})] \\
&= \ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) \\
&= \sum_i \theta_i [f_i(\xi[m]) - f_i(\xi^{\text{MAP}}(\boldsymbol{\theta}))].
\end{aligned}$$

If we average this expression over all data instances and take the partial derivative relative to θ_i, we obtain an expression whose gradient is precisely equation (20.17).

The first term in equation (20.19) is an average of expressions of the form $\ln \tilde{P}(\xi \mid \boldsymbol{\theta})$. Each such expression is a linear function in $\boldsymbol{\theta}$, and hence their average is also linear in $\boldsymbol{\theta}$. The second term, $\tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta})$, may appear to be the log-probability of an instance. However, as indicated by the notation, $\xi^{\text{MAP}}(\boldsymbol{\theta})$ is itself a function of $\boldsymbol{\theta}$: in different regions of the parameter space, the MAP assignment changes. In fact, this term is equal to:

$$\ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) = \max_{\xi} \ln P(\xi \mid \boldsymbol{\theta}).$$

This is a maximum of linear functions, which is a convex, piecewise-linear function. Therefore, its negation is concave, and so the entire objective of equation (20.19) is also concave and hence has a global optimum.

Although reasonable at first glance, a closer examination reveals some important issues with this objective. Consider again a single data instance $\xi[m]$. Because $\xi^{\text{MAP}}(\boldsymbol{\theta})$ is the MAP assignment, it follows that $\ln P(\xi[m] \mid \boldsymbol{\theta}) \leq \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta})$, and therefore the objective is always nonpositive. The maximal value of 0 can be achieved in two ways. The first is if we manage to find a setting of $\boldsymbol{\theta}$ in which the empirical feature counts match the feature counts in $\xi^{\text{MAP}}(\boldsymbol{\theta})$. This optimum may be hard to achieve: Because the counts in $\xi^{\text{MAP}}(\boldsymbol{\theta})$ are discrete, they take on only a finite set of values; for example, if we have a feature that is an indicator function for the event $X_i = x_i$, its count can take on only the values 0 or 1, depending on whether the MAP assignment has $X_i = x_i$ or not. Thus, we may never be able to match the feature counts exactly. The second way of achieving the optimal value of 0 is to set all of the parameters θ_i to 0. In this case, we obtain the uniform distribution over assignments, and the objective achieves its maximum value of 0. This possible behavior may not be obvious when we consider the gradient, but it becomes apparent when we consider the objective we are trying to optimize.

That said, we note that in the early stages of the optimization, when the expected counts are far from the MAP counts, the gradient still makes progress in the general direction of increasing the relative log-probability of the data instances. This approach can therefore work fairly well in practice, especially if not optimized to convergence.

---

protein structure

Box 20.B — Case Study: CRFs for Protein Structure Prediction. *One interesting application of CRFs is to the task of predicting the three-dimensional* structure *of proteins. Proteins are constructed as chains of residues, each containing one of twenty possible amino acids. The amino acids are linked together into a common backbone structure onto which amino-specific side-chains are attached. An important computational problem is that of predicting the side-chain conformations given the backbone. The full configuration for a side-chain consists of up to four angles, each of which takes on a continuous value. However, in practice, angles tend to cluster into bins of very similar angles, so that the common practice is to discretize the value space of each angle into a small number (usually up to three) bins, called* rotamers.

With this transformation, side-chain prediction can be formulated as a discrete optimization problem, where the objective is an energy over this discrete set of possible side-chain conformations. Several energy functions have been proposed, all of which include various repulsive and attractive terms between the side-chain angles of nearby residues, as well as terms that represent a prior and internal constraints within the side chain for an individual residue. Rosetta, a state-of-the-art system, uses a combination of eight energy terms, and uses simulated annealing to search for the minimal energy configuration. However, even this highly engineered system still achieves only moderate accuracies (around 72 percent of the discretized angles predicted correctly). An obvious question is whether the errors are due to suboptimal answers returned by the optimization algorithm, or to the design of the energy function, which may not correctly capture the true energy "preferences" of protein structures.

Yanover, Schueler-Furman, and Weiss (2007) propose to address this optimization problem using MAP inference techniques. The energy functions used in this type of model can also be viewed as the

log-potentials of a Markov network, where the variables represent the different angles to be inferred, and their values the discretized rotamers. The problem of finding the optimal configuration is then simply the MAP inference problem, and can be tackled using some of the algorithms described in chapter 13. Yanover et al. show that the TRW algorithm of box 13.A finds the provably global optimum of the Rosetta energy function for approximately 85 percent of the proteins in a standard benchmark set; this computation took only a few minutes per protein on a standard workstation. They also tackled the problem by directly solving the LP relaxation of the MAP problem using a commercial LP solver; this approach found the global optimum of the energy function for all proteins in the test set, but at a higher computational cost. However, finding the global minimum gave only negligible improvements on the actual accuracy of the predicted angles, suggesting that the primary source of inaccuracy in these models is in the energy function, not the optimization.

Thus, this problem seems like a natural candidate for the application of learning methods. The task was encoded as a CRF, whose input is a list of amino acids that make up the protein as well as the three-dimensional shape of the backbone. Yanover et al. encoded this distribution as a log-linear model whose features were the (eight) different components of the Rosetta energy function, and whose parameters were the weights of these features. Because exact inference for this model is intractable, it was trained by using a TRW variant for sum-product algorithms (see section 11.3.7.2). This variant uses a set of convex counting numbers to provide a convex approximation, and a lower bound, to the log-partition function. These properties guarantee that the learning process is stable and is continually improving a lower bound on the true objective. This new energy function improves performance from 72 percent to 78 percent, demonstrating that learning can significantly improve models, even those that are carefully engineered and optimized by a human expert. Notably, for the learned energy function, and for other (yet more sophisticated) energy functions, the use of globally optimal inference does lead to improvements in accuracy. Overall, a combination of these techniques gave rise to an accuracy of 82.6 percent, a significant improvement.

20.6 Alternative Objectives

Another class of approximations can be obtained directly by replacing the objective that we aim to optimize with one that is more tractable. To motivate the alternative objectives we present in this chapter, let us consider again the form of the log-likelihood objective, focusing, for simplicity, on the case of a single data instance ξ:

$$\begin{aligned}\ell(\boldsymbol{\theta} : \xi) &= \ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta}) \\ &= \ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \left(\sum_{\xi'} \tilde{P}(\xi' \mid \boldsymbol{\theta}) \right).\end{aligned}$$

Considering the first term, this objective aims to increase the log-measure (logarithm of the unnormalized probability) of the observed data instance ξ. Of course, because the log-measure is a linear function of the parameters in our log-linear representation, that goal can be achieved simply by increasing all of the parameters associated with positive empirical expectations in ξ, and decreasing all of the parameters associated with negative empirical expectations. Indeed,

we can increase the first term unboundedly using this approach. The second term, however, balances the first, since it is the logarithm of a sum of the unnormalized measures of instances, in this case, all possible instances in $Val(\mathcal{X})$. In a sense, then, we can view the log-likelihood objective as aiming to increasing the distance between the log-measure of ξ and the aggregate of the measures of all instances. We can thus view it as contrasting two terms. The key difficulty with this formulation, of course, is that the second term involves a summation over the exponentially many instances in $Val(\mathcal{X})$, and therefore requires inference in the network.

This formulation does, however, suggest one approach to approximating this objective: **perhaps we can still move our parameters in the right direction if we aim to increase the difference between the log-measure of the data instances and a *more tractable set* of other instances, one that does not require summation over an exponential space.** The *contrastive objectives* that we describe in this section all take that form.

contrastive objective

20.6.1 Pseudolikelihood and Its Generalizations

Perhaps the earliest method for circumventing the intractability of network inference is the pseudolikelihood objective. As one motivation for this approximation, consider the likelihood of a single instance ξ. Using the chain rule, we can write

$$P(\xi) = \prod_{j=1}^{n} P(x_j \mid x_1, \ldots, x_{j-1}).$$

We can approximate this formulation by replacing each term $P(x_i \mid x_1, \ldots, x_{i-1})$ by the conditional probability of x_i given all other variables:

$$P(\xi) \approx \prod_{j} P(x_j \mid x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n).$$

pseudolikelihood

This approximation leads to the *pseudolikelihood* objective:

$$\ell_{\mathrm{PL}}(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \sum_{m} \sum_{j} \ln P(x_j[m] \mid \boldsymbol{x}_{-j}[m], \boldsymbol{\theta}), \tag{20.20}$$

where $\boldsymbol{x}_{-j}$ stands for $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Intuitively, this objective measures our ability to predict each variable in the model given a full observation over all other variables. The predictive model takes a form that generalizes the *multinomial logistic CPD* of definition 5.10 and is identical to it in the case where the network contains only pairwise features — factors over edges in the network. As usual, we can use the conditional independence properties in the network to simplify this expression, removing from the right-hand side of $P(X_j \mid \boldsymbol{X}_{-j})$ any variable that is not a neighbor of X_j.

multinomial logistic CPD

At first glance, this objective may appear to be more complex than the likelihood objective. However, a closer examination shows that we have eliminated the exponential summation over instances with several summations, each of which is far more tractable. In particular:

$$\begin{aligned} P(x_j \mid \boldsymbol{x}_{-j}) = \frac{P(x_j, \boldsymbol{x}_{-j})}{P(\boldsymbol{x}_{-j})} &= \frac{\tilde{P}(x_j, \boldsymbol{x}_{-j})}{\tilde{P}(\boldsymbol{x}_{-j})} \\ &= \frac{\tilde{P}(x_j, \boldsymbol{x}_{-j})}{\sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j})}. \end{aligned}$$

The critical feature of this expression is that the global partition function has disappeared, and instead we have a local partition function that requires summing only over the values of X_j.

The contrastive perspective that we described earlier provides an alternative insight on this derivation. Consider the pseudolikelihood objective applied to a single data instance ξ:

$$\begin{aligned}\sum_j \ln P(x_j \mid \boldsymbol{x}_{-j}) &= \sum_j \left(\ln \tilde{P}(x_j, \boldsymbol{x}_{-j}) - \ln \sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j}) \right) \\ &= \sum_j \left(\ln \tilde{P}(\xi) - \ln \sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j}) \right).\end{aligned}$$

Each of the terms in this final summation is a contrastive term, where we aim to increase the difference between the log-measure of our training instance ξ and an aggregate of the log-measures of instances that differ from ξ in the assignment to precisely one variable. In other words, we are increasing the contrast between our training instance ξ and the instances in a local neighborhood around it.

We can further simplify each of the summands in this expression, obtaining:

$$\ln P(x_j \mid \boldsymbol{x}_{-j}) = \left(\sum_{i \,:\, \mathit{Scope}[f_i] \ni X_j} \theta_i f_i(x_j, \boldsymbol{x}_{-j}) \right) - \ln \left(\sum_{x'_j} \exp \left\{ \sum_{i \,:\, \mathit{Scope}[f_i] \ni X_j} \theta_i f_i(x'_j, \boldsymbol{x}_{-j}) \right\} \right). \tag{20.21}$$

Each of these terms is precisely a log-conditional-likelihood term for a Markov network over a single variable X_j, conditioned on all the remaining variables. Thus, it follows from corollary 20.2 that the function is concave in the parameters $\boldsymbol{\theta}$. Since a sum of concave functions is also concave, we have that the pseudolikelihood objective of equation (20.20) is concave. Thus, we are guaranteed that gradient ascent over this objective will converge to the global maximum.

To compute the gradient, we use equation (20.21), to obtain:

$$\frac{\partial}{\partial \theta_i} \ln P(x_j \mid \boldsymbol{x}_{-j}) = f_i(x_j, \boldsymbol{x}_{-j}) - \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}}(X_j \mid \boldsymbol{x}_{-j})} \big[f_i(x'_j, \boldsymbol{x}_{-j}) \big]. \tag{20.22}$$

If X_j is not in the scope of f_i, then $f_i(x_j, \boldsymbol{x}_{-j}) = f_i(x'_j, \boldsymbol{x}_{-j})$ for any x'_j, and the two terms are identical, making the derivative 0. Inserting this expression into equation (20.20), we obtain:

Proposition 20.4

$$\frac{\partial}{\partial \theta_i} \ell_{\mathrm{PL}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_{j : X_j \in \mathit{Scope}[f_i]} \left(\frac{1}{M} \sum_m f_i(\xi[m]) - \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}}(X_j \mid \boldsymbol{x}_{-j}[m])} \big[f_i(x'_j, \boldsymbol{x}_{-j}[m]) \big] \right). \tag{20.23}$$

While this term looks somewhat more involved than the gradient of the likelihood in equation (20.4), it is much easier to compute: each of the expectation terms requires a summation

over only a single random variable X_j, conditioned on all of its neighbors, a computation that can generally be performed very efficiently.

What is the relationship between maximum likelihood estimation and maximum pseudo-likelihood? In one specific situation, the two estimators return the same set of parameters.

Theorem 20.3 *Assume that our data are generated by a log-linear model $P_{\boldsymbol{\theta}^*}$ that is of the form of equation (20.1). Then, as the number of data instances M goes to infinity, with probability that approaches 1, $\boldsymbol{\theta}^*$ is a global optimum of the pseudolikelihood objective of equation (20.20).*

PROOF To prove the result, we need to show that because the size of the data set tends to infinity, the gradient of the pseudolikelihood objective at $\boldsymbol{\theta}^*$ tends to zero. Owing to the concavity of the objective, this equality implies that $\boldsymbol{\theta}^*$ is necessarily an optimum of the pseudolikelihood objective. We provide a somewhat informal sketch of the gradient argument, but one that contains all the essential ideas.

Because $M \longrightarrow \infty$, the empirical distribution $\hat{P}$ gets arbitrarily close to $P_{\boldsymbol{\theta}^*}$. Thus, the statistics in the data are precisely representative of their expectations relative to $P_{\boldsymbol{\theta}^*}$. Now, consider one of the summands in equation (20.23), associated with a feature f_i. Due to the convergence of the sufficient statistics,

$$\frac{1}{M} \sum_m f_i(\xi[m]) \longrightarrow \boldsymbol{E}_{\xi \sim P_{\boldsymbol{\theta}^*}(\mathcal{X})}[f_i(\xi)].$$

Conversely,

$$\begin{aligned}
&\frac{1}{M} \sum_m \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}^*}(X_j \mid \boldsymbol{x}_{-j}[m])} \left[f_i(x'_j, \boldsymbol{x}_{-j}[m]) \right] \\
&\quad = \quad \sum_{\boldsymbol{x}_{-j}} P_{\mathcal{D}}(\boldsymbol{x}_{-j}) \sum_{x'_j} P_{\boldsymbol{\theta}^*}(x'_j \mid \boldsymbol{x}_{-j}) f_i(x'_j, \boldsymbol{x}_{-j}) \\
&\quad \longrightarrow \quad \sum_{\boldsymbol{x}_{-j}} P_{\boldsymbol{\theta}^*}(\boldsymbol{x}_{-j}) \sum_{x'_j} P_{\boldsymbol{\theta}^*}(x'_j \mid \boldsymbol{x}_{-j}) f_i(x'_j, \boldsymbol{x}_{-j}) \\
&\quad = \quad \boldsymbol{E}_{\xi \sim P_{\boldsymbol{\theta}^*}}[f_i(\xi)].
\end{aligned}$$

Thus, at the limit, the empirical and expected counts are equal, so that the gradient is zero. ■

consistent

This theorem states that, like the likelihood objective, the pseudolikelihood objective is also *consistent*. If we assume that the models are nondegenerate so that the two objectives are strongly concave, the maxima are unique, and hence the two objectives have the same maximum.

While this result is an important one, it is important to be cognizant of its limitations. In particular, we note that the two assumptions are central to this argument. First, in order for the empirical and expected counts to match, the model being learned needs to be sufficiently expressive to represent the generating distribution. Second, the data distribution needs to be close enough to the generating distribution to be well captured within the model, a situation that is only guaranteed to happen at the large-sample limit. Without these assumptions, the two objectives can have quite different optima that lead to different results.

In practice, these assumptions rarely hold: our model is never a perfect representation of the true underlying distribution, and we often do not have enough data to be close to the large sample limit. Therefore, one must consider the question of how good this objective is in practice. The answer to this question depends partly on the types of queries for which we intend to use the model.  **If we plan to run queries where we condition on most of the variables and query the values of only a few, the pseudolikelihood objective is a very close match to the type of predictions we would like to make, and therefore pseudolikelihood may well provide a better training objective than likelihood.** For example, if we are trying to learn a Markov network for collaborative filtering (box 18.C), we generally take the user's preference for all items except the query item to be observed. **Conversely, if a typical query involves most or all of the variables in the model, the likelihood objective is more appropriate.** For example, if we are trying to learn a model for image segmentation (box 4.B), the segment value of all of the pixels is unobserved. (We note that this last application is a CRF, where we would generally use a conditional likelihood objective, conditioned on the actual pixel values.) In this case, a (conditional) likelihood is a more appropriate objective than the (conditional) pseudolikelihood.

However, even in cases where the likelihood is the more appropriate objective, we may have to resort to pseudolikelihood for computational reasons. In many cases, this objective performs surprisingly well. However, in others, it can provide a fairly poor approximation.

Example 20.3

Consider a Markov network over three variables X_1, X_2, Y, where each pair is connected by an edge. Assume that X_1, X_2 are very highly correlated (almost identical) and both are somewhat (but not as strongly) correlated with Y. In this case, the best predictor for X_1 is X_2, and vice versa, so the pseudolikelihood objective is likely to overestimate significantly the parameters on the X_1—X_2, and almost entirely dismiss the X_1—Y and X_2—Y edges. The resulting model would be an excellent predictor for X_2 when X_1 is observed, but virtually useless when only Y and not X_1 is observed. ■

This example is typical of a general phenomenon: Pseudolikelihood, by assuming that each variable's local neighborhood is fully observed, is less able to exploit information obtained from weaker or longer-range dependencies in the distribution.

generalized pseudolikelihood

This limitation also suggests a spectrum of approaches known as *generalized pseudolikelihood*, which can reduce the extent of this problem. In particular, in the objective of equation (20.20), rather than using a product of terms over individual variables, we can consider terms where the left-hand side consists of several variables, conditioned on the rest. More precisely, we can define a set of subsets of variables $\{\boldsymbol{X}_s : s \in \mathcal{S}\}$, and then define an objective:

$$\ell_{\text{GPL}}(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \sum_m \sum_s \ln P(\boldsymbol{x}_s[m] \mid \boldsymbol{x}_{-s}[m], \boldsymbol{\theta}), \tag{20.24}$$

where $\boldsymbol{X}_{-s} = \mathcal{X} - \boldsymbol{X}_s$.

Clearly, there are many possible choices of subsets $\{\boldsymbol{X}_s\}$. For different such choices, this expression generalizes several objectives: the likelihood, the pseudolikelihood, and even the conditional likelihood. When variables are together in the same subset $\boldsymbol{X}_s$, the relationship between them is subject (at least in part) to a likelihood-like objective, which tends to induce a more correct model of the joint distribution over them. However, as for the likelihood,

this objective requires that we compute expected counts over the variables in each $\boldsymbol{X}_s$ given an assignment to $\boldsymbol{X}_{-s}$. Thus, the choice of $\boldsymbol{X}_s$ offers a trade-off between "accuracy" and computational cost. One common choice of subsets is the set of all cliques in the Markov networks, which guarantees that the factor associated with each clique is optimized in at least one likelihood-like term in the objective.

20.6.2 Contrastive Optimization Criteria

As we discussed, both likelihood and pseudolikelihood can be viewed as attempting to increase the "log-probability gap" between the log-probability of the observed instances in $\mathcal{D}$ and the logarithm of the aggregate probability of a set of instances. Building on this perspective, one can construct a range of methods that aim to increase the log-probability gap between $\mathcal{D}$ and some other instances. The intuition is that, by driving the probability of the observed data higher relative to other instances, we are tuning our parameters to predict the data better.

More precisely, consider again the case of a single training instance ξ. We can define a "contrastive" objective where we aim to maximize the log-probability gap:

$$\left(\ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi' \mid \boldsymbol{\theta})\right),$$

where ξ' is some other instance, whose selection we discuss shortly. Importantly, this expression takes a very simple form:

$$\left(\ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi' \mid \boldsymbol{\theta})\right) = \boldsymbol{\theta}^T[\boldsymbol{f}(\xi) - \boldsymbol{f}(\xi')]. \tag{20.25}$$

Note that, for a fixed instantiation ξ', this expression is a linear function of $\boldsymbol{\theta}$ and hence is unbounded. Thus, in order for this type of function to provide a coherent optimization objective, the choice of ξ' will generally have to change throughout the course of the optimization. Even then, we must take care to prevent the parameters from growing unboundedly, an easy way of arbitrarily increasing the objective.

One can construct many variants of this type of method. Here, we briefly survey two that have been particularly useful in practice.

20.6.2.1 Contrastive Divergence

contrastive divergence

One approach whose popularity has recently grown is the *contrastive divergence* method. In this method, we "contrast" our data instances $\mathcal{D}$ with a set of randomly perturbed "neighbors" $\mathcal{D}^-$. In particular, we aim to maximize:

$$\ell_{\mathrm{CD}}(\boldsymbol{\theta} : \mathcal{D} \| \mathcal{D}^-) = \left[\boldsymbol{E}_{\xi \sim \hat{P}_{\mathcal{D}}}\left[\ln \tilde{P}_{\boldsymbol{\theta}}(\xi)\right] - \boldsymbol{E}_{\xi \sim \hat{P}_{\mathcal{D}^-}}\left[\ln \tilde{P}_{\boldsymbol{\theta}}(\xi)\right]\right], \tag{20.26}$$

where $\hat{P}_{\mathcal{D}}$ and $\hat{P}_{\mathcal{D}^-}$ are the empirical distributions relative to $\mathcal{D}$ and $\mathcal{D}^-$, respectively.

As we discussed, the set of "contrasted" instances $\mathcal{D}^-$ will necessarily differ at different stages in the search. Given a current parameterization $\boldsymbol{\theta}$, what is a good choice of instances to which we want to contrast our data instances $\mathcal{D}$? One intuition is that we want to move our parameters $\boldsymbol{\theta}$ in a direction that increases the probability of instances in $\mathcal{D}$ relative to "typical" instances in our current distribution; that is, we want to increase the probability gap between instances

$\xi \in \mathcal{D}$ and instances ξ sampled randomly from $P_{\boldsymbol{\theta}}$. Thus, we can generate a contrastive set $\mathcal{D}^-$ by sampling from $P_{\boldsymbol{\theta}}$, and then maximizing the objective in equation (20.26).

How do we sample from $\mathcal{P}_{\boldsymbol{\theta}}$? As in section 12.3, we can run a Markov chain defined by the Markov network $P_{\boldsymbol{\theta}}$, using, for example, Gibbs sampling, and initializing from the instances in $\mathcal{D}$; once the chain mixes, we can collect samples from the distribution $P_{\boldsymbol{\theta}}$. Unfortunately, sampling from the chain for long enough to achieve mixing usually takes far too long to be feasible as the inner loop of a learning algorithm. However, there is an alternative approach, which is both less expensive and more robust. Rather than run the chain defined by $P_{\boldsymbol{\theta}}$ to convergence, we initialize from the instances in $\mathcal{D}$, and run the chain only for a few steps; we then use the instances generated by these short sampling runs to define $\mathcal{D}^-$.

Intuitively, this approach has significant appeal: We want our model to give high probability to the instances in $\mathcal{D}$; our current parameters, initialized at $\mathcal{D}$, are causing us to move away from the instances in $\mathcal{D}$. Thus, we want to move our parameters in a direction that increases the probability of the instances in $\mathcal{D}$ relative to the "perturbed" instances in $\mathcal{D}^-$.

The gradient of this objective is also very intuitive, and easy to compute:

$$\frac{\partial}{\partial \theta_i} \ell_{\text{CD}}(\boldsymbol{\theta} : \mathcal{D} \| \mathcal{D}^-) = \boldsymbol{E}_{\hat{P}_{\mathcal{D}}}[f_i(\mathcal{X})] - \boldsymbol{E}_{\hat{P}_{\mathcal{D}^-}}[f_i(\mathcal{X})]. \tag{20.27}$$

Note that, if we run the Markov chain to the limit, the samples in $\mathcal{D}^-$ are generated from $P_{\boldsymbol{\theta}}$; in this case, the second term in this difference converges to $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_i]$, which is precisely the second term in the gradient of the log-likelihood objective in equation (20.4). Thus, at the limit of the Markov chain, this learning procedure is equivalent (on expectation) to maximizing the log-likelihood objective. However, in practice, the approximation that we get by taking only a few steps in the Markov chain provides a good direction for the search, at far lower computational cost. In fact, empirically it appears that, because we are taking fewer sampling steps, there is less variance in our estimation of the gradient, leading to more robust convergence.

20.6.2.2 Margin-Based Training ⋆

A very different intuition arises in settings where our goal is to use the learned network for predicting a MAP assignment. For example in our image segmentation application of box 4.B, we want to use the learned network to predict a single high-probability assignment to the pixels that will encode our final segmentation output. This type of reasoning only arises in the context of conditional queries, since otherwise there is only a single MAP assignment (in the unconditioned network). Thus, we describe the objective in this section in the context of conditional Markov networks.

Recall that, in this setting, our training set consists of a set of pairs $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^{M}$. Given an observation $\boldsymbol{x}[m]$, we would like our learned model to give the highest probability to $\boldsymbol{y}[m]$. In other words, we would like the probability $P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m])$ to be higher than any other probability $P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m])$ for $\boldsymbol{y} \neq \boldsymbol{y}[m]$. In fact, to increase our confidence in this prediction, we would like to increase the log-probability gap as much as possible, by increasing:

$$\ln P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \left[\max_{\boldsymbol{y} \neq \boldsymbol{y}[m]} \ln P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m]) \right].$$

This difference between the log-probability of the target assignment $\boldsymbol{y}[m]$ and that of the "next best" assignment is called the *margin*. The higher the margin, the more confident the model is

margin-based estimation

in selecting $\boldsymbol{y}[m]$. Roughly speaking, *margin-based estimation* methods usually aim to maximize the margin.

One way of formulating this of *max-margin* objective as an optimization problem is as follows:

$$\begin{array}{ll} \textbf{Find} & \gamma, \boldsymbol{\theta} \\ \textbf{maximizing} & \gamma \\ \textbf{subject to} & \\ & \ln P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \ln P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m]) \quad \geq \quad \gamma \qquad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m]. \end{array}$$

The objective here is to maximize a single parameter γ, which encodes the worst-case margin over all data instances, by virtue of the constraints, which impose that the log-probability gap between $\boldsymbol{y}[m]$ and any other assignment $\boldsymbol{y}$ (given $\boldsymbol{x}[m]$) is at least γ. Importantly, due to equation (20.25), the first set of constraints can be rewritten in a simple linear form:

$$\boldsymbol{\theta}^T(\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq \gamma.$$

With this reformulation of the constraints, it becomes clear that, if we find any solution that achieves a positive margin, we can increase the margin unboundedly simply by multiplying all the parameters through by a positive constant factor. To make the objective coherent, we can bound the magnitude of the parameters by constraining their L_2-norm: $\|\boldsymbol{\theta}\|_2^2 = \boldsymbol{\theta}^T\boldsymbol{\theta} = \sum_i \theta_i^2 = 1$; or, equivalently, we can decide on a fixed margin and try to reduce the magnitude of the parameters as much as possible. With the latter approach, we obtain the following optimization problem:

Simple-Max-Margin:

$$\begin{array}{ll} \textbf{Find} & \boldsymbol{\theta} \\ \textbf{minimizing} & \|\boldsymbol{\theta}\|_2^2 \\ \textbf{subject to} & \\ & \boldsymbol{\theta}^T(\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq 1 \qquad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m] \end{array}$$

quadratic program

convex optimization

At some level, this objective is simple: it is a *quadratic program* (QP) with linear constraints, and hence is a convex problem that can be solved using a variety of *convex optimization* methods. However, a more careful examination reveals that the problem contains a constraint for every m, and (more importantly) for every assignment $\boldsymbol{y} \neq \boldsymbol{y}[m]$. Thus, the number of constraints is exponential in the number of variables $\boldsymbol{Y}$, generally an intractable number.

constraint generation

However, these are not arbitrary constraints: the structure of the underlying Markov network is reflected in the form of the constraints, opening the way toward efficient solution algorithms. One simple approach uses *constraint generation*, a general-purpose method for solving optimization problems with a large number of constraints. Constraint generation is an iterative method, which repeatedly solves for $\boldsymbol{\theta}$, each time using a larger set of constraints. Assume we have some algorithm for performing constrained optimization. We initially run this algorithm using none of the margin constraints, and obtain the optimal solution $\boldsymbol{\theta}^0$. In most cases, this solution will not satisfy many of the margin constraints, and it is thus not a feasible solution to our original QP. We add one or more constraints that are violated by $\boldsymbol{\theta}^0$ into a set of *active constraints*. We now repeat the constrained optimization process to obtain a new solution $\boldsymbol{\theta}^1$, which is guaranteed

to satisfy the active constraints. We again examine the constraints, find ones that are violated, and add them to our active constraints. This process repeats until no constraints are violated by our solution. Clearly, since we only add constraints, this procedure is guaranteed to terminate: eventually there will be no more constraints to add. Moreover, when it terminates, the solution is guaranteed to be optimal: At any iteration, the optimization procedure is solving a relaxed problem, whose value is at least as good as that of the fully constrained problem. If the optimal solution to this relaxed problem happens to satisfy all of the constraints, no better solution can be found to the fully constrained problem.

This description leaves unanswered two important questions. First, how many constraints we will have to add before this process terminates? Fortunately, it can be shown that, under reasonable assumptions, at most a polynomial number of constraints will need to be added prior to termination. Second, how do we find violated constraints without exhaustively enumerating and checking every one? As we now show, we can perform this computation by running MAP inference in the Markov network induced by our current parameterization $\boldsymbol{\theta}$. To see how, recall that we either want to show that

$$\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) \geq \ln \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]) + 1$$

for every $\boldsymbol{y} \in \mathit{Val}(\boldsymbol{Y})$ except $\boldsymbol{y}[m]$, or we want to find an assignment $\boldsymbol{y}$ that violates this inequality constraint. Let

$$\boldsymbol{y}^{\mathit{map}} = \arg \max_{\boldsymbol{y} \neq \boldsymbol{y}[m]} \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]).$$

There are now two cases: If $\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) < \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1$, then this is a violated constraint, which can be added to our constraint set. Alternatively, if $\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) > \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1$, then, due to the selection of $\boldsymbol{y}^{\mathit{map}}$, we are guaranteed that

$$\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) > \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1 \geq \ln \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]) + 1,$$

for every $\boldsymbol{y} \neq \boldsymbol{y}[m]$. That is, in this second case, all of the exponentially many constraints for the m'th data instance are guaranteed to be satisfied. As written, the task of finding $\boldsymbol{y}^{\mathit{map}}$ is not a simple MAP computation, due to the constraint that $\boldsymbol{y}^{\mathit{map}} \neq \boldsymbol{y}[m]$. However, this difficulty arises only in the case where the MAP assignment is $\boldsymbol{y}[m]$, in which case we need only find the second-best assignment. Fortunately, it is not difficult to adapt most MAP solution methods to the task of finding the second-best assignment (see, for example, exercise 13.5).

The use of MAP rather than sum-product as the inference algorithm used in the inner loop of the learning algorithm can be of significance. As we discussed, MAP inference admits the use of more efficient optimization algorithms that are not applicable to sum-product. In fact, as we discussed in section 13.6, there are even cases where sum-product is intractable, whereas MAP can be solved in polynomial time.

However, the margin constraints we use here fail to address two important issues. First, we are not guaranteed that there exists a model that can correctly select $\boldsymbol{y}[m]$ as the MAP assignment for every data instance m: First, our training data may be noisy, in which case $\boldsymbol{y}[m]$ may not be the actual desired assignment. More importantly, our model may not be expressive enough to always pick the desired target assignment (and the "simple" solution of increasing its expressive power may lead to overfitting). Because of the worst-case nature of our optimization objective, when we cannot achieve a positive margin for every data instance, there is no longer

any incentive in getting a better margin for those instances where a positive margin can be achieved. Thus, the solution we obtain becomes meaningless. To address this problem, we must allow for instances to have a nonpositive margin and simply penalize such exceptions in the objective; the penalization takes the form of *slack variables* η_m that measure the extent of the violation for the m'th data instances. This approach allows the optimization to trade off errors in the labels of a few instances for a better solution overall.

A second, related problem arises from our requirement that our model achieve a uniform margin for all $\boldsymbol{y} \neq \boldsymbol{y}[m]$. To see why this requirement can be problematic, consider again our image segmentation problem. Here, $\boldsymbol{x}[m]$ are features derived from the image, $\boldsymbol{y}[m]$ is our "ground truth" segmentation, and other assignments $\boldsymbol{y}$ are other candidate segmentations. Some of these candidate segmentations differ from $\boldsymbol{y}[m]$ only in very limited ways (perhaps a few pixels are assigned a different label). In this case, we expect that a reasonable model $P_{\boldsymbol{\theta}}$ will ascribe a probability to these "almost-correct" candidates that is very close to the probability of the ground truth. If so, it will be difficult to find a good model that achieves a high margin. Again, due to the worst-case nature of the objective, this can lead to inferior models. We address this concern by allowing the required margin $\ln P(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \ln P(\boldsymbol{y} \mid \boldsymbol{x}[m])$ to vary with the "distance" between $\boldsymbol{y}[m]$ and $\boldsymbol{y}$, with assignments $\boldsymbol{y}$ that are more similar to $\boldsymbol{y}[m]$ requiring a smaller margin. In particular, using the ideas of the *Hamming loss*, we can define $\Delta_m(\boldsymbol{y})$ to be the number of variables $Y_i \in \boldsymbol{Y}$ such that $y_i \neq y_i[m]$, and require that the margin increase linearly in this discrepancy.

Hamming loss

Putting these two modifications together, we obtain our final optimization problem:

Max-Margin:

Find $\boldsymbol{\theta}$

maximizing $\|\boldsymbol{\theta}\|_2^2 + C \sum_m \eta_m$

subject to

$$\boldsymbol{\theta}^T (\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq \Delta_m(\boldsymbol{y}) - \eta_m \quad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m].$$

Here, C is a constant that determines the balance between the two parts of the objective: how much we choose to penalize mistakes (negative margins) for some instances, versus achieving a higher margin overall.

Fortunately, the same constraint generation approach that we discussed can also be applied in this case (see exercise 20.14).

20.7 Structure Learning

model selection

We now move to the problem of *model selection*: learning a network structure from data. As usual, there are two types of solution to this problem: the constraint-based approaches, which search for a graph structure satisfying the independence assumptions that we observe in the empirical distribution; and the score-based approaches, which define an objective function for different models, and then search for a high-scoring model.

From one perspective, the constraint-based approaches appear relatively more advantageous here than they did in the case of Bayesian network learning. First, the independencies associated with separation in a Markov network are much simpler than those associated with d-separation

in a Bayesian network; therefore, the algorithms for inferring the structure are much simpler here. Second, recall that all of our scoring functions were based on the likelihood function; here, unlike in the case of Bayesian networks, even evaluating the likelihood function is a computationally expensive procedure, and often an intractable one.

On the other side, the disadvantage of the constraint-based approaches remains: their lack of robustness to statistical noise in the empirical distribution, which can give rise to incorrect independence assumptions. We also note that the constraint based approaches produce only a structure, and not a fully specified model of a distribution. To obtain such a distribution, we need to perform parameter estimation, so that we eventually encounter the computational costs associated with the likelihood function. Finally, in the context of Markov network learning, it is not clear that learning the global independence structure is necessarily the appropriate problem. In the context of learning Bayesian networks we distinguished between learning the global structure (the directed graph) and local structure (the form of each CPD). In learning undirected models we can similarly consider both the problem of learning the undirected graph structure and the particular set of factors or features that represent the parameterization of the graph. Here, however, it is quite common to find distributions that have a compact factorization yet have a complex graph structure. One extreme example is the fully connected network with pairwise potentials. Thus, in many domains we want to learn the factorization of the joint distribution, which often cannot be deduced from the global independence assumptions.

We will review both types of approach, but we will focus most of the discussion on score-based approaches, since these have received more attention.

20.7.1 Structure Learning Using Independence Tests

constraint-based structure learning

independence tests

We first consider the idea of *constraint-based structure learning*. Recall that the structure of a Markov network specifies a set of independence assertions. We now show how we can use *independence tests* to reconstruct the Markov network structure. For this discussion, assume that the generating distribution P^* is positive and can be represented as a Markov network $\mathcal{H}^*$ that is a perfect map of P^*. Thus, we want to perform a set of independence tests on P^* and recover $\mathcal{H}^*$. To make the problem tractable, we further assume that the degree of nodes in $\mathcal{H}^*$ is at most d^*.

Recall that in section 4.3.2 we considered three set sets of independencies that characterize a Markov network: global independencies that include all consequences of separation in the graph; Markov independencies that describe the independence of each variable X from the rest of the variables given its Markov blanket; and pairwise independencies that describe the independence of each nonadjacent pair of variables X, Y given all other variables. We showed there that these three definitions are equivalent in positive distributions.

Can we use any of these concepts to recover the structure of $\mathcal{H}^*$? Intuitively, we would prefer to examine a smaller set of independencies, since they would require fewer independence tests. Thus, we should focus either on the local Markov independencies or pairwise independencies.

local Markov independencies

Recall that *local Markov independencies* are of the form

$$(X \perp \mathcal{X} - \{X\} - \mathrm{MB}_{\mathcal{H}^*}(X) \mid \mathrm{MB}_{\mathcal{H}^*}(X)) \quad \forall X$$

pairwise independencies

and *pairwise independencies* are of the form

$$(X \perp Y \mid \mathcal{X} - \{X, Y\}) \;\; \forall (X — Y) \notin \mathcal{H}.$$

Unfortunately, as written, neither of these sets of independencies can be checked tractably, since both involve the entire set of variables $\mathcal{X}$ and hence require measuring the probability of exponentially many events. The computational infeasibility of this requirement is obvious. But equally problematic are the statistical issues: these independence assertions are evaluated not on the true distribution, but on the empirical distribution. Independencies that involve many variables lead to fragmentation of the data, and are much harder to evaluate without error. To estimate the distribution sufficiently well as to evaluate these independencies reliably, we would need exponentially many data points.

Thus, we need to consider alternative sets of independencies that involve only smaller subsets of variables. Several such approaches have been proposed; we review only one, as an example. Consider the network $\mathcal{H}^*$. Clearly, if X and Y are not neighbors in $\mathcal{H}^*$, then they are separated by the *Markov blanket* $\mathrm{MB}_{\mathcal{H}^*}(X)$ and also by $\mathrm{MB}_{\mathcal{H}^*}(Y)$. Thus, we can find a set $\boldsymbol{Z}$ with $|\boldsymbol{Z}| \leq \min(|\mathrm{MB}_{\mathcal{H}^*}(X)|, |\mathrm{MB}_{\mathcal{H}^*}(Y)|)$ so that $sep_{\mathcal{H}^*}(X; Y \mid \boldsymbol{Z})$ holds. On the other hand, if X and Y are neighbors in $\mathcal{H}^*$, then we cannot find such a set $\boldsymbol{Z}$. Because $\mathcal{H}^*$ is a perfect map of P^*, we can show that

Markov blanket

$$X — Y \notin \mathcal{H}^* \text{ if and only if } \exists \boldsymbol{Z}, |\boldsymbol{z}| \leq d^* \& P^* \models (X \perp Y \mid \boldsymbol{Z}).$$

Thus, we can determine whether $X — Y$ is in $\mathcal{H}^*$ using $\sum_{k=0}^{d^*} \binom{n-2}{k}$ independence tests. Each of these independence tests involves only $d^* + 2$ variables, which, for low values of d^*, can be tractable. We have already encountered this test in section 3.4.3.1, as part of our Bayesian network construction procedure. If fact, it is not hard to show that, given our assumptions and perfect independence tests, the Build-PMap-Skeleton procedure of algorithm 3.3 reconstructs the correct Markov structure $\mathcal{H}^*$ (exercise 20.15).

This procedure uses a polynomial number of tests. Thus, the procedure runs in polynomial time. Moreover, if the probability of a false answer in any single independence test is at most ϵ, then the probability that any one of the independence tests fails is at most $\sum_{k=0}^{d^*} \binom{n-2}{k} \epsilon$. Therefore, for sufficiently small ϵ, we can use this analysis to prove that we can reconstruct the correct network structure $\mathcal{H}^*$ with high probability.

While this result is satisfying at some level, there are significant limitations. **First, the number of samples required to obtain correct answers for all of the independence tests can be very large in practice. Second, the correctness of the algorithm is based on several important assumptions: that there is a Markov network that is a perfect map of P^*; that this network has a bounded degree; and that we have enough data to obtain reliable answers to the independence tests. When these assumptions are violated, this algorithm can learn incorrect network structures.**

Example 20.4

Assume that the underlying distribution P^* is a Bayesian network with a v-structure $X \rightarrow Z \leftarrow Y$. We showed in section 3.4.3 that, assuming perfect independence tests, Build-PMap-Skeleton *learns the skeleton of $\mathcal{G}^*$. However, the Markov network $\mathcal{H}^*$ that is an I-map for P^* is the moralized network, which contains, in addition to the skeleton edges, edges between parents of a joint child. These edges will not be learned correctly by this procedure. In particular, we have that $(X \perp Y \mid \emptyset)$ holds, and so the algorithm will allow us to remove the edge between X and Y, even though it exists in the true network $\mathcal{H}^*$.* ■

The failure in this example results from the fact that the distribution P^* does not have a perfect

map that is a Markov network. Because many real-life distributions do not have a perfect map that is a compact graph, the applicability of this approach can be limited.

Moreover, as we discussed, this approach focuses solely on reconstructing the network structure and does not attempt to learn the the structure of the factorization, or to estimate the parameters. In particular, we may not have enough data to reliably estimate parameters for the structure learned by this procedure, limiting its usability in practice. Nevertheless, as in the case of Bayesian network structure learning, constraint-based approaches can be a useful tool for obtaining qualitative insight into the global structure of the distribution, and as a starting point for the search in the score-based methods.

20.7.2 Score-Based Learning: Hypothesis Spaces

hypothesis space

We now move to the score-based structure learning approach. As we discussed earlier, this approach formulates structure learning as an optimization problem: We define a *hypothesis space* consisting of a set of possible networks; we also define an objective function, which is used to score different candidate networks; and then we construct a search algorithm that attempts to identify a high-scoring network in the hypothesis space. We begin in this section by discussing the choice of hypothesis space for learning Markov networks. We discuss objective functions and the search strategy in subsequent sections.

There are several ways of formulating the search space for Markov networks, which vary in terms of the granularity at which they consider the network parameterization. At the coarsest-grained, we can pose the hypothesis space as the space of different structures of the Markov network itself and measure the model complexity in terms of the size of the cliques in the network. At the next level, we can consider parameterizations at the level of the factor graph, and measure complexity in terms of the sizes of the factors in this graph. At the finest level of granularity, we can consider a search space at the level of individual features in a log-linear model, and measure sparsity at the level of features included in the model.

The more fine-grained our hypothesis space, the better it allows us to select a parameterization that matches the properties of our distribution without overfitting. For example, the factor-graph approach allows us to distinguish between a single large factor over k variables and a set of $\binom{k}{2}$ pairwise factors over the same variables, requiring far fewer parameters. The feature-based approach also allows us to distinguish between a full factor over k variables and a single log-linear feature over the same set of variables.

Conversely, the finer-grained spaces can obscure the connection to the network structure, in that sparsity in the space of features selected does not correspond directly to sparsity in the model structure. For example, introducing even a single feature $f(\boldsymbol{d})$ into the model has the structural effect of introducing edges between all of the variables in $\boldsymbol{d}$. Thus, even models with a fairly small number of features can give rise to dense connectivity in the induced network. While this is not a problem from the statistical perspective of reliably estimating the model parameters from limited data, it can give rise to significant problems from the perspective of performing inference in the model. Moreover, a finer-grained hypothesis space also means that search algorithms take smaller steps in the space, potentially increasing the cost of our learning procedure. We will return to some of these issues.

We focus our presentation on the formulation of the search space in terms of log-linear models. Here, we have a set of features Ω, which are those that can potentially have nonzero

weight. Our task is to select a log-linear model structure $\mathcal{M}$, which is defined by some subset $\Phi[\mathcal{M}] \subseteq \Omega$. Let $\Theta[\mathcal{M}]$ be the set of parameterizations $\boldsymbol{\theta}$ that are compatible with the model structure: that is, those where $\theta_i \neq 0$ only if $f_i \in \Phi[\mathcal{M}]$. A structure and a compatible parameterization define a log-linear distribution via:

$$P(\mathcal{X} \mid \mathcal{M}, \boldsymbol{\theta}) = \frac{1}{Z} \exp\left\{ \sum_{i \in \Phi[\mathcal{M}]} \theta_i f_i(\xi) \right\} = \frac{1}{Z} \exp\left\{ \boldsymbol{f}^T \boldsymbol{\theta} \right\},$$

where, because of the compatibility of $\boldsymbol{\theta}$ with $\mathcal{M}$, a feature not in $\Phi[\mathcal{M}]$ does not influence in the final vector product, since it is multiplied by a parameter that is 0.

Regardless of the formulation chosen, we may sometimes wish to impose structural constraints that restrict the set of graph structures that can be selected, in order to ensure that we learn a network with certain sparsity properties. In particular, one choice that has received some attention is to restrict the class of networks learned to those that have a certain bound on the *tree-width*. By placing a tight bound on the tree-width, we prevent an overly dense network from being selected, and thereby reduce the chance of overfitting. Moreover, because models of low tree-width allow exact inference to be performed efficiently (to some extent), this restriction also allows the computational steps required for evaluating the objective during the search to be performed efficiently. However, this approach also has limitations. First, it turns out to be nontrivial to implement, since computing the tree-width of a graph is itself an intractable problem (see theorem 9.7); even keeping the graph under the required width is not simple. Moreover, many of the distributions that arise in real-world applications cannot be well represented by networks of low tree-width.

bounded tree-width

20.7.3 Objective Functions

We now move to considering the objective function that we aim to optimize in the score-based approach. We note that our discussion in this section uses the likelihood function as the basis for the objectives we consider; however, we can also consider similar objectives based on various approximations to the likelihood (see section 20.6); most notably, the pseudolikelihood has been used effectively as a substitute for the likelihood in the context of structure learning, and most of our discussion carries over without change to that setting.

20.7.3.1 Likelihood Function

The most straightforward objective function is the likelihood of the training data. As before, we take the score to be the log-likelihood, defining:

$$\text{score}_L(\mathcal{M} : \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta[\mathcal{M}]} \ln P(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}) = \ell(\langle \mathcal{M}, \hat{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}),$$

where $\hat{\boldsymbol{\theta}}_{\mathcal{M}}$ are the maximum likelihood parameters compatible with $\mathcal{M}$.

The likelihood score measures the fitness of the model to the data. However, for the same reason discussed in chapter 18, it prefers more complex models. In particular, if $\Phi[\mathcal{M}_1] \subset \Phi[\mathcal{M}_2]$ then $\text{score}_L(\mathcal{M}_1 : \mathcal{D}) \leq \text{score}_L(\mathcal{M}_2 : \mathcal{D})$. Typically, this inequality is strict, due to the ability of the richer model to capture noise in the data.

Therefore, the likelihood score can be used only with very strict constraints on the expressive model of the model class that we are considering. Examples include bounds on the structure of the Markov network (for example, networks with low tree-width) or on the number of features used. A second option, which also provides some regularization of parameter values, is to use an alternative objective that penalizes the likelihood in order to avoid overfitting.

20.7.3.2 Bayesian Scores

Bayesian score

Recall that, for Bayesian networks, we used a *Bayesian score*, whose primary term is a marginal likelihood that integrates the likelihood over all possible network parameterizations: $\int P(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{M}) d\boldsymbol{\theta}$. This score accounts for our uncertainty over parameters using a Bayesian prior; it avoided overfitting by preventing overly optimistic assessments of the model fit to the training data. In the case of Bayesian networks, we could efficiently evaluate the marginal likelihood. In contrast, in the case of undirected models, this quantity is difficult to evaluate, even using approximate inference methods.

BIC score

Instead, we can use asymptotic approximations of the marginal likelihood. The simplest approximation is the *BIC score*:

$$\text{score}_{BIC}(\mathcal{M} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \hat{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) - \frac{\dim(\mathcal{M})}{2} \ln M,$$

model dimension

where $\dim(\mathcal{M})$ is the *dimension* of the model and M the number of instances in $\mathcal{D}$. This quantity measures the degrees of freedom of our parameter space. When the model has nonredundant features, $\dim(\mathcal{M})$ is exactly the number of features. When there is redundancy, the dimension is smaller than the number of features. Formally, it is the rank of the matrix whose rows are complete assignments ξ_i to $\mathcal{X}$, whose columns are features f_j, and whose entries are $f_j(\xi_i)$. This matrix, however, is exponential in the number of variables, and therefore its rank cannot be computed efficiently. Nonetheless, we can often estimate the number of nonredundant parameters in the model. As a very coarse upper bound, we note that the number of nonredundant features is always upper-bounded by the size of the full table representation of the Markov network, which is the total number of entries in the factors.

Laplace approximation

The BIC approximation penalizes each degree of freedom (that is, free parameter) by a fixed amount, which may not be the most appropriate penalty. Several more refined alternatives have been proposed. One common choice is the *Laplace approximation*, which provides a more explicit approximation to the marginal likelihood:

$$\text{score}_{Laplace}(\mathcal{M} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \tilde{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) + \ln P(\tilde{\boldsymbol{\theta}}_{\mathcal{M}} \mid \mathcal{M}) + \frac{\dim(\mathcal{M})}{2} \ln(2\pi) - \frac{1}{2} \ln |A|,$$

MAP estimation

where $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$ are the parameters for $\mathcal{M}$ obtained from *MAP estimation*:

$$\tilde{\boldsymbol{\theta}}_{\mathcal{M}} = \arg\max_{\boldsymbol{\theta}} P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) P(\boldsymbol{\theta} \mid \mathcal{M}), \tag{20.28}$$

Hessian

and A is the negative *Hessian* matrix:

$$A_{i,j} = -\frac{\partial}{\partial \theta_i \partial \theta_j} \left(\ell(\langle \mathcal{M}, \boldsymbol{\theta} \rangle : \mathcal{D}) + \ln P(\boldsymbol{\theta} \mid \mathcal{M}) \right),$$

evaluated at the point $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$.

As we discussed in section 19.4.1.1, the Laplace score also takes into account the local shape of the posterior distribution around the MAP parameters. It therefore provides a better approximation than the BIC score. However, as we saw in equation (20.5), to compute the Hessian, we need to evaluate the pairwise covariance of every feature pair given the model, a computation that may be intractable in many cases.

20.7.3.3 Parameter Penalty Scores

An alternative to approximations of the marginal likelihood are methods that simply evaluate the maximum posterior probability

$$\text{score}_{MAP}(\mathcal{M} \ : \ \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta[\mathcal{M}]} \ell(\langle \mathcal{M}, \tilde{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) + \ln P(\tilde{\boldsymbol{\theta}}_{\mathcal{M}} \mid \mathcal{M}), \tag{20.29}$$

where $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$ are the MAP parameters for $\mathcal{M}$, as defined in equation (20.28). One intuition for this type of *MAP score* is that the prior "regularizes" the likelihood, moving it away from the maximum likelihood values. If the likelihood of these parameters is still high, it implies that the model is not too sensitive to particular choice of maximum likelihood parameters, and thus it is more likely to generalize.

MAP score

Although the regularized parameters may achieve generalization, this approach achieves model selection only for certain types of prior. To understand why, note that the MAP score is based on a distribution not over structures, but over parameters. We can view any parameterization $\boldsymbol{\theta}_{\mathcal{M}}$ as a parameterization to the "universal" model defined over our entire set of features Ω: one where features not in $\Phi[\mathcal{M}]$ receive weight 0. Assuming that our parameter prior simply ignores zero weights, we can view our score as simply evaluating different choices of parameterizations $\boldsymbol{\theta}_{\Omega}$ to this universal model.

We have already discussed several parameter priors and their effect on the learned parameters. Most parameter priors are associated with the magnitude of the parameters, rather than the complexity of the graph as a discrete data structure. In particular, as we discussed, although *L_2-regularization* will tend to drive the parameters toward zero, few will actually hit zero, and so structural sparsity will not be achieved. Thus, like the likelihood score, the L_2-regularized MAP objective will generally give rise to a fully connected structure. Therefore, this approach is generally not used in the context of model selection (at least not in isolation).

L_2-regularization

A more appropriate approach for this task is *L_1-regularization*, which does have the effect of driving model parameters toward zero, and thus can give rise to a sparse set of features. In other words, the structure that optimizes the *L_1-MAP score* is not, in general, the universal structure Ω. Indeed, as we will discuss, an L_1 prior has other useful properties when used as the basis for a structure selection objective.

L_1-regularization

L_1-MAP score

However, as we have discussed, feature-level sparsity does not necessarily induce sparsity in the network. An alternative that does tend to have this property is the *block-L_1-regularization*. Here, we partition all the parameters into groups $\boldsymbol{\theta}_i = \{\theta_{i,1}, \ldots, \theta_{i,k_i}\}$ (for $i = 1, \ldots, l$). We now define a variant of the L_1 penalty that tends to make each parameter *group* either go to zero together, or not:

block-L_1-regularization

$$-\sum_{i=1}^{l} \left| \sqrt{\sum_{j=1}^{k_i} \theta_{i,j}^2} \right|. \tag{20.30}$$

To understand the behavior of this penalty term, let us consider its derivative for the simple case where we have two parameters in the same group, so that our expression takes the form $\sqrt{\theta_1^2 + \theta_2^2}$. We now have that:

$$\frac{\partial}{\partial \theta_1}\left[-\sqrt{\theta_1^2 + \theta_2^2}\right] = -\frac{\theta_1}{\sqrt{\theta_1^2 + \theta_2^2}}.$$

We therefore see that, when θ_2 is large, the derivative relative to θ_1 is fairly small, so that there is no pressure on θ_1 to go to 0. Conversely, when θ_2 is small, the derivative relative to θ_1 tends to -1, which essentially gives the same behavior as L_1 regularization. Thus, this prior tends to have the following behavior: if the overall magnitude of the parameters in the group is small, all of them will be forced toward zero; if the overall magnitude is large, there is little downward pressure on any of them.

In our setting, we can naturally apply this prior to give rise to sparsity in network structure. Assume that we are willing to consider, within our network, factors over scopes $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_l$. For each $\boldsymbol{Y}_i$, let $f_{i,j}$, for $j = 1, \ldots, k_i$, be all of the features whose scope is $\boldsymbol{Y}_i$. We now define a block-L_1 prior where we have a block for each set of parameters $\theta_{i1}, \ldots, \theta_{ik_i}$. The result of this prior would be to select together nonzero parameters for an entire set of features associated with a particular scope.

Finally, we note that one can also use multiple penalty terms on the likelihood function. For example, a combination of a parameter penalty and a structure penalty can often provide both regularization of the parameters and a greater bias toward sparse structures.

20.7.4 Optimization Task

Having selected an objective function for our model structure, it remains to address the optimization problem.

20.7.4.1 Greedy Structure Search

local search

As in the approach used for structure learning of general Bayesian networks (section 18.4.3), the external search over structures is generally implemented by a form of *local search*. Indeed, the general form of the algorithms in appendix A.4.2 applies to our feature-based view of Markov network learning. The general template is shown in algorithm 20.1. Roughly speaking, the algorithm maintains a current structure, defined in terms of a set of features $\mathcal{F}$ in our log-linear model. At each point in the search, the algorithm optimizes the model parameters relative to the current feature set and the structure score. Using the current structure and parameters, it estimates the improvement of different structure modification steps. It then selects some subset of modifications to implement, and returns to the parameter optimization step, initializing from the current parameter setting. This process is repeated until a termination condition is reached. This general template can be instantiated in many ways, including the use of different hypothesis spaces (as in section 20.7.2) and different scoring functions (as described in section 20.7.3).

20.7.4.2 Successor Evaluation

Although this approach is straightforward at a high level, there are significant issues with its implementation. Importantly, the reasons that made this approach efficient in the context of

Algorithm 20.1 Greedy score-based structure search algorithm for log-linear models

```
Procedure Greedy-MN-Structure-Search (
  Ω,    // All possible features
  F_0,    // initial set of features
  score(· : D),    // Score
)
1   F' ← F_0    // New feature set
2   θ ← 0
3   do
4     F ← F'
5     θ ← Parameter-Optimize(F, θ, score(· : D))
6       // Find parameters that optimize the score objective, relative to
          current feature set, initializing from the current parameters
7     for each f_k ∈ F such that θ_k = 0
8       F ← F − f_k
9         // Remove inactive features
10    for each operator o applicable to F
11      Let Δ̂_o be the approximate improvement for o
12    Choose some subset O of operators based on Δ̂
13    F' ← O(F)    // Apply selected operators to F
14    while termination condition not reached
15  return (F, θ)
```

score decomposability

Bayesian networks do not apply here. In the case of Bayesian networks, evaluating the score of a candidate structure is a very easy task, which can be executed in closed form, at very low computation cost. Moreover, the Bayesian network score satisfies an important property: it *decomposes* according to the structure of the network. As we discussed, this property has two major implications. First, a local modification to the structure involves changing only a single term in the score (proposition 18.5); second, the change in score incurred by a particular change (for example, adding an edge) remains unchanged after modifications to other parts of the network (proposition 18.6). These properties allowed us to design efficient search procedure that does not need to reevaluate all possible candidates after every step, and that can cache intermediate computations to evaluate candidates in the search space quickly.

Unfortunately, none of these properties hold for Markov networks. For concreteness, consider the likelihood score, which is comparable across both network classes. First, as we discussed, even computing the likelihood of a fully specified model — structure as well as parameters — requires that we run inference for every instance in our training set. Second, to score a structure, we need to estimate the parameters for it, a problem for which there is no closed-form solution. Finally, none of the decomposition properties hold in the case of undirected models. By adding a new feature (or a set of features, for example, a factor), we change the weight ($\sum_i \theta_i f_i(\xi)$) associated with different instances. This change can be decomposed, since it is a linear function of the different features. However, this change also affects the partition function, and, as we saw in the context of parameter estimation, the partition function couples the effects of changes in

one parameter on the other. We can clearly see this phenomenon in figure 20.1, where the effect on the likelihood of modifying $f_1(b^1, c^1)$ clearly depends on the current value of the parameter for $f_2(a^1, b^1)$.

As a consequence, a local search procedure is considerably more expensive in the context of Markov networks. At each stage of the search, we need to evaluate the score for all of the candidates we wish to examine at that point in the search. This evaluation requires that we estimate the parameters for the structure, a process that itself requires multiple iterations of a numerical optimization algorithm, each involving inference over all of the instances in our training set. We can reduce somewhat the computational cost of the algorithm by using the observation that a single change to the structure of the network often does not result in drastic changes to the model. Thus, if we begin our optimization process from the current set of parameters, a reasonably small number of iterations often suffices to achieve convergence to the new set of parameters. Importantly, because all of the parameter objectives described are convex (when we have fully observable data), the initialization has no effect, and convergence to the global optimum remains guaranteed. Thus, this approach simply provides a way of speeding up the convergence to the optimal answer. (We note, however, that this statement holds only when we use exact inference; the choice of initialization can affect the accuracy of some approximate inference algorithms, and therefore the answers that we get.)

Unfortunately, although this observation does reduce the cost, the number of candidate hypotheses at each step is generally quite large. **The cost of running inference on each of the candidate successors is prohibitive, especially in cases where, to fit our target distribution well, we need to consider nontrivial structures. Thus, much of the work on the problem of structure learning for Markov networks has been devoted to reducing the computational cost of evaluating the score of different candidates during the search.** In particular, when evaluating different structure-modification operators in line 11, most algorithms use some heuristic to rank different candidates, rather than computing the exact delta-score of each operator. These heuristic estimates can be used either as the basis for the final selection, or as a way of pruning the set of possible successors, where the high-ranking candidates are then evaluated exactly. This design decision is a trade-off between the quality of our operator selection and the computational cost.

Even with the use of heuristics, the cost of taking a step in the search can be prohibitive, since it requires a reestimation of the network parameters and a reevaluation of the (approximate) delta-score for all of the operators. This suggests that it may be beneficial to select, at each structure modification step (line 12), not a single operator but a subset $\mathcal{O}$ of operators. This approach can greatly reduce the computational cost, but at a cost: our (heuristic) estimate of each operator can deteriorate significantly if we fail to take into account interactions between the effects of different operators. Again, this is a trade-off between the quality and cost of the operator selection.

20.7.4.3 Choice of Scoring Function

As we mentioned, the rough template of algorithm 20.1 can be applied to any objective function. However, the choice of objective function has significant implications on our ability to effectively optimize it. Let us consider several of the choices discussed earlier.

We first recall that both the log-likelihood objective and the L_2-regularized log-likelihood

generally give nonzero values to all parameters. In other words, if we allow the model to consider a set of features $\mathcal{F}$, an optimal model (maximum-likelihood or maximum L_2-regularized likelihood) over $\mathcal{F}$ will give a nonzero value to the parameters for all of these features. In other words, we cannot rely on these objectives to induce sparsity in the model structure. Thus, if we simply want to optimize these objectives, we should simply choose the richest model available in our hypothesis space and then optimize its parameters relative to the chosen objective.

One approach for deriving more compact models is to restrict the class of models to ones with a certain bound on the complexity (for example, networks of bounded tree-width, or with a bound on the number of edges or features allowed). However, these constraints generally introduce nontrivial combinatorial trade-offs between features, giving rise to a search space with multiple local optima, and making it generally intractable to find a globally optimal solution. A second approach is simply to halt the search when the improvement in score (or an approximation to it) obtained by a single step does not exceed a certain threshold. This heuristic is not unreasonable, since good features are generally introduced earlier, and so there is a general trend of diminishing returns. However, there is no guarantee that the solution we obtain is even close to the optimum, since there is no bound on how much the score would improve if we continue to optimize beyond the current step.

Scoring functions that explicitly penalize structure complexity — such as the BIC score or Laplace approximation — also avoid this degeneracy. Here, as in the case of Bayesian networks, we can consider a large hypothesis space and attempt to find the model in this space that optimizes the score. However, due to the discrete nature of the structure penalty, the score is discontinuous and therefore nonconcave. Thus, there is generally no guarantee of convergence to the global optimum. Of course, this limitation was also the case when learning Bayesian networks; as there, it can be somewhat alleviated by methods that avoid local maxima (such as tabu search, random restarts, or data perturbation).

However, in the case of Markov networks, we have another solution available to us, one that avoids the prospect of combinatorial search spaces and the ensuing problem of local optima. This solution is the use of L_1-regularized likelihood. As we discussed, the L_1-regularized likelihood is a concave function that has a unique global optimum. Moreover, this objective function naturally gives rise to sparse models, in that, at the optimum, many parameters have value 0, corresponding to the elimination of features from the model. We discuss this approach in more detail in the next section.

20.7.4.4 L_1-Regularization for Structure Learning

Recall that the L_1-regularized likelihood is simply the instantiation of equation (20.29) to the case of an L_1-prior:

$$\text{score}_{L_1}(\boldsymbol{\theta} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \boldsymbol{\theta} \rangle : \mathcal{D}) - \|\boldsymbol{\theta}\|_1. \tag{20.31}$$

Somewhat surprisingly, the L_1-regularized likelihood can be optimized in a way that guarantees convergence to the globally optimal solution. To understand why, recall that the task of optimizing the L_1-regularized log-likelihood is a convex optimization problem that has no local optima.[1] Indeed, in theory, we can entirely avoid the combinatorial search component when

1. There might be multiple global optima due to redundancy in the parameter space, but these global optima all form a single convex region. Therefore, we use the term "the global optimum" to refer to any point in this optimal region.

using this objective. We can simply introduce all of the possible features into the model and optimize the resulting parameter vector $\boldsymbol{\theta}$ relative to our objective. The sparsifying effect of the L_1 penalty will drive some of the parameters to zero. The parameters that, at convergence, have zero values correspond to features that are absent from the log-linear model. In this approach, we are effectively making a structure selection decision as part of our parameter optimization procedure. Although appealing, this approach is not generally feasible. In most cases, the number of potential features we may consider for inclusion in the model is quite large. Including all of them in the model simultaneously gives rise to an intractable structure for the inference that we use as part of the computation of the gradient.

Therefore, even in the context of the L_1-regularized likelihood, we generally implement the optimization as a double-loop algorithm where we separately consider the structure and parameters. However, there are several benefits to the L_1-regularized objective:

- We do not need to consider feature deletion steps in our combinatorial search.
- We can consider feature introduction steps in any (reasonable) order, and yet achieve convergence to the global optimum.
- We have a simple and efficient test for determining convergence.
- We can prove a PAC-learnability generalization bound for this type of learning.

We now discuss each of these points.

For the purpose of this discussion, assume that we currently have a model over a set of features $\mathcal{F}$, and assume that $\boldsymbol{\theta}^l$ optimizes our L_1-regularized objective, subject to the constraint that θ_k^l can be nonzero only if $f_k \in \mathcal{F}$. At this convergence point, any feature deletion step cannot improve the score: Consider any $f_k \in \mathcal{F}$; the case where f_k is deleted is already in the class of models that was considered when we optimized the choice of $\boldsymbol{\theta}^l$ — it is simply the model where $\theta_k^l = 0$. Indeed, the algorithm already discards features whose parameter was zeroed by the continuous optimization procedure (line 7 of algorithm 20.1). If our current optimized model $\boldsymbol{\theta}^l$ has $\theta_k^l \neq 0$, it follows that setting θ_k to 0 is suboptimal, and so deleting f_k can only reduce the score. Thus, there is no value to considering discrete feature deletion steps: features that should be deleted will have their parameters set to 0 by the continuous optimization procedure. We note that this property also holds, in principle, for other smooth objectives, such as the likelihood or the L_2-regularized likelihood; the difference is that for those objectives, parameters will generally not be set to 0, whereas the L_1 objective does tend to induce sparsity.

The second benefit arises directly from the fact that optimizing the L_1-regularized objective is a convex optimization problem. In such problems, any sequence of steps that continues to improve the objective (when possible) is guaranteed to converge to the global optimum. The restriction imposed by the set $\mathcal{F}$ induces a coordinate ascent approach: at each step, we are optimizing only the features in $\mathcal{F}$, leaving at 0 those parameters θ_k for $f_k \notin \mathcal{F}$. As long as each step continues to improve the objective, we are making progress toward the global optimum. At each point in the search, we consider the steps that we can take. If some step leads to an improvement in the score, we can take that step and continue with our search. If none of the steps lead to an improvement in the score, we are guaranteed that we have reached convergence to the global optimum. **Thus, the decision on which operators to consider at each point in the algorithm (line 12 of algorithm 20.1) is not relevant to the convergence of the**

algorithm to the true global optimum: As long as we repeatedly consider each operator until convergence, we are guaranteed that the global optimum is reached regardless of the order in which the operators are applied.

While this guarantee is an important one, we should interpret it with care. First, when we add features to the model, the underlying network becomes more complex, raising the cost of inference. Because inference is executed many times during the algorithm, adding many irrelevant features, even if they were eventually eliminated, can greatly degrade the computational performance time of the algorithm. Even more problematic is the effect when we utilize approximate inference, as is often the case. As we discussed, for many approximate inference algorithms, not only the running time but also the accuracy tend to degrade as the network becomes more complex. Because inference is used to compute the gradient for the continuous optimization, the degradation of inference quality can lead to models that are suboptimal. Moreover, because the resulting model is generally also used to estimate the benefit of adding new features, any inaccuracy can propagate further, causing yet more suboptimal features to be introduced into the model. **Hence, especially when the quality of approximate inference is a concern, it is worthwhile to select with care the features to be introduced into the model rather than blithely relying on the "guaranteed" convergence to a global optimum.**

Another important issue to be addressed is the problem of determining convergence in line 14 of Greedy-MN-Structure-Search. In other words, how do we test that none of the search operators we currently have available can improve the score? A priori, this task appears daunting, since we certainly do not want to try all possible feature addition/deletion steps, reoptimize the parameters for each of them, and then check whether the score has improved. Fortunately, there is a much more tractable solution. Specifically, we can show the following proposition:

Proposition 20.5 *Let $\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D})$ denote the gradient of the likelihood relative to θ_k, evaluated at $\boldsymbol{\theta}^l$. Let β be the hyperparameter defining the L_1 prior. Let $\boldsymbol{\theta}^l$ be a parameter assignment for which the following conditions hold:*

- *For any k for which $\theta_k^l \neq 0$ we have that*

$$\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D}) - \frac{1}{\beta}\text{sign}(\theta_k^l) = 0.$$

- *For any k for which $\theta_k^l = 0$ we have that*

$$|\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D})| < \frac{1}{2\beta}.$$

Then $\boldsymbol{\theta}^l$ is a global optimum of the L_1-regularized log-likelihood function:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \frac{1}{\beta}\sum_{i=1}^{k}|\theta_i|.$$

Proof We provide a rough sketch of the proof. The first condition guarantees that the gradient relative to any parameter for which $\theta_k^l \neq 0$ is zero, and hence the objective function cannot be improved by changing its value. The second condition deals with parameters $\theta_k^l = 0$, for which

the gradient is discontinuous at the convergence point. However, consider a point $\boldsymbol{\theta}'$ in the nearby vicinity of $\boldsymbol{\theta}$, so that $\theta'_k \neq 0$. At $\boldsymbol{\theta}'$, the gradient of the function relative to θ_k is very close to

$$\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D}) - \frac{1}{\beta}\text{sign}(\theta'_k).$$

The value of this expression is positive if $\theta'_k < 0$ and negative if $\theta'_k > 0$. Thus, $\boldsymbol{\theta}^l$ is a local optimum of the L_1-regularized objective function. Because the function has only global optima, $\boldsymbol{\theta}^l$ must be a global optimum. ■

Thus, we can test convergence easily as a direct by-product of the continuous parameter optimization procedure executed at each step. We note that we still have to consider every feature that is not included in the model and compute the relevant gradient; but we do not have to go through the (much more expensive) process of trying to introduce the feature, optimizing the resulting model, and evaluating its score.

L-BFGS algorithm

So far, we have avoided the discussion of optimizing this objective. As we mentioned in section 20.3.1, a commonly used method for optimizing the likelihood is the *L-BFGS algorithm*, which uses gradient descent combined with line search (see appendix A.5.2). The problem with applying this method to the L_1-regularized likelihood is that the regularization term is not continuously differentiable: the gradient relative to any parameter θ_i changes at $\theta_i = 0$ from $+1$ to -1. Perhaps the simplest solution to this problem is to adjust the line-search procedure to avoid changing the sign of any parameter θ_i: If, during our line search, θ_i crosses from positive to negative (or vice versa), we simply fix it to be 0, and continue with the line search for the remaining parameters. Note that this decision corresponds to taking f_i out of the set of active features in this iteration. If the optimal parameter assignment has a nonzero value for θ_i, we are guaranteed that f_i will be introduced again in a later stage in the search, as we have discussed.

PAC-bound

Finally, as we mentioned, we can prove a useful theoretical guarantee for the results of L_1-regularized Markov network learning. Specifically, we can show the following *PAC-bound*:

Theorem 20.4

Let $\mathcal{X}$ be a set of variables such that $|Val(X_i)| \leq d$ for all i. Let P^* be a distribution, and $\delta, \epsilon, B > 0$. Let $\mathcal{F}$ be a set of all indicator features over all subsets of variables $\boldsymbol{X} \subset \mathcal{X}$ such that $|\boldsymbol{X}| \leq c$, and let

$$\Theta_{c,B} = \{\boldsymbol{\theta} \in \Theta[\mathcal{F}] \ : \ \|\boldsymbol{\theta}\|_1 \leq B\}$$

be all parameterizations of $\mathcal{F}$ whose L_1-norm is at most B. Let $\beta = \sqrt{c\ln(2nd/\delta)/(2M)}$. Let

$$\boldsymbol{\theta}^*_{c,B} = \arg\max_{\boldsymbol{\theta}\in\Theta_{c,B}} \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}})$$

be the best parameterization achievable within the class $\Theta_{c,B}$. For any data set $\mathcal{D}$, let

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}\in\Theta[\mathcal{F}]} \text{score}_{L_1}(\boldsymbol{\theta} \ : \ \mathcal{D}).$$

Then, for

$$M \geq \frac{2cB^2}{\epsilon^2}\ln\left(\frac{2nd}{\delta}\right),$$

with probability at least $1 - \delta$,

$$\boldsymbol{I}D(P^* \| P_{\hat{\boldsymbol{\theta}}}) \leq \boldsymbol{I}D(P^* \| P_{\boldsymbol{\theta}^{*c,B}}) + \epsilon.$$

In other words, this theorem states that, with high probability over data sets $\mathcal{D}$, the relative entropy to P^* achieved by the best L_1-regularized model is at most ϵ worse than the relative entropy achieved by the best model within the class of limited-degree Markov networks. This guarantee is achievable with a number of samples that is polynomial in ϵ, c, and B, and logarithmic in δ and d. The logarithmic dependence on n may feel promising, but we note that B is a sum of the absolute values of all network parameters; assuming we bound the magnitude of individual parameters, this terms grows linearly with the total number of network parameters. Thus, L_1-regularized learning provides us with a model that is close to optimal (within the class $\Theta_{c,B}$), using a polynomial number of samples.

20.7.5 Evaluating Changes to the Model

We now consider in more detail the candidate evaluation step that takes place in line 11 of Greedy-MN-Structure-Search. As we discussed, the standard way to reduce the cost of the candidate evaluation step is simply to avoid computing the exact score of each of the candidate successors, and rather to select among them using simpler heuristics. Many approximations are possible, ranging from ones that are very simple and heuristic to ones that are much more elaborate and provide certain guarantees.

Most simply, we can examine statistics of the data to determine features that may be worth including. For example, if two variables X_i and X_j are strongly correlated in the data, it may be worthwhile to consider introducing a factor over X_i, X_j (or a pairwise feature over one or more combinations of their values). The limitation of this approach is that it does not take into account the features that have already been introduced into the model and the extent to which they already explain the observed correlation.

grafting

A somewhat more refined approach, called *grafting*, estimates the benefit of introducing a feature f_k by compute the gradient of the likelihood relative to θ_k, evaluated at the current model. More precisely, assume that our current model is $(\mathcal{F}, \boldsymbol{\theta}^0)$. The *gradient heuristic* estimate to the delta-score (for score X) obtained by adding $f_k \notin \mathcal{F}$ is defined as:

gradient heuristic

$$\Delta_X^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}) = \frac{\partial}{\partial \theta_k} \text{score}_X(\boldsymbol{\theta} \ : \ \mathcal{D}), \tag{20.32}$$

evaluated at the current parameters $\boldsymbol{\theta}^0$.

The gradient heuristic does account for the parameters already selected; thus, for example, it can avoid introducing features that are not relevant given the parameters already introduced. Intuitively, features that have a high gradient can induce a significant immediate improvement in the score, and therefore they are good candidates for introduction into the model. Indeed, we are guaranteed that, if θ_k has a positive gradient, introducing f_k into $\mathcal{F}$ will result in some improvement to the score. The problem with this approach is that it does not attempt to evaluate how large this improvement can be. Perhaps we can increase θ_k only by a small amount before further changes stop improving the score.

An even more precise approximation is to evaluate a change to the model by computing the score obtained in a model where we keep all other parameters fixed. Consider a step where

we introduce or delete a single feature f_k in our model. We can obtain an approximation to the score by evaluating the change in score when we change only the parameter θ_k associated with f_k, keeping all other parameters unchanged. To formalize this idea, let $(\mathcal{F}, \boldsymbol{\theta}^0)$ be our current model, and consider changes involving f_k. We define the *gain heuristic* estimate to be the change in the score of the model for different values for θ_k, assuming the other parameters are kept fixed:

gain heuristic

$$\Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}) = \text{score}_X((\theta_k, \boldsymbol{\theta}^0_{-k}) \ : \ \mathcal{D}) - \text{score}_X(\boldsymbol{\theta}^0 \ : \ \mathcal{D}), \tag{20.33}$$

where $\boldsymbol{\theta}^0_{-k}$ is the vector of all parameters other than θ^0_k. As we discussed, due to the nondecomposability of the likelihood, when we change θ_k, the current assignment θ^0_{-k} to the other parameters is generally no longer optimal. However, it is still reasonable to use this function as a heuristic to rank different steps: Parameters that give rise to a larger improvement in the objective by themselves often also induce a larger improvement when other parameters are optimized. Indeed, changing those other parameters to optimize the score can only improve it further. Thus, the change in score that we obtain when we "freeze" the other parameters and change only θ_k is a lower bound on the change in the score.

The gain function can be used to provide a lower bound on the improvement in score derived from the deletion of a feature f_k currently in the model: we simply evaluate the gain function setting $\theta_k = 0$. We can also obtain a lower bound on the value of a step where we introduce into the model a new feature f_k (that is, one for which the current parameter $\theta^0_k = 0$). The improvement we can get if we freeze all parameters but one is clearly a lower bound on the improvement we can get if we optimize over all of the parameters. Thus, the value of $\Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D})$ is a lower bound on the improvement in the objective that can be gained by setting θ_k to its chosen value and optimizing all other parameters. To compute the best lower bound, we must maximize the function relative to different possible values of θ_k, giving us the score of the best possible model when all parameters other than θ_k are frozen. In particular, we can define:

$$\text{Gain}_X(\boldsymbol{\theta}^0 \ : \ f_k, \mathcal{D}) = \max_{\theta_k} \Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}).$$

This is a lower bound on the change in the objective obtained from introducing a $f_k \notin \mathcal{F}$.

In principle, lower bounds are more useful than simple approximations. If our lower bound of the candidate's score is higher than that of our current best model, then we definitely want to evaluate that candidate; this will result in a better current candidate and allow us to prune additional candidates and focus on the ones that seem more promising for evaluation. Upper bounds are useful as well. If we have a candidate model and obtain an upper bound on its score, then we can remove it from consideration once we evaluate another candidate with higher score; thus, upper bounds help us prune models for which we would never want to evaluate the true score. In practice, however, fully evaluating the score for all but a tiny handful of candidate structures is usually too expensive a proposition. Thus, the gain is generally used simply as an approximation rather than a lower bound.

How do we evaluate the gain function efficiently, or find its optimal value? The gain function is a univariate function of θ_k, which is a projection of the score function onto this single dimension. Importantly, all of our scoring functions — including any of the penalized likelihood functions we described — are concave (for a given set of active features). The projection

of a concave function onto a single dimension is also concave, so that this single-parameter delta-score is also concave and therefore has a global optimum.

Nevertheless, given the complexity of the likelihood function, it is not clear how this global optimum can be efficiently found. We now show how the difference between two log-likelihood terms can be considerably simplified, even allowing a closed-form solution in certain important cases. Recall from equation (20.3) that

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_k \theta_k \boldsymbol{E}_{\mathcal{D}}[f_k] - \ln Z(\boldsymbol{\theta}).$$

Because parameters other than θ_k are the same in the two models, we have that:

$$\frac{1}{M}[\ell((\theta_k, \boldsymbol{\theta}^0_{-k}) : \mathcal{D}) - \ell(\boldsymbol{\theta}^0 : \mathcal{D})] = (\theta_k - \theta^0_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \left[\ln Z(\theta_k, \boldsymbol{\theta}^0_{-k}) - \ln Z(\boldsymbol{\theta}^0)\right].$$

The first term is a linear function in θ_k, whose coefficient is the empirical expectation of f_k in the data. For the second term, we have:

$$\begin{aligned}
\ln \frac{Z(\theta_k, \boldsymbol{\theta}^0_{-k})}{Z(\boldsymbol{\theta}^0)} &= \ln \left[\frac{1}{Z(\boldsymbol{\theta}^0)} \sum_{\xi} \exp\left\{ \sum_j \theta^0_j f_j(\xi) + (\theta_k - \theta^0_k) f_k(\xi) \right\} \right] \\
&= \ln \sum_{\xi} \frac{\tilde{P}_{\boldsymbol{\theta}^0}(\xi)}{Z(\boldsymbol{\theta}^0)} \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\xi)\right\}\right] \\
&= \ln \boldsymbol{E}_{\boldsymbol{\theta}^0}\left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right].
\end{aligned}$$

Thus, the difference of these two log-partition functions can be rewritten as a log-expectation relative to our original distribution. We can convert this expression into a univariate function of θ_k by computing (via inference in our current model $\boldsymbol{\theta}^0$) the marginal distribution over the variables $\boldsymbol{d}_k$. Altogether, we obtain that:

$$\begin{aligned}
&\frac{1}{M}[\ell((\theta_k, \boldsymbol{\theta}^0_{-k}) : \mathcal{D}) - \ell(\boldsymbol{\theta}^0 : \mathcal{D})] = \\
&\quad (\theta_k - \theta^0_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \ln \sum_{\boldsymbol{d}_k} P_{\boldsymbol{\theta}^0}(\boldsymbol{d}_k) \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right].
\end{aligned} \tag{20.34}$$

We can incorporate this simplified form into equation (20.33) for any penalized-likelihood scoring function. We can now easily provide our lower-bound estimates for feature deletions. For introducing a feature f_k, the optimal lower bound can be computed by optimizing the univariate function defined by equation (20.33) over the parameter θ_k. Because this function is concave, it can be optimized using a variety of univariate numerical optimization algorithms. For example, to compute the lower bound for an L_2-regularized likelihood, we would compute:

$$\max_{\theta_k} \left\{ \theta_k \boldsymbol{E}_{\mathcal{D}}[f_k] - \ln \sum_{\boldsymbol{d}_k} P_{\boldsymbol{\theta}^0}(\boldsymbol{d}_k) \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right] - \frac{\theta_k^2}{2\sigma^2} \right\}.$$

However, in certain special cases, we can actually provide a closed-form solution for this optimization problem. We note that this derivation applies only in restricted cases: only in the case of generative training (that is, not for CRFs); only for the likelihood or L_1-penalized objective; and only for binary-valued features.

Proposition 20.6 *Let f_k be a binary-valued feature and let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model. Let $\hat{p}_k = \boldsymbol{E}_{\mathcal{D}}[f_k]$ be the empirical probability of f_k in $\mathcal{D}$, and $p_k^0 = P_{\boldsymbol{\theta}}(f_k)$ be its probability relative to the current model. Then:*

$$\max_{\theta_k} \left[\text{score}_L((\theta_k, \boldsymbol{\theta}^0_{-k}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{M}(\hat{p}_k \| p_k^0),$$

where the KL-divergence is the relative entropy between the two Bernoulli distributions parameterized by $\hat{p}_k$ and p_k^0 respectively.

The proof is left as an exercise (exercise 20.16).

To see why this result is intuitive, recall that when we maximize the likelihood relative to some log-linear model, we obtain a model where the expected counts match the empirical counts. In the case of a binary-valued feature, in the optimized model we have that the final probability of f_k would be the same as the empirical probability $\hat{p}_k$. Thus, it is reasonable that the amount of improvement we obtain from this optimization is a function of the discrepancy between the empirical probability of the feature and its probability given the current model. The bigger the discrepancy, the bigger the improvement in the likelihood.

A similar analysis applies when we consider several binary-valued features $f_1, \ldots, f_k$, as long as they are mutually exclusive and exhaustive; that is, as long as there are no assignments for which both $f_{i'}(\xi) = 1$ and $f_j(\xi) = 1$. In particular, we can show the following:

Proposition 20.7 *Let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model, and consider introducing into the model a complete factor ϕ over scope $\boldsymbol{d}$, parameterized with $\boldsymbol{\theta_d}$ that correspond to the different assignments to $\boldsymbol{d}$. Then*

$$\max_{\boldsymbol{\theta_d}} \left[\text{score}_L((\theta_{\boldsymbol{d}}, \boldsymbol{\theta}^0_{-\boldsymbol{d}}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{M}(\hat{P}(\boldsymbol{d}) \| P_{\boldsymbol{\theta}}(\boldsymbol{d})).$$

The proof is left as an exercise (exercise 20.17).

Although the derivations here were performed for the likelihood function, a similar closed-form solution, in the same class of cases, can also be performed for the L_1-regularized likelihood (see exercise 20.18), but not for the L_2-regularized likelihood. Intuitively, the penalty in the L_1-regularized likelihood is a linear function in each θ_k, and therefore it does not complicate the form of equation (20.34), which already contains such a term. However, the L_2 penalty is quadratic, and introducing a quadratic term into the function prevents an analytic solution.

One issue that we did not address is the task of computing the expressions in equation (20.34), or even the closed-form expressions in proposition 20.6 and proposition 20.7. All of these expressions involve expectations over the scope $\boldsymbol{D}_k$ of f_k, where f_k is the feature that we want to eliminate from or introduce into the model. Let us consider first the case where f_k is already in the model. In this case, if we use a *belief propagation* algorithm (whether a clique tree or a loopy cluster graph), the family preservation property guarantees that the feature's scope $\boldsymbol{D}_k$ is necessarily a subset of some cluster in our inference data structure. Thus, we can easily compute the necessary expectations. However, for a feature not currently in the model, we would not generally expect its scope to be included in any cluster. If not, we must somehow compute expectations of sets of variables that are not together in the same cluster. In the case of clique trees, we can use the out-of-clique inference methods described in section 10.3.3.2. For the case of loopy cluster graphs, this problem is more challenging (see exercise 11.22).

20.8 Summary

In this chapter, we discussed the problem of learning undirected graphical models from data. The key challenge in learning these models is that the global partition function couples the parameters, with significant consequences: There is no closed-form solution for the optimal parameters; moreover, we can no longer optimize each of the parameters independently of the others. Thus, even simple maximum-likelihood parameter estimation is no longer trivial. For the same reason, full Bayesian estimation is computationally intractable, and even approximations are expensive and not often used in practice.

Following these pieces of bad news, there are some good ones: the likelihood function is concave, and hence it has no local optima and can be optimized using efficient gradient-based methods over the space of possible parameterizations. We can also extend this method to MAP estimation when we are given a prior over the parameters, which allows us to reduce the overfitting to which maximum likelihood is prone.

The gradient of the likelihood at a point $\boldsymbol{\theta}$ has a particularly compelling form: the gradient relative to the parameter θ_i corresponding to the feature f_i is the difference between the empirical expectation of f_i in the data and its expectation relative to the distribution $P_{\boldsymbol{\theta}}$. While very intuitive and simple in principle, the form of the gradient immediately gives rise to some bad news: to compute the gradient at the point $\boldsymbol{\theta}$, we need to run inference over the model $P_{\boldsymbol{\theta}}$, a costly procedure to execute at every gradient step.

This complexity motivates the use of myriad alternative approaches: ones involving the use of approximate inference for computing the gradient; and ones that utilize a different objective than the likelihood. Methods in the first class included using message passing algorithms such as belief propagation, and methods based on sampling. We also showed that many of the methods that use approximate inference for optimizing the likelihood can be reformulated as exactly optimizing an approximate objective. This perspective can offer significant insight. For example, we showed that learning with belief propagation can be reformulated as optimizing a joint objective that involves both inference and learning; this alternative formulation is more general and allows the use of alternative optimization methods that are more stable and convergent than using BP to estimate the gradient.

Methods that use an approximate objective include pseudolikelihood, contrastive divergence, and maximum-margin (which is specifically geared for discriminative training of conditional models). Importantly, both likelihood and these objectives can be viewed as trying to increase the distance between the log-probability of assignments in our data and those of some set other assignments. This "contrastive" view provides a different view of these objectives, and it suggests that they are only representatives of a much more general class of approximations.

The same analysis that we performed for optimizing the likelihood can also be extended to other cases. In particular, we showed a very similar derivation for conditional training, where the objective is to maximize the likelihood of a set of target variables $\boldsymbol{Y}$ given some set of observed feature variables $\boldsymbol{X}$.

We also showed that similar approaches can be applied to learning with missing data. Here, the optimization task is no longer convex, but the gradient has a very similar form and can be optimized using the same gradient-ascent methods. However, as in the case of Bayesian network learning with missing data, the likelihood function is generally multimodal, and so the gradient ascent algorithm can get stuck in local optima. Thus, we may need to resort to techniques such

as data perturbation or random restarts.

We also discussed the problem of structure learning of undirected models. Here again, we can use both constraint-based and score-based methods. Owing to the difficulties arising from the form of the likelihood function, full Bayesian scoring, where we score a model by integrating over all of the parameters, is intractable, and even approximations are generally impractical. Thus, we generally use a simpler scoring function, which combines a likelihood term (measuring fit to data) with some penalty term. We then search over some space of structures for ones that optimize this objective. For most objectives, the resulting optimization problem is combinatorial with multiple local optima, so that we must resort to heuristic search. One notable exception is the use of an L_1-regularized likelihood, where the penalty on the absolute value of the parameters tends to drive many of the parameters to zero, and hence often results in sparse models. This objective allows the structure learning task to be formulated as a convex optimization problem over the space of parameters, allowing the optimization to be performed efficiently and with guaranteed convergence to a global optimum. Of course, even here inference is still an unavoidable component in the inner loop of the learning algorithm, with all of the ensuing difficulties.

As we mentioned, the case of discriminative training is a setting where undirected models are particularly suited, and are very commonly used. However, it is important to carefully weigh the trade-offs of generative versus discriminative training. As we discussed, there are significant differences in the computational cost of the different forms of training, and the trade-off can go either way. More importantly, as we discussed in section 16.3.2, **generative models incorporate a higher bias by making assumptions — ones that are often only approximately correct — about the underlying distribution. Discriminative models make fewer assumptions, and therefore tend to require more data to train; generative models, due to the stronger bias, often perform better in the sparse-data regime. But incorrect modeling assumptions also hurt performance; therefore, as the amount of training data grows, the discriminative model, which makes fewer assumptions, often performs better. This difference between the two classes of models is particularly significant when we have complex features whose correlations are hard to model.** However, it is important to remember that models trained discriminatively to predict $\boldsymbol{Y}$ given $\boldsymbol{X}$ will perform well primarily in this setting, and even slight changes may lead to a degradation in performance. For example, a model for predicting $P(Y \mid X_1, X_2)$ would not be useful for predicting $P(Y \mid X_1)$ in situations where X_2 is not observed. In general, discriminative models are much less flexible in their ability to handle missing data.

We focused most of our discussion of learning on the problem of learning log-linear models defined in terms of a set of features. Log-linear models are a finer-grained representation than a Markov network structure or a set of factors. Thus, they can make better trade-offs between model complexity and fit to data. However, sparse log-linear models (with few features) do not directly correspond to sparse Markov network structures, so that we might easily end up learning a model that does not lend itself to tractable inference. It would be useful to consider the development of Markov network structure learning algorithms that more easily support efficient inference. Indeed, some work has been done on learning Markov networks of bounded tree-width, but networks of low tree-width are often poor approximations to the target distribution. Thus, it would be interesting to explore alternative approaches that aim at structures that support approximate inference.

This chapter is structured as a core idea with a set of distinct extensions that build on it: The core idea is the use of the likelihood function and the analysis of its properties. The extensions include conditional likelihood, learning with missing data, the use of parameter priors, approximate inference and/or approximate objectives, and even structure learning. In many cases, these extensions are orthogonal, and we can easily combine them in various useful ways. For example, we can use parameter priors with conditional likelihood or in the case of missing data; we can also use them with approximate methods such as pseudolikelihood, contrastive divergence or in the objective of equation (20.15). Perhaps more surprising is that we can easily perform structure learning with missing data by adding an L_1-regularization term to the likelihood function of equation (20.8) and then using the same ideas as in section 20.7.4.4. In other cases, the combination of the different extensions is more involved. For example, as we discussed, structure learning requires that we be able to evaluate the expected counts for variables that are not in the same family; this task is not so easy if we use an approximate algorithm such as belief propagation. As another example, it is not immediately obvious how we can extend the pseudolikelihood objective to deal with missing data. These combinations provide useful directions for future work.

20.9 Relevant Literature

iterative proportional scaling

iterative proportional fitting

Log-linear models and contingency tables have been used pervasively in a variety of communities, and so key ideas have often been discovered multiple times, making a complete history too long to include. Early attempts for learning log-linear models were based on the *iterative proportional scaling* algorithm and its extension, *iterative proportional fitting*. These methods were first developed for contingency tables by Deming and Stephan (1940) and applied to log-linear models by Darroch and Ratcliff (1972). The convex duality between the maximum likelihood and maximum entropy problems appears to have been proved independently in several papers in diverse communities, including (at least) Ben-Tal and Charnes (1979); Dykstra and Lemke (1988); Berger, Della-Pietra, and Della-Pietra (1996). It appears that the first application of gradient algorithms to maximum likelihood estimation in graphical models is due to Ackley, Hinton, and Sejnowski (1985) in the context of Boltzmann machines. The importance of the method used to optimize the likelihood was highlighted in the comparative study of Minka (2001a); this study focused on learning for logistic regression, but many of the conclusions hold more broadly. Since then, several better methods have been developed for optimizing likelihood. Successful methods include conjugate gradient, L-BFGS (Liu and Nocedal 1989), and stochastic meta-descent (Vishwanathan et al. 2006).

Conditional random fields were first proposed by Lafferty et al. (2001). They have since been applied in a broad range of applications, such as labeling multiple webpage on a website (Taskar et al. 2002), image segmentation (Shental et al. 2003), or information extraction from text (Sutton and McCallum 2005). The application to protein-structure prediction in box 20.B is due to Yanover et al. (2007).

The use of approximate inference in learning is an inevitable consequence of the intractability of the inference problem. Several papers have studied the interaction between belief propagation and Markov network learning. Teh and Welling (2001) and Wainwright et al. (2003b) present methods for certain special cases; in particular, Wainwright, Jaakkola, and Willsky (2003b) derive

the pseudo-moment matching argument. Inspired by the moment-matching behavior of learning with belief propagation, Sutton and McCallum (2005); Sutton and Minka (2006) define the piecewise training objective that directly performs moment matching on all network potentials. Wainwright (2006) provides a strong argument, both theoretical and empirical, for using the same approximate inference method in training as will be used in performing the prediction using the learned model. Indeed, he shows that, if an approximate method is used for inference, then we get better performance guarantees if we use that same method to train the model than if we train a model using exact inference. He also shows that it is detrimental to use an unstable inference algorithm (such as sum-product BP) in the inner loop of the learning algorithm. Ganapathi et al. (2008) define the unified CAMEL formulation that encompasses learning and inference in a single joint objective, allowing the nonconvexity of the BP objective to be taken out of the inner loop of learning.

Although maximum (conditional) likelihood is the most commonly used objective for learning Markov networks, several other objectives have been proposed. The earliest is pseudolikelihood, proposed by Besag (1977b), of which several extensions have been proposed (Huang and Ogata 2002; McCallum et al. 2006). The asymptotic consistency of both the likelihood and the pseudolikelihood objectives is shown by Gidas (1988). The statistical efficiency (convergence as a function of the number of samples) of the pseudolikelihood estimator has also been analyzed (for example, (Besag 1977a; Geyer and Thompson 1992; Guyon and Künsch 1992; Liang and Jordan 2008)).

The use of margin-based estimation methods for probabilistic models was first proposed by Collins (2002) in the context of parsing and sequence modeling, building on the voted-perceptron algorithm (Freund and Schapire 1998). The methods described in this chapter build on a class of large-margin methods called *support vector machines* (Shawe-Taylor and Cristianini 2000; Hastie et al. 2001; Bishop 2006), which have the important benefit of allowing a large or even infinite feature space to be used and trained very efficiently. This formulation was first proposed by Altun, Tsochantaridis, and Hofmann (2003); Taskar, Guestrin, and Koller (2003), who proposed two different approaches for addressing the exponential number of constraints. Altun et al. use a constraint-generation scheme, which was subsequently proven to require at most a polynomial number of steps (Tsochantaridis et al. 2004). Taskar et al. use a closed-form polynomial-size reformulation of the optimization problem that uses a clique tree-like data structure. Taskar, Chatalbashev, and Koller (2004) also show that this formulation also allows tractable training for networks where conditional probability products are intractable, but the MAP assignment can be found efficiently. The contrastive divergence approach was introduced by Hinton (2002); Teh, Welling, Osindero, and Hinton (2003), and was shown to work well in practice in various studies (for example, (Carreira-Perpignan and Hinton 2005)). This work forms part of a larger trend of training using a range of alternative, often contrastive, objectives. LeCun et al. (2007) provide an excellent overview of this area.

support vector machine

Much discussion has taken place in the machine learning community on the relative merits of discriminative versus generative training. Some insightful papers of particular relevance to graphical models include the work of Minka (2005) and LeCun et al. (2007). Also of interest are the theoretical analyses of Ng and Jordan (2002) and Liang and Jordan (2008) that discuss the statistical efficiency of discriminative versus generative training, and provide theoretical support for the empirical observation that generative models, even if not consistent with the true underlying distribution, often work better in the sparse data case, but discriminative models

tend to work better as the amount of data grows.

The work of learning Markov networks with hidden variables goes back to the seminal paper of Ackley, Hinton, and Sejnowski (1985), who used gradient ascent to train Boltzmann machines with hidden variables. This line of work, largely dormant for many years, has seen a resurgence in the work on *deep belief networks* (Hinton et al. 2006; Hinton and Salakhutdinov 2006), a training regime for a multilayer restricted Boltzmann machine that iteratively tries to learn deeper and deeper hidden structure in the data.

deep belief networks

Parameter priors and regularization methods for log-linear models originate in statistics, where they have long been applied to a range of statistical models. Many of the techniques described here were first developed for traditional statistical models such as linear or logistic regression, and then extended to the general case of Markov networks and CRFs. See Hastie et al. (2001) for some background on this extensive literature.

The problem of learning the structure of Markov networks has not received as much attention as the task of Bayesian network structure learning. One line of work has focused on the problem of learning a Markov network of bounded tree-width, so as to allow tractable inference. The work of Chow and Liu (1968) shows that the maximum-likelihood tree-structured network can be found in quadratic time.

A tree is a network of tree-width 1. Thus, the obvious generalization of is to learning the class of Markov networks whose tree-width is at most k. Unfortunately, there is a sharp threshold phenomenon, since Srebro (2001) proves that for any tree-width k greater than 1, finding the maximum likelihood tree-width-k network is $\mathcal{NP}$-hard. Interestingly, Narasimhan and Bilmes (2004) provide a constraint-based algorithm for PAC-learning Markov networks of tree-width at most k: Their algorithm is guaranteed to find, with probability $1 - \delta$, a network whose relative entropy is within ϵ of optimal, in polynomial time, and using a polynomial number of samples. Importantly, their result does not contradict the hardness result of Srebro, since their analysis applies only in the consistent case, where the data is derived from a k-width network. This discrepancy highlights again the significant difference between learnability in the consistent and the inconsistent case. Several search-based heuristic algorithms for learning models with small tree-width have been proposed (Bach and Jordan 2001; Deshpande et al. 2001); so far, none of these algorithms have been widely adopted, perhaps because of the limited usefulness of bounded tree-width networks.

Abbeel, Koller, and Ng (2006) provide a different PAC-learnability result in the consistent case, for networks of bounded connectivity. Their constraint-based algorithm is guaranteed to learn, with high probability, a network $\tilde{\mathcal{M}}$ whose (symmetric) relative entropy to the true distribution ($\boldsymbol{D}(\tilde{P}\|P^*) + \boldsymbol{D}(P^*\|\tilde{P})$) is at most ϵ. The complexity, both in time and in number of samples, grows exponentially in the maximum number of assignments to any local neighborhood (a factor and its Markov blanket). This result is somewhat surprising, since it shows that the class of low-connectivity Markov networks (such as grids) is PAC-learnable, even though inference (including computing the partition function) can be intractable.

Of highest impact has been the work on using local search to optimize a (regularized) likelihood. This line of work originated with the seminal paper of Della Pietra, Della Pietra, and Lafferty (1997), who defined the single-feature gain, and proposed the gain as an effective heuristic for feature selection in learning Markov network structure. McCallum (2003) describes some heuristic approximations that allow this heuristic to be applied to CRFs. The use of L_1-regularization for feature selection originates from the Lasso model proposed for linear re-

gression by Tibshirani (1996). It was first proposed for logistic regression by Perkins et al. (2003); Goodman (2004). Perkins et al. also suggested the gradient heuristic for feature selection and the L_1-based stopping rule. L_1-regularized priors were first proposed for learning log-linear distributions by Riezler and Vasserman (2004); Dudík, Phillips, and Schapire (2004). The use of L_1-regularized objectives for learning the structure of general Markov networks was proposed by Lee et al. (2006). Building on the results of Dudík et al., Lee et al. also showed that the number of samples required to achieve close-to-optimal relative entropy (within the target class) grows only polynomially in the size of the network. Importantly, unlike the PAC-learnability results mentioned earlier, this result also holds in the inconsistent case.

Pseudolikelihood has also been used as a criterion for model selection. Ji and Seymour (1996) define a pseudolikelihood-based objective and show that it is asymptotically consistent, in that the probability of selecting an incorrect model goes to zero as the number of training examples goes to infinity. However, they did not provide a tractable algorithm for finding the highest scoring model in the superexponentially large set of structures. Wainwright et al. (2006) suggested the use of an L_1-regularized pseudolikelihood for model selection, and also proved a theorem that provides guarantees on the near-optimality of the learned model, using a polynomial number of samples. Like the result of Lee et al. (2006), this result applies also in the inconsistent case.

This chapter has largely omitted discussion of the Bayesian learning approach for Markov networks, for both parameter estimation and structure learning. Although an exact approach is computationally intractable, some interesting work has been done on approximate methods. Some of this work uses MCMC methods to sample from the parameter posterior. Murray and Ghahramani (2004) propose and study several diverse methods; owing to the intractability of the posterior, all of these methods are approximate, in that their stationary distribution is only an approximation to the desired parameter posterior. Of these, the most successful methods appear to be a method based on Langevin sampling with approximate gradients given by contrastive divergence, and a method where the acceptance probability is approximated by replacing the log partition function with the Bethe free energy. Two more restricted methods (Møller et al. 2006; Murray et al. 2006) use an approach called "perfect sampling" to avoid the need for estimating the partition function; these methods are elegant but of limited applicability. Other approaches approximate the parameter posterior by a Gaussian distribution, using either expectation propagation (Qi et al. 2005) or a combination of a Bethe and a Laplace approximation (Welling and Parise 2006a). The latter approach was also used to approximate the Bayesian score in order to perform structure learning (Welling and Parise 2006b). Because of the fundamental intractability of the problem, all of these methods are somewhat complex and computationally expensive, and they have therefore not yet made their way into practical applications.

20.10 Exercises

Exercise 20.1

Consider the network of figure 20.2, where we assume that some of the factors share parameters. Let $\boldsymbol{\theta}_i^y$ be the parameter vector associated with all of the features whose scope is Y_i, Y_{i+1}. Let $\boldsymbol{\theta}_{i,j}^{xy}$ be the parameter vector associated with all of the features whose scope is Y_i, X_j.

a. Assume that, for all i, i', $\boldsymbol{\theta}_i^y = \boldsymbol{\theta}_{i'}^y$, and that for all i, i' and j, $\boldsymbol{\theta}_{i,j}^{xy} = \boldsymbol{\theta}_{i',j}^{xy}$. Derive the gradient update for this model.

b. Now (without the previous assumptions) assume for all i and j, j', $\boldsymbol{\theta}^{xy}_{i,j} = \boldsymbol{\theta}^{xy}_{i,j'}$. Derive the gradient update for this model.

Exercise 20.2

relational Markov network

In this exercise, we show how to learn Markov networks with shared parameters, such as a *relational Markov network* (RMN).

a. Consider the log-linear model of example 6.18, where we assume that the *Study-Pair* relationship is determined in the relational skeleton. Thus, we have a single template feature, with a single weight, which is applied to all study pairs. Derive the likelihood function for this model, and the gradient.
b. Now provide a formula for the likelihood function and the gradient for a general RMN, as in definition 6.14.

Exercise 20.3

Assume that our data are generated by a log-linear model $P_{\boldsymbol{\theta}^*}$ that is of the form of equation (20.1). Show that, as the number of data instances M goes to infinity, with probability that approaches 1, $\boldsymbol{\theta}^*$ is a global optimum of the likelihood objective of equation (20.3). (Hint: Use the characterization of theorem 20.1.)

Exercise 20.4

Use the techniques described in this chapter to provide a method for performing maximum likelihood estimation for a CPD whose parameterization is a generalized linear model, as in definition 5.10.

Exercise 20.5⋆

Show using Lagrange multipliers and the definitions of appendix A.5.4 that the problem of maximizing $\boldsymbol{H}_Q(\mathcal{X})$ subject to equation (20.10) is dual to the problem of maximizing the log likelihood $\max \ell(\boldsymbol{\theta} : \mathcal{D})$.

Exercise 20.6⋆

In this problem, we will show an analogue to theorem 20.2 for the problems of maximizing conditional likelihood and maximizing conditional entropy.

Consider a data set $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^M$ as in section 20.3.2, and define the following conditional entropy maximization problem:

Maximum-Conditional-Entropy:

Find	$Q(\boldsymbol{Y} \mid \boldsymbol{X})$
maximizing	$\sum_{m=1}^M \boldsymbol{H}_Q(\boldsymbol{Y} \mid \boldsymbol{x}[m])$
subject to	

$$\sum_{m=1}^M \boldsymbol{E}_{Q(\boldsymbol{Y} \mid \boldsymbol{x}[m])}[f_k] = \sum_{m=1}^M f_k(\boldsymbol{y}[m], \boldsymbol{x}[m]) \quad i = 1, \ldots, k. \tag{20.35}$$

Show that $Q^*(\boldsymbol{Y} \mid \boldsymbol{X})$ optimizes this objective if and only if $Q^* = P_{\hat{\boldsymbol{\theta}}}$ where $P_{\hat{\boldsymbol{\theta}}}$ maximizes $\ell_{\boldsymbol{Y} \mid \boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D})$ as in equation (20.6).

Exercise 20.7⋆

iterative proportional scaling

One of the earliest approaches for finding maximum likelihood parameters is called *iterative proportional scaling* (IPS). The idea is essentially to use coordinate ascent to improve the match between the empirical feature counts and the expected feature counts. In other words, we change θ_k so as to make $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_k]$ closer to $\boldsymbol{E}_{\mathcal{D}}[f_k]$. Because our model is multiplicative, it seems natural to multiply the weight of instances where $f_k(\xi) = 1$ by the ratio between the two expectations. This intuition leads to the following update rule:

$$\theta'_k \leftarrow \theta_k + \ln \frac{\boldsymbol{E}_{\mathcal{D}}[f_k]}{\boldsymbol{E}_{P_\theta}[f_k]}. \tag{20.36}$$

The IPS algorithm iterates over the different parameters and updates each of them in turn, using this update rule.

Somewhat surprisingly, one can show that each iteration increases the likelihood until it reaches a maximum point. Because the likelihood function is concave, there is a single maximum, and the algorithm is guaranteed to find it.

Theorem 20.5

Let $\boldsymbol{\theta}$ be a parameter vector, and $\boldsymbol{\theta}'$ the vector that results from it after an application of equation (20.36). Then $\ell(\boldsymbol{\theta}' : \mathcal{D}) \geq \ell(\boldsymbol{\theta} : \mathcal{D})$ with equality if only if $\frac{\partial}{\partial \theta_k} \ell(\boldsymbol{\theta}^t : \mathcal{D}) = 0$.

In this exercise, you will prove this theorem for the special case where f_k is binary-valued. More precisely, let $\Delta(\theta_k)$ denote the change in likelihood obtained from modifying a single parameter θ_k, keeping the others fixed. This expression was computed in equation (20.34). You will now show that the IPS update step for θ_k maximizes a lower bound on this single parameter gain.

Define

$$\begin{aligned}\tilde{\Delta}(\theta_k) &= (\theta'_k - \theta_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \frac{Z(\boldsymbol{\theta}')}{Z(\boldsymbol{\theta})} + 1 \\ &= (\theta'_k - \theta_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \boldsymbol{E}_{P_{\boldsymbol{\theta}}}[1 - f_k] - e^{\theta'_k - \theta_k}\boldsymbol{E}_{P_{\boldsymbol{\theta}'}}[f_k] + 1.\end{aligned}$$

a. Show that $\Delta(\theta'_k) \geq \tilde{\Delta}(\theta'_k)$. (Hint: use the bound $\ln(x) \leq x - 1$.)

b. Show that $\theta_k + \ln \frac{\boldsymbol{E}_{\mathcal{D}}[f_k]}{\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_k]} = \arg\max_{\theta'_k} \tilde{\Delta}(\theta'_k)$.

c. Use these two facts to conclude that IPS steps are monotonically nondecreasing in the likelihood, and that convergence is achieved only when the log-likelihood is maximized.

d. This result shows that we can view IPS as performing coordinatewise ascent on the likelihood surface. At each iteration we make progress along on dimension (one parameter) while freezing the others. Why is coordinate ascent a wasteful procedure in the context of optimizing the likelihood?

Exercise 20.8

hyperbolic prior

Consider the following *hyperbolic prior* for parameters in log-linear models.

$$P(\theta) = \frac{1}{(e^{\theta} + e^{-\theta})/2}.$$

a. Derive a gradient-based update rule for this parameter prior.

b. Qualitatively describe the expected behavior of this parameter prior, and compare it to those of the L_2 or L_1 priors discussed in section 20.4. In particular, would you expect this prior to induce sparsity?

Exercise 20.9

piecewise training

We now consider an alternative local training method for Markov networks, known as *piecewise training*. For simplicity, we focus on Markov networks parameterized via full table factors. Thus, we have a set of factors $\phi_c(\boldsymbol{X}_c)$, where $\boldsymbol{X}_c$ is the scope of factor c, and $\phi_c(\boldsymbol{x}_c^j) = \exp(\theta_{cj})$. For a particular parameter assignment $\boldsymbol{\theta}$, we define $Z_c(\boldsymbol{\theta})$ to be the local partition function for this factor in isolation:

$$Z_c(\boldsymbol{\theta}_c) = \sum_{\boldsymbol{x}_c} \phi_c(\boldsymbol{x}_c),$$

where $\boldsymbol{\theta}_c$ is the parameter vector associated with the factor $\phi_c(\boldsymbol{X}_c)$. We can approximate the global partition function in the log-likelihood objective of equation (20.3) as a product of the local partition functions, replacing $Z(\theta)$ with $\prod_c Z_c(\boldsymbol{\theta}_c)$.

a. Write down the form of the resulting objective, simplify it, and derive the assignment of parameters that optimizes it.
b. Compare the result of this optimization to the result of the pseudo-moment matching approach described in section 20.5.1.

Exercise 20.10⋆

CAMEL

In this exercise, we analyze the following simplification of the *CAMEL* optimization problem of equation (20.15):

Simple-Approx-Maximum-Entropy:

Find $\boldsymbol{Q}$
maximizing $\sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i)$
subject to

$$\begin{aligned} \boldsymbol{E}_{\beta_i}[f_i] &= \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k \\ \sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) &= 1 \quad i = 1, \ldots, k \\ \boldsymbol{Q} &\geq 0 \end{aligned}$$

Here, we approximate both the objective and the constraints. The objective is approximated by the removal of all of the negative entropy terms for the sepsets. The constraints are relaxed by removing the requirement that the potentials in $\boldsymbol{Q}$ be locally consistent (sum-calibrated) — we now require only that they be legal probability distributions.

Show that this optimization problem is the Lagrangian dual of the piecewise training objective in exercise 20.9.

Exercise 20.11⋆

multiconditional training

Consider a setting, as in section 20.3.2, where we have two sets of variables $\boldsymbol{Y}$ and $\boldsymbol{X}$. *Multiconditional training* provides a spectrum between pure generative and pure discriminative training by maximizing the following objective:

$$\alpha \ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) + (1 - \alpha)\ell_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{\theta} : \mathcal{D}). \tag{20.37}$$

Consider the model structure shown in figure 20.2, and a partially labeled data set $\mathcal{D}$, where in each instance m we observe all of the feature variables $\boldsymbol{x}[m]$, but only the target variables in $\boldsymbol{O}[m]$.

Write down the objective of equation (20.37) for this case and compute its derivative.

Exercise 20.12⋆

Consider the problem of maximizing the approximate log-likelihood shown in equation (20.16).

a. Derive the gradient of the approximate likelihood, and show that it is equivalent to utilizing an importance sampling estimator directly to approximate the expected counts in the gradient of equation (20.4).
b. Characterize properties of the maximum point (when the gradient is 0). Is such a maximum always attainable? Prove or suggest a counterexample.

Exercise 20.13⋆⋆

One approach to providing a lower bound to the log-likelihood is by upper-bounding the partition function. Assume that we can decompose our model as a convex combination of (hopefully) simpler models, each with a weight α_k and a set of parameters ψ^k. We define these submodels as follows: $\psi^k(\boldsymbol{\theta}) = \boldsymbol{w}^k \bullet \boldsymbol{\theta}$, where we require that, for any feature i,

$$\sum_k \alpha_k w_i^k = 1. \tag{20.38}$$

a. Under this assumption, prove that

$$\ln Z(\boldsymbol{\theta}) \leq \sum_k \alpha_k \ln Z(\boldsymbol{w}^k \bullet \boldsymbol{\theta}). \tag{20.39}$$

This result allows us to define an approximate log-likelihood function:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) \geq \ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \boldsymbol{E}_{\mathcal{D}}[f_i] - \sum_k \alpha_k \ln Z(\boldsymbol{w}^k \bullet \boldsymbol{\theta}).$$

b. Assuming that the submodels are more tractable, we can efficiently evaluate this lower bound, and also compute its derivatives to be used during optimization. Show that

$$\frac{\partial}{\partial \theta_i}\ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_{\mathcal{D}}[f_i] - \sum_k \alpha_k \boldsymbol{w}_i^k \boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i]. \tag{20.40}$$

c. We can provide a bound on the error of this approximation. Specifically, show that:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_k \alpha_k \boldsymbol{D}(P_{\boldsymbol{\theta}} \| P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}),$$

where the KL-divergence measures are defined in terms of the natural logarithm. Thus, we see that the error is an average of the divergence between the true distribution and each of the approximating submodels.

d. The justification for this approach is that we can make the submodels simpler than the original model by having some parameters be equal to 0, thereby eliminating the resulting feature from the model structure. Other than this constraint, however, we still have considerable freedom in choosing the submodel weight vectors $\boldsymbol{w}^k$. Assume that each weight vector $\{\boldsymbol{w}_k\}$ maximizes $\ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D})$ subject to the constraint of equation (20.38) plus additional constraints requiring that certain entries w_i^k be equal to 0. Show that if i, k and l are such that $\theta_i \neq 0$ and neither w_i^k nor w_i^l is constrained to be zero, then

$$\boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i] = \boldsymbol{E}_{P_{\boldsymbol{w}^l \bullet \boldsymbol{\theta}}}[f_i].$$

Conclude from this result that for each such i and k, we have that

$$\boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i].$$

Exercise 20.14

Consider a particular parameterization $(\boldsymbol{\theta}, \eta)$ to Max-margin. Show how we can use second-best MAP inference to either find a violated constraint or guarantee that all constraints are satisfied.

Exercise 20.15

Let $\mathcal{H}^*$ be a Markov network where the maximum degree of a node is d^*. Show that if we have an infinitely large data set $\mathcal{D}$ generated from $\mathcal{H}^*$ (so that independence tests are evaluated perfectly), then the Build-PMap-Skeleton procedure of algorithm 3.3 reconstructs the correct Markov structure $\mathcal{H}^*$.

Exercise 20.16⋆

Prove proposition 20.6. (Hint: Take the derivative of equation (20.34) and set it to zero.)

Exercise 20.17

In this exercise, you will prove proposition 20.7, which allows us to find a closed-form optimum to multiple features in a log-linear model.

a. Prove the following proposition.

Proposition 20.8 *Let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model, and suppose that $f_1, \ldots, f_l$ are mutually exclusive binary features, that is, there is no ξ and $i \neq j$, so that $f_i(\xi) = 1$ and $f_j(\xi) = 1$. Then,*

$$\max_{\theta_1,\ldots,\theta_l} \left[\text{score}_L((\theta_1, \ldots, \theta_l, \boldsymbol{\theta}^0_{-\{1,\ldots,l\}}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{I}\boldsymbol{D}(\hat{\boldsymbol{p}} \| \boldsymbol{p}^0),$$

where $\hat{\boldsymbol{p}}$ is a distribution over $l+1$ values with $\hat{p}_i = \boldsymbol{E}_{\mathcal{D}}[f_i]$, and $\boldsymbol{p}^0$ is a distribution with $p^0(i) = P_{\boldsymbol{\theta}}(f_i)$.

b. Use this proposition to prove proposition 20.7.

Exercise 20.18

Derive an analog to proposition 20.6 for the case of the L_1 regularized log-likelihood objective.