Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–87

DOI: xxxxxx

now
the essence of knowledge

An Introduction to Conditional Random Fields

Charles Sutton[1] and Andrew McCallum[2]

[1] *EdinburghEH8 9AB, UK, csutton@inf.ed.ac.uk*
[2] *Amherst, MA01003, USA, mccallum@cs.umass.edu*

Abstract

Often we wish to predict a large number of variables that depend on each other as well as on other observed variables. Structured prediction methods are essentially a combination of classification and graphical modeling, combining the ability of graphical models to compactly model multivariate data with the ability of classification methods to perform prediction using large sets of input features. This tutorial describes *conditional random fields*, a popular probabilistic method for structured prediction. CRFs have seen wide application in natural language processing, computer vision, and bioinformatics. We describe methods for inference and parameter estimation for CRFs, including practical issues for implementing large scale CRFs. We do not assume previous knowledge of graphical modeling, so this tutorial is intended to be useful to practitioners in a wide variety of fields.

Contents

1

Introduction

Fundamental to many applications is the ability to predict multiple variables that depend on each other. Such applications are as diverse as classifying regions of an image [60], estimating the score in a game of Go [111], segmenting genes in a strand of DNA [5], and extracting syntax from natural-language text [123]. In such applications, we wish to predict a vector $\mathbf{y} = \{y_0, y_1, \ldots, y_T\}$ of random variables given an observed feature vector $\mathbf{x}$. A relatively simple example from natural-language processing is part-of-speech tagging, in which each variable y_s is the part-of-speech tag of the word at position s, and the input $\mathbf{x}$ is divided into feature vectors $\{\mathbf{x}_0, \mathbf{x}_1 \ldots \mathbf{x}_T\}$. Each $\mathbf{x}_s$ contains various information about the word at position s, such as its identity, orthographic features such as prefixes and suffixes, membership in domain-specific lexicons, and information in semantic databases such as WordNet.

One approach to this multivariate prediction problem, especially if our goal is to maximize the number of labels y_s that are correctly classified, is to learn an independent per-position classifier that maps $\mathbf{x} \mapsto y_s$ for each s. The difficulty, however, is that the output variables have complex dependencies. For example, neighboring words in a doc-

ument or neighboring regions in a image tend to have similar labels. Or the output variables may represent a complex structure such as a parse tree, in which a choice of what grammar rule to use near the top of the tree can have a large effect on the rest of the tree.

A natural way to represent the manner in which output variables depend on each other is provided by graphical models. Graphical models—which include such diverse model families as Bayesian networks, neural networks, factor graphs, Markov random fields, Ising models, and others—represent a complex distribution over many variables as a product of local *factors* on smaller subsets of variables. It is then possible to describe how a given factorization of the probability density corresponds to a particular set of conditional independence relationships satisfied by the distribution. This correspondence makes modeling much more convenient, because often our knowledge of the domain suggests reasonable conditional independence assumptions, which then determine our choice of factors.

Much work in learning with graphical models, especially in statistical natural-language processing, has focused on *generative* models that explicitly attempt to model a joint probability distribution $p(\mathbf{y}, \mathbf{x})$ over inputs and outputs. Although there are advantages to this approach, it also has important limitations. Not only can the dimensionality of $\mathbf{x}$ be very large, but the features have complex dependencies, so constructing a probability distribution over them is difficult. Modelling the dependencies among inputs can lead to intractable models, but ignoring them can lead to reduced performance.

A solution to this problem is to model the conditional distribution $p(\mathbf{y}|\mathbf{x})$ directly, which is all that is needed for classification. This is a conditional random field (CRF). CRFs are essentially a way of combining the advantages of classification and graphical modeling, combining the ability to compactly model multivariate data with the ability to leverage a large number of input features for prediction. The advantage to a conditional model is that dependencies that involve only variables in $\mathbf{x}$ play no role in the conditional model, so that an accurate conditional model can have much simpler structure than a joint model. The difference between generative models and CRFs is thus exactly analogous to the difference between the naive Bayes and logistic re-

gression classifiers. Indeed, the multinomial logistic regression model can be seen as the simplest kind of CRF, in which there is only one output variable.

There has been a large amount of applied interest in CRFs. Successful applications have included text processing [89, 107, 108], bioinformatics [106, 65], and computer vision [43, 53]. Although early applications of CRFs used linear chains, recent applications of CRFs have also used more general graphical structures. General graphical structures are useful for predicting complex structures, such as graphs and trees, and for relaxing the iid assumption among entities, as in relational learning [121].

This tutorial describes modeling, inference, and parameter estimation using conditional random fields. We do not assume previous knowledge of graphical modeling, so this tutorial is intended to be useful to practitioners in a wide variety of fields. We begin by describing modelling issues in CRFs (Chapter 2), including linear-chain CRFs, CRFs with general graphical structure, and hidden CRFs that include latent variables. We describe how CRFs can be viewed both as a generalization of the well-known logistic regression procedure, and as a discriminative analogue of the hidden Markov model.

In the next two chapters, we describe inference (Chapter 3) and learning (Chapter 4) in CRFs. The two procedures are closely coupled, because learning usually calls inference as a subroutine. Although the inference algorithms that we discuss are standard algorithms for graphical models, the fact that inference is embedded within an outer parameter estimation procedure raises additional issues. Finally, we discuss relationships between CRFs and other families of models, including other structured prediction methods, neural networks, and maximum entropy Markov models (Chapter 5).

Implementation Details

Throughout this monograph, we try to point out implementation details that are sometimes elided in the research literature. For example, we discuss issues relating to feature engineering (Section 2.6), avoiding numerical overflow during inference (Section 3.3), and the scalability

of CRF training on some benchmark problems (Section 4.5).

Since this is the first of our sections on implementation details, it seems appropriate to mention some of the available implementations of CRFs. At the time of writing, a few popular implementations are:

CRF++	`http://crfpp.sourceforge.net/`
MALLET	`http://mallet.cs.umass.edu/`
GRMM	`http://mallet.cs.umass.edu/grmm/`
CRFSuite	`http://www.chokkan.org/software/crfsuite/`
FACTORIE	`http://www.factorie.cc`

Also, software for Markov Logic networks (such as Alchemy: `http://alchemy.cs.washington.edu/`) can be used to build CRF models. Alchemy, GRMM, and FACTORIE are the only toolkits of which we are aware that handle arbitrary graphical structure.

2

Modeling

In this chapter, we describe conditional random fields from a modeling perspective, explaining how a CRF represents distributions over structured outputs as a function of a high-dimensional input vector. CRFs can be understood both as an extension of the logistic regression classifier to arbitrary graphical structures, or as a discriminative analog of generative models of structured data, an such as hidden Markov models.

We begin with a brief introduction to graphical modeling (Section 2.1) and a description of generative and discriminative models in NLP (Section 2.2). Then we will be able to present the formal definition of conditional random field, both for the commonly-used case of linear chains (Section 2.3), and for general graphical structures (Section 2.4). Finally, we present some examples of how different structures are used in applications (Section 2.5), and some implementation details concerning feature engineering (Section 2.6).

2.1 Graphical Modeling

Graphical modeling is a powerful framework for representation and inference in multivariate probability distributions. It has proven useful in diverse areas of stochastic modeling, including coding theory [77], computer vision [34], knowledge representation [88], Bayesian statistics [33], and natural-language processing [54, 9].

Distributions over many variables can be expensive to represent naïvely. For example, a table of joint probabilities of n binary variables requires storing $O(2^n)$ floating-point numbers. The insight of the graphical modeling perspective is that a distribution over very many variables can often be represented as a product of *local functions* that each depend on a much smaller subset of variables. This factorization turns out to have a close connection to certain conditional independence relationships among the variables—both types of information being easily summarized by a graph. Indeed, this relationship between factorization, conditional independence, and graph structure comprises much of the power of the graphical modeling framework: the conditional independence viewpoint is most useful for designing models, and the factorization viewpoint is most useful for designing inference algorithms.

In the rest of this section, we introduce graphical models from both the factorization and conditional independence viewpoints, focusing on those models which are based on undirected graphs. A more detailed modern perspective on graphical modelling and approximate inference is available in a textbook by Koller and Friedman [49].

2.1.1 Undirected Models

We consider probability distributions over sets of random variables $V = X \cup Y$, where X is a set of *input variables* that we assume are observed, and Y is a set of *output variables* that we wish to predict. Every variable $s \in V$ takes outcomes from a set $\mathcal{V}$, which can be either continuous or discrete, although we consider only the discrete case in this tutorial. An arbitrary assignment to X is denoted by a vector $\mathbf{x}$. Given a variable $s \in X$, the notation x_s denotes the value assigned to s by $\mathbf{x}$, and similarly for an assignment to a subset $a \subset X$ by $\mathbf{x}_a$. The notation

$\mathbf{1}_{\{x=x'\}}$ denotes an indicator function of x which takes the value 1 when $x = x'$ and 0 otherwise. We also require notation for marginalization. For a fixed variable assignment y_s, we use the summation $\sum_{\mathbf{y}\setminus y_s}$ to indicate a summation over all possible assignments $\mathbf{y}$ whose value for variable s is equal to y_s.

Suppose that we believe that a probability distribution p of interest can be represented by a product of *factors* of the form $\Psi_a(\mathbf{x}_a, \mathbf{y}_a)$, where each factor has scope $a \subseteq V$. This factorization can allow us to represent p much more efficiently, because the sets a may be much smaller than the full variable set V. We assume that without loss of generality that each distinct set a has at most one factor Ψ_a.

An undirected graphical model is a family of probability distributions that factorize according to given collection of scopes. Formally, given a collection of subsets $\mathcal{F} = a \subset V$, an *undirected graphical model* is defined as the set of all distributions that can be written in the form

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{a \in \mathcal{F}} \Psi_a(\mathbf{x}_a, \mathbf{y}_a), \tag{2.1}$$

for any choice of *local function* $F = \{\Psi_a\}$, where $\Psi_a : \mathcal{V}^{|a|} \to \Re^+$. (These functions are also called *factors* or *compatibility functions.*) We will occasionally use the term *random field* to refer to a particular distribution among those defined by an undirected model. The reason for the term *graphical model* will become apparent shortly, when we discuss how the factorization of (2.1) can be represented as a graph.

The constant Z is a normalization factor that ensures the distribution p sums to 1. It is defined as

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{a \in \mathcal{F}} \Psi_a(\mathbf{x}_a, \mathbf{y}_a). \tag{2.2}$$

The quantity Z, considered as a function of the set F of factors, is sometime called the *partition function.* Notice that the summation in (2.2) is over the exponentially many possible assignments to $\mathbf{x}$ and $\mathbf{y}$. For this reason, computing Z is intractable in general, but much work exists on how to approximate it.

We will generally assume further that each local function has the

form

$$\Psi_a(\mathbf{x}_a, \mathbf{y}_a) = \exp\left\{\sum_k \theta_{ak} f_{ak}(\mathbf{x}_a, \mathbf{y}_a)\right\}, \tag{2.3}$$

for some real-valued parameter vector θ_a, and for some set of *feature functions* or *sufficient statistics* $\{f_{ak}\}$. If $\mathbf{x}$ and $\mathbf{y}$ are discrete, then this assumption is without loss of generality, because we can have features have indicator functions for every possible value, that is, if we include one feature function $f_{ak}(\mathbf{x}_a, \mathbf{y}_a) = \mathbf{1}_{\{\mathbf{x}_a = \mathbf{x}_a^*\}} \mathbf{1}_{\{\mathbf{y}_a = \mathbf{y}_a^*\}}$ for every possible value $\mathbf{x}_a^*$ and $\mathbf{y}_a^*$.

Also, a consequence of this parameterization is that the family of distributions over V parameterized by θ is an exponential family. Indeed, much of the discussion in this tutorial about parameter estimation for CRFs applies to exponential families in general.

As we have mentioned, there is a close connection between the factorization of a graphical model and the conditional independencies among the variables in its domain. This connection can be understood by means of an undirected graph known as a *Markov network*, which directly represents conditional independence relationships in a multivariate distribution. Let G be an undirected graph with variables V, that is, G has one node for every random variable of interest. For a variable $s \in V$, let $N(s)$ denote the neighbors of s. Then we say that a distribution p is *Markov with respect to* G if it meets the local Markov property: for any two variables $s, t \in V$, the variable s is independent of t conditioned on its neighbors $N(s)$. Intuitively, this means that the neighbors of s contain all of the information necessary to predict its value.

Given a factorization of a distribution p as in (2.1), an equivalent Markov network can be constructed by connecting all pairs of variables that share a local function. It is straightforward to show that p is Markov with respect to this graph, because the conditional distribution $p(x_s|\mathbf{x}_{N(s)})$ that follows from (2.1) is a function only of variables that appear in the Markov blanket. In other words, if p factorizes according to G, then p is Markov with respect to G.

The converse direction also holds, as long as p is strictly positive. This is stated in the following classical result [42, 7]:

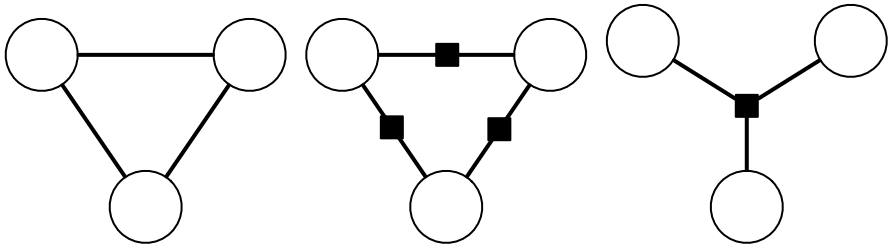

Fig. 2.1 A Markov network with an ambiguous factorization. Both of the factor graphs at right factorize according to the Markov network at left.

Theorem 2.1 (Hammersley-Clifford). Suppose p is a strictly positive distribution, and G is an undirected graph that indexes the domain of p. Then p is Markov with respect to G if and only if p factorizes according to G.

A Markov network has an undesirable ambiguity from the factorization perspective, however. Consider the three-node Markov network in Figure 2.1 (left). Any distribution that factorizes as $p(x_1, x_2, x_3) \propto f(x_1, x_2, x_3)$ for some positive function f is Markov with respect to this graph. However, we may wish to use a more restricted parameterization, where $p(x_1, x_2, x_3) \propto f(x_1, x_2)g(x_2, x_3)h(x_1, x_3)$. This second model family is smaller, and therefore may be more amenable to parameter estimation. But the Markov network formalism cannot distinguish between these two parameterizations. In order to state models more precisely, the factorization (2.1) can be represented directly by means of a *factor graph* [50]. A factor graph is a bipartite graph $G = (V, F, E)$ in which a variable node $v_s \in V$ is connected to a factor node $\Psi_a \in F$ if v_s is an argument to Ψ_a. An example of a factor graph is shown graphically in Figure 2.2 (right). In that figure, the circles are variable nodes, and the shaded boxes are factor nodes. Notice that, unlike the undirected graph, the factor graph depicts the factorization of the model unambiguously.

2.1.2 Directed Models

Whereas the local functions in an undirected model need not have a direct probabilistic interpretation, a *directed graphical model* describes how a distribution factorizes into local conditional probability distributions. Let $G = (V, E)$ be a directed acyclic graph, in which $\pi(v)$ are the parents of v in G. A directed graphical model is a family of distributions that factorize as:

$$p(\mathbf{y}, \mathbf{x}) = \prod_{v \in V} p(y_v | \mathbf{y}_{\pi(v)}). \tag{2.4}$$

It can be shown by structural induction on G that p is properly normalized. Directed models can be thought of as a kind of factor graph, in which the individual factors are locally normalized in a special fashion so that globally $Z = 1$. Directed models are often used as generative models, as we explain in Section 2.2.3. An example of a directed model is the naive Bayes model (2.5), which is depicted graphically in Figure 2.2 (left).

2.2 Generative versus Discriminative Models

In this section we discuss several examples applications of simple graphical models to natural language processing. Although these examples are well-known, they serve both to clarify the definitions in the previous section, and to illustrate some ideas that will arise again in our discussion of conditional random fields. We devote special attention to the hidden Markov model (HMM), because it is closely related to the linear-chain CRF.

2.2.1 Classification

First we discuss the problem of *classification*, that is, predicting a single discrete class variable y given a vector of features $\mathbf{x} = (x_1, x_2, \ldots, x_K)$. One simple way to accomplish this is to assume that once the class label is known, all the features are independent. The resulting classifier is called the *naive Bayes classifier*. It is based on a joint probability

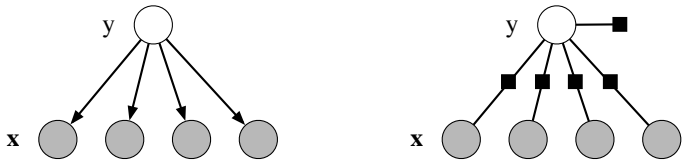

Fig. 2.2 The naive Bayes classifier, as a directed model (left), and as a factor graph (right).

model of the form:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^{K} p(x_k|y). \tag{2.5}$$

This model can be described by the directed model shown in Figure 2.2 (left). We can also write this model as a factor graph, by defining a factor $\Psi(y) = p(y)$, and a factor $\Psi_k(y, x_k) = p(x_k|y)$ for each feature x_k. This factor graph is shown in Figure 2.2 (right).

Another well-known classifier that is naturally represented as a graphical model is logistic regression (sometimes known as the *maximum entropy classifier* in the NLP community). In statistics, this classifier is motivated by the assumption that the log probability, $\log p(y|\mathbf{x})$, of each class is a linear function of $\mathbf{x}$, plus a normalization constant. This leads to the conditional distribution:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\theta_y + \sum_{j=1}^{K} \theta_{y,j} x_j\right\}, \tag{2.6}$$

where $Z(\mathbf{x}) = \sum_y \exp\{\theta_y + \sum_{j=1}^{K} \theta_{y,j} x_j\}$ is a normalizing constant, and θ_y is a bias weight that acts like $\log p(y)$ in naive Bayes. Rather than using one weight vector per class, as in (2.6), we can use a different notation in which a single set of weights is shared across all the classes. The trick is to define a set of *feature functions* that are nonzero only for a single class. To do this, the feature functions can be defined as $f_{y',j}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}} x_j$ for the feature weights and $f_{y'}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}}$ for the bias weights. Now we can use f_k to index each feature function $f_{y',j}$, and θ_k to index its corresponding weight $\theta_{y',j}$. Using this notational

trick, the logistic regression model becomes:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y, \mathbf{x})\right\}. \tag{2.7}$$

We introduce this notation because it mirrors the notation for conditional random fields that we will present later.

2.2.2 Sequence Models

Classifiers predict only a single class variable, but the true power of graphical models lies in their ability to model many variables that are interdependent. In this section, we discuss perhaps the simplest form of dependency, in which the output variables are arranged in a sequence. To motivate this kind of model, we discuss an application from natural language processing, the task of *named-entity recognition* (NER). NER is the problem of identifying and classifying proper names in text, including locations, such as *China*; people, such as *George Bush*; and organizations, such as the *United Nations*. The named-entity recognition task is, given a sentence, to segment which words are part of entities, and to classify each entity by type (person, organization, location, and so on). The challenge of this problem is that many named entities are too rare to appear even in a large training set, and therefore the system must identify them based only on context.

One approach to NER is to classify each word independently as one of either PERSON, LOCATION, ORGANIZATION, or OTHER (meaning not an entity). The problem with this approach is that it assumes that given the input, all of the named-entity labels are independent. In fact, the named-entity labels of neighboring words are dependent; for example, while *New York* is a location, *New York Times* is an organization. One way to relax this independence assumption is to arrange the output variables in a linear chain. This is the approach taken by the hidden Markov model (HMM) [96]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^{\mathrm{T}}$ by assuming that there is an underlying sequence of *states* $Y = \{y_t\}_{t=1}^{\mathrm{T}}$ drawn from a finite state set S. In the named-entity example, each observation x_t is the identity of the word at position t, and each state y_t is the named-entity label,

that is, one of the entity types PERSON, LOCATION, ORGANIZATION, and OTHER.

To model the joint distribution $p(\mathbf{y}, \mathbf{x})$ tractably, an HMM makes two independence assumptions. First, it assumes that each state depends only on its immediate predecessor, that is, each state y_t is independent of all its ancestors $y_1, y_2, \ldots, y_{t-2}$ given the preceding state y_{t-1}. Second, it also assumes that each observation variable x_t depends only on the current state y_t. With these assumptions, we can specify an HMM using three probability distributions: first, the distribution $p(y_1)$ over initial states; second, the transition distribution $p(y_t|y_{t-1})$; and finally, the observation distribution $p(x_t|y_t)$. That is, the joint probability of a state sequence $\mathbf{y}$ and an observation sequence $\mathbf{x}$ factorizes as

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{\mathrm{T}} p(y_t|y_{t-1})p(x_t|y_t), \tag{2.8}$$

where, to simplify notation, we write the initial state distribution $p(y_1)$ as $p(y_1|y_0)$. In natural language processing, HMMs have been used for sequence labeling tasks such as part-of-speech tagging, named-entity recognition, and information extraction.

2.2.3 Comparison

Of the models described in this section, two are generative (the naive Bayes and hidden Markov models) and one is discriminative (the logistic regression model). In a general, *generative models* are models of the joint distribution $p(y, \mathbf{x})$, and like naive Bayes have the form $p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$. In other words, they describe how the output is probabilistically generated as a function of the input. *Discriminative models*, on the other hand, focus solely on the conditional distribution $p(y|\mathbf{x})$. In this section, we discuss the differences between generative and discriminative modeling, and the potential advantages of discriminative modeling. For concreteness, we focus on the examples of naive Bayes and logistic regression, but the discussion in this section applies equally as well to the differences between arbitrarily structured generative models and conditional random fields.

The main difference is that a conditional distribution $p(\mathbf{y}|\mathbf{x})$ does

not include a model of $p(\mathbf{x})$, which is not needed for classification anyway. The difficulty in modeling $p(\mathbf{x})$ is that it often contains many highly dependent features that are difficult to model. For example, in named-entity recognition, an HMM relies on only one feature, the word's identity. But many words, especially proper names, will not have occurred in the training set, so the word-identity feature is uninformative. To label unseen words, we would like to exploit other features of a word, such as its capitalization, its neighboring words, its prefixes and suffixes, its membership in predetermined lists of people and locations, and so on.

The principal advantage of discriminative modeling is that it is better suited to including rich, overlapping features. To understand this, consider the family of naive Bayes distributions (2.5). This is a family of joint distributions whose conditionals all take the "logistic regression form" (2.7). But there are many other joint models, some with complex dependencies among $\mathbf{x}$, whose conditional distributions also have the form (2.7). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(\mathbf{x})$. CRFs make independence assumptions among $\mathbf{y}$, and assumptions about how the $\mathbf{y}$ can depend on $\mathbf{x}$, but not among $\mathbf{x}$. This point can also be understood graphically: Suppose that we have a factor graph representation for the joint distribution $p(\mathbf{y}, \mathbf{x})$. If we then construct a graph for the conditional distribution $p(\mathbf{y}|\mathbf{x})$, any factors that depend only on $\mathbf{x}$ vanish from the graphical structure for the conditional distribution. They are irrelevant to the conditional because they are constant with respect to $\mathbf{y}$.

To include interdependent features in a generative model, we have two choices: enhance the model to represent dependencies among the inputs, or make simplifying independence assumptions, such as the naive Bayes assumption. The first approach, enhancing the model, is often difficult to do while retaining tractability. For example, it is hard to imagine how to model the dependence between the capitalization of a word and its suffixes, nor do we particularly wish to do so, since we always observe the test sentences anyway. The second approach—to include a large number of dependent features in a generative model, but to include independence assumptions among them—is possible, and in

some domains can work well. But it can also be problematic because the independence assumptions can hurt performance. For example, although the naive Bayes classifier performs well in document classification, it performs worse on average across a range of applications than logistic regression [16].

Furthermore, naive Bayes can produce poor probability estimates. As an illustrative example, imagine training naive Bayes on a data set in which all the features are repeated, that is, $\mathbf{x} = (x_1, x_1, x_2, x_2, \ldots, x_K, x_K)$. This will increase the confidence of the naive Bayes probability estimates, even though no new information has been added to the data. Assumptions like naive Bayes can be especially problematic when we generalize to sequence models, because inference essentially combines evidence from different parts of the model. If probability estimates of the label at each sequence position are overconfident, it might be difficult to combine them sensibly.

The difference between naive Bayes and logistic regression is due *only* to the fact that the first is generative and the second discriminative; the two classifiers are, for discrete input, identical in all other respects. Naive Bayes and logistic regression consider the same hypothesis space, in the sense that any logistic regression classifier can be converted into a naive Bayes classifier with the same decision boundary, and vice versa. Another way of saying this is that the naive Bayes model (2.5) defines the same family of distributions as the logistic regression model (2.7), if we interpret it generatively as

$$p(y, \mathbf{x}) = \frac{\exp\{\sum_k \theta_k f_k(y, \mathbf{x})\}}{\sum_{\tilde{y}, \tilde{\mathbf{x}}} \exp\{\sum_k \theta_k f_k(\tilde{y}, \tilde{\mathbf{x}})\}}. \tag{2.9}$$

This means that if the naive Bayes model (2.5) is trained to maximize the conditional likelihood, we recover the same classifier as from logistic regression. Conversely, if the logistic regression model is interpreted generatively, as in (2.9), and is trained to maximize the joint likelihood $p(y, \mathbf{x})$, then we recover the same classifier as from naive Bayes. In the terminology of Ng and Jordan [85], naive Bayes and logistic regression form a *generative-discriminative pair*. For a recent theoretical perspective on generative and discriminative models, see Liang and Jordan [61].

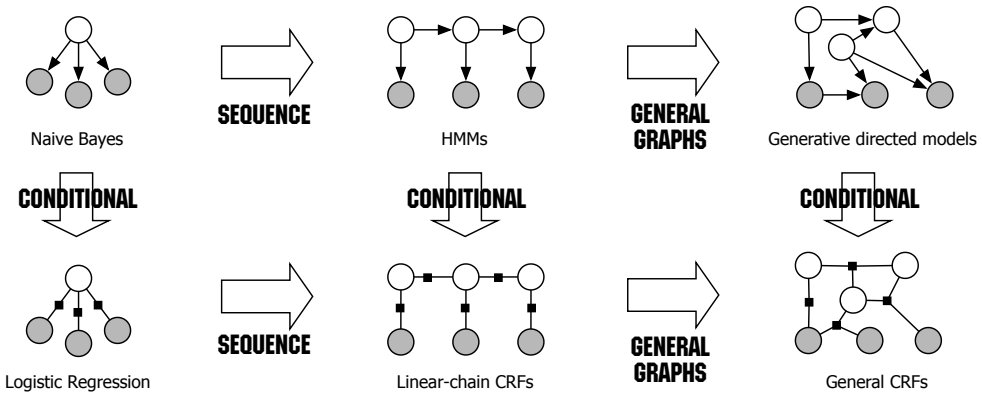

Fig. 2.3 Diagram of the relationship between naive Bayes, logistic regression, HMMs, linear-chain CRFs, generative models, and general CRFs.

One perspective for gaining insight into the difference between generative and discriminative modeling is due to Minka [80]. Suppose we have a generative model p_g with parameters θ. By definition, this takes the form

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta) p_g(\mathbf{x}|\mathbf{y}; \theta). \tag{2.10}$$

But we could also rewrite p_g using Bayes rule as

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta) p_g(\mathbf{y}|\mathbf{x}; \theta), \tag{2.11}$$

where $p_g(\mathbf{x}; \theta)$ and $p_g(\mathbf{y}|\mathbf{x}; \theta)$ are computed by inference, i.e., $p_g(\mathbf{x}; \theta) = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta)$ and $p_g(\mathbf{y}|\mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta)/p_g(\mathbf{x}; \theta)$.

Now, compare this generative model to a discriminative model over the same family of joint distributions. To do this, we define a prior $p(\mathbf{x})$ over inputs, such that $p(\mathbf{x})$ could have arisen from p_g with some parameter setting. That is, $p(\mathbf{x}) = p_c(\mathbf{x}; \theta') = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}|\theta')$. We combine this with a conditional distribution $p_c(\mathbf{y}|\mathbf{x}; \theta)$ that could also have arisen from p_g, that is, $p_c(\mathbf{y}|\mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta)/p_g(\mathbf{x}; \theta)$. Then the resulting distribution is

$$p_c(\mathbf{y}, \mathbf{x}) = p_c(\mathbf{x}; \theta') p_c(\mathbf{y}|\mathbf{x}; \theta). \tag{2.12}$$

By comparing (2.11) with (2.12), it can be seen that the conditional approach has more freedom to fit the data, because it does not require

that $\theta = \theta'$. Intuitively, because the parameters θ in (2.11) are used in both the input distribution and the conditional, a good set of parameters must represent both well, potentially at the cost of trading off accuracy on $p(\mathbf{y}|\mathbf{x})$, the distribution we care about, for accuracy on $p(\mathbf{x})$, which we care less about. On the other hand, this added freedom brings about an increased risk of overfitting the training data, and generalizing worse on unseen data.

To be fair, however, generative models have several advantages of their own. First, generative models can be more natural for handling latent variables, partially-labeled data, and unlabelled data. In the most extreme case, when the data is entirely unlabeled, generative models can be applied in an unsupervised fashion, whereas unsupervised learning in discriminative models is less natural and is still an active area of research.

Second, on some data a generative model can perform better than a discriminative model, intuitively because the input model $p(\mathbf{x})$ may have a smoothing effect on the conditional. Ng and Jordan [85] argue that this effect is especially pronounced when the data set is small. For any particular data set, it is impossible to predict in advance whether a generative or a discriminative model will perform better. Finally, sometimes either the problem suggests a natural generative model, or the application requires the ability to predict both future inputs and future outputs, making a generative model preferable.

Because a generative model takes the form $p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$, it is often natural to represent a generative model by a directed graph in which in outputs $\mathbf{y}$ topologically precede the inputs. Similarly, we will see that it is often natural to represent a discriminative model by a undirected graph, although this need not always be the case.

The relationship between naive Bayes and logistic regression mirrors the relationship between HMMs and linear-chain CRFs. Just as naive Bayes and logistic regression are a generative-discriminative pair, there is a discriminative analogue to the hidden Markov model, and this analogue is a particular special case of conditional random field, as we explain in the next section. This analogy between naive Bayes, logistic regression, generative models, and conditional random fields is depicted

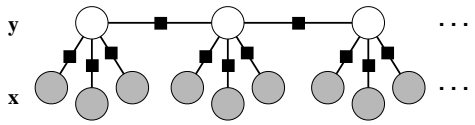

Fig. 2.4 Graphical model of an HMM-like linear-chain CRF.

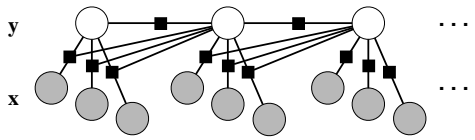

Fig. 2.5 Graphical model of a linear-chain CRF in which the transition score depends on the current observation.

in Figure 2.3.

2.3 Linear-chain CRFs

To motivate our introduction of linear-chain conditional random fields, we begin by considering the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that follows from the joint distribution $p(\mathbf{y}, \mathbf{x})$ of an HMM. The key point is that this conditional distribution is in fact a conditional random field with a particular choice of feature functions.

First, we rewrite the HMM joint (2.8) in a form that is more amenable to generalization. This is

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{i,j \in S} \theta_{ij} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_{i \in S} \sum_{o \in O} \mu_{oi} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{x_t = o\}} \right\}, \tag{2.13}$$

where $\theta = \{\theta_{ij}, \mu_{oi}\}$ are the real-valued parameters of the distribution and Z is a normalization constant chosen so the distribution sums to one.[1] It can be seen that (2.13) describes exactly the class of HMMs.

[1] Not all choices of θ are valid, because the summation defining Z, that is, $Z = \sum_{\mathbf{y}} \sum_{\mathbf{x}} \prod_{t=1}^{T} \exp \left\{ \sum_{i,j \in S} \theta_{ij} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_{i \in S} \sum_{o \in O} \mu_{oi} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{x_t = o\}} \right\}$, might not converge. An example of this is a model with one state where $\theta_{00} > 0$. This issue is typically not an issue for CRFs, because in a CRF the summation within Z is

Every HMM can be written in this form by setting $\theta_{ij} = \log p(y' = i|y = j)$ and $\mu_{oi} = \log p(x = o|y = i)$. The converse direction is more complicated, and not relevant for our purposes here. The main point is that despite this added flexibility in the parameterization (2.13), we have not added any distributions to the family.

We can write (2.13) more compactly by introducing the concept of *feature functions*, just as we did for logistic regression in (2.7). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (2.13), there needs to be one feature $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$ for each transition (i, j) and one feature $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{x=o\}}$ for each state-observation pair (i, o). We refer to a feature function generically as f_k, where f_k ranges over both all of the f_{ij} and all of the f_{io}. Then we can write an HMM as:

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t, y_{t-1}, x_t)\right\}. \tag{2.14}$$

Again, equation (2.14) defines exactly the same family of distributions as (2.13), and therefore as the original HMM equation (2.8).

The last step is to write the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that results from the HMM (2.14). This is

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}'} p(\mathbf{y}', \mathbf{x})} = \frac{\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{\mathbf{y}'}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y'_t, y'_{t-1}, x_t)\right\}}. \tag{2.15}$$

This conditional distribution (2.15) is a particular kind of linear-chain CRF, namely, one that includes features only for the current word's identity. But many other linear-chain CRFs use richer features of the input, such as prefixes and suffixes of the current word, the identity of surrounding words, and so on. Fortunately, this extension requires little change to our existing notation. We simply allow the feature functions to be more general than indicator functions of the word's identity. This leads to the general definition of linear-chain CRFs:

---

usually over a finite set.

Definition 2.1. Let Y, X be random vectors, $\theta = \{\theta_k\} \in \Re^K$ be a parameter vector, and $\{f_k(y, y', \mathbf{x}_t)\}_{k=1}^K$ be a set of real-valued feature functions. Then a *linear-chain conditional random field* is a distribution $p(\mathbf{y}|\mathbf{x})$ that takes the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}, \tag{2.16}$$

where $Z(\mathbf{x})$ is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \tag{2.17}$$

We have just seen that if the joint $p(\mathbf{y}, \mathbf{x})$ factorizes as an HMM, then the associated conditional distribution $p(\mathbf{y}|\mathbf{x})$ is a linear-chain CRF. This HMM-like CRF is pictured in Figure 2.4. Other types of linear-chain CRFs are also useful, however. For example, typically in an HMM, a transition from state i to state j receives the same score, $\log p(y_t = j|y_{t-1} = i)$, regardless of the input. In a CRF, we can allow the score of the transition (i, j) to depend on the current observation vector, simply by adding a feature $\mathbf{1}_{\{y_t=j\}}\mathbf{1}_{\{y_{t-1}=1\}}\mathbf{1}_{\{x_t=o\}}$. A CRF with this kind of transition feature, which is commonly used in text applications, is pictured in Figure 2.5.

To indicate in the definition of linear-chain CRF that each feature function can depend on observations from any time step, we have written the observation argument to f_k as a vector $\mathbf{x}_t$, which should be understood as containing all the components of the global observations $\mathbf{x}$ that are needed for computing features at time t. For example, if the CRF uses the next word x_{t+1} as a feature, then the feature vector $\mathbf{x}_t$ is assumed to include the identity of word x_{t+1}.

Finally, note that the normalization constant $Z(\mathbf{x})$ sums over all possible state sequences, an exponentially large number of terms. Nevertheless, it can be computed efficiently by forward-backward, as we explain in Section 3.1.

2.4 General CRFs

Now we present the general definition of a conditional random field, as it was originally introduced [54]. The generalization from linear-chain CRFs to general CRFs is fairly straightforward. We simply move from using a linear-chain factor graph to a more general factor graph, and from forward-backward to more general (perhaps approximate) inference algorithms.

Definition 2.2. Let G be a factor graph over Y. Then $p(\mathbf{y}|\mathbf{x})$ is a conditional random field if for any fixed $\mathbf{x}$, the distribution $p(\mathbf{y}|\mathbf{x})$ factorizes according to G.

Thus, every conditional distribution $p(\mathbf{y}|\mathbf{x})$ is a CRF for some, perhaps trivial, factor graph. If $F = \{\Psi_a\}$ is the set of factors in G, and each factor takes the exponential family form (2.3), then the conditional distribution can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\Psi_A \in G} \exp \left\{ \sum_{k=1}^{K(A)} \theta_{ak} f_{ak}(\mathbf{y}_a, \mathbf{x}_a) \right\}. \tag{2.18}$$

In addition, practical models rely extensively on parameter tying. For example, in the linear-chain case, often the same weights are used for the factors $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ at each time step. To denote this, we partition the factors of G into $\mathcal{C} = \{C_1, C_2, \ldots C_P\}$, where each C_p is a *clique template* whose parameters are tied. This notion of clique template generalizes that in Taskar et al. [121], Sutton et al. [119], Richardson and Domingos [98], and McCallum et al. [76]. Each clique template C_p is a set of factors which has a corresponding set of sufficient statistics $\{f_{pk}(\mathbf{x}_p, \mathbf{y}_p)\}$ and parameters $\theta_p \in \Re^{K(p)}$. Then the CRF can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p), \tag{2.19}$$

where each factor is parameterized as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = \exp \left\{ \sum_{k=1}^{K(p)} \theta_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \right\}, \tag{2.20}$$

and the normalization function is

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p). \tag{2.21}$$

This notion of clique template specifies both repeated structure and parameter tying in the model. For example, in a linear-chain conditional random field, typically one clique template $C_0 = \{\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)\}_{t=1}^{\mathrm{T}}$ is used for the entire network, so $\mathcal{C} = \{C_0\}$ is a singleton set. If instead we want each factor Ψ_t to have a separate set of parameters, this would be accomplished using T templates, by taking $\mathcal{C} = \{C_t\}_{t=1}^{\mathrm{T}}$, where $C_t = \{\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)\}$. Both the set of clique templates and the number of outputs can depend on the input $\mathbf{x}$; for example, to model images, we may use different clique templates at different scales depending on the results of an algorithm for finding points of interest.

One of the most important considerations in defining a general CRF lies in specifying the repeated structure and parameter tying. A number of formalisms have been proposed to specify the clique templates. For example, *dynamic conditional random fields* [119] are sequence models which allow multiple labels at each time step, rather than single label, in a manner analogous to dynamic Bayesian networks. Second, *relational Markov networks* [121] are a type of general CRF in which the graphical structure and parameter tying are determined by an SQL-like syntax. *Markov logic networks* [98, 110] use logical formulae to specify the scopes of local functions in an undirected model. Essentially, there is a set of parameters for each first-order rule in a knowledge base. The logic portion of an MLN can be viewed as essentially a programming convention for specifying the repeated structure and parameter tying of an undirected model. *Imperatively defined factor graphs* [76] use the full expressivity of Turing-complete functions to define the clique templates, specifying both the structure of the model and the sufficient statistics f_{pk}. These functions have the flexibility to employ advanced programming ideas including recursion, arbitrary search, lazy evaluation, and memoization.

2.5 Applications of CRFs

CRFs have been applied to a variety of domains, including text processing, computer vision, and bioinformatics. One of the first large-scale applications of CRFs was by Sha and Pereira [108], who matched state-of-the-art performance on segmenting noun phrases in text. Since then, linear-chain CRFs have been applied to many problems in natural language processing, including named-entity recognition [72], feature induction for NER [71], shallow parsing [108, 120], identifying protein names in biology abstracts [107], segmenting addresses in Web pages [26], information integration [134], finding semantic roles in text [103], prediction of pitch accents [40], phone classification in speech processing [41], identifying the sources of opinions [17], word alignment in machine translation [10], citation extraction from research papers [89], extraction of information from tables in text documents [91], Chinese word segmentation [90], Japanese morphological analysis [51], and many others.

In bioinformatics, CRFs have been applied to RNA structural alignment [106] and protein structure prediction [65]. Semi-Markov CRFs [105] add somewhat more flexibility in choosing features, by allowing features functions to depend on larger segments of the input that depend on the output labelling. This can be useful for certain tasks in information extraction and especially bioinformatics.

General CRFs have also been applied to several tasks in NLP. One promising application is to performing multiple labeling tasks simultaneously. For example, Sutton et al. [119] show that a two-level dynamic CRF for part-of-speech tagging and noun-phrase chunking performs better than solving the tasks one at a time. Another application is to *multi-label classification*, in which each instance can have multiple class labels. Rather than learning an independent classifier for each category, Ghamrawi and McCallum [35] present a CRF that learns dependencies between the categories, resulting in improved classification performance. Finally, the skip-chain CRF [114] is a general CRF that represents long-distance dependencies in information extraction.

An interesting graphical CRF structure has been applied to the problem of proper-noun coreference, that is, of determining which men-

tions in a document, such as *Mr. President* and *he*, refer to the same underlying entity. McCallum and Wellner [73] learn a distance metric between mentions using a fully-connected conditional random field in which inference corresponds to graph partitioning. A similar model has been used to segment handwritten characters and diagrams [22, 93].

In computer vision, several authors have used grid-shaped CRFs [43, 53] for labeling and segmenting images. Also, for recognizing objects, Quattoni et al. [95] use a tree-shaped CRF in which latent variables are designed to recognize characteristic parts of an object.

In some applications of CRFs, efficient dynamic programs exist even though the graphical model is difficult to specify. For example, McCallum et al. [75] learn the parameters of a string-edit model in order to discriminate between matching and nonmatching pairs of strings. Also, there is work on using CRFs to learn distributions over the derivations of a grammar [99, 19, 127, 31].

2.6 Feature Engineering

In this section we describe some "tricks of the trade" that involve feature engineering. Although these apply especially to language applications, they are also useful more generally.

First, when the predicted variables are discrete, the features f_{pk} of a clique template C_p are ordinarily chosen to have a particular form:

$$f_{pk}(\mathbf{y}_c, \mathbf{x}_c) = \mathbf{1}_{\{\mathbf{y}_c = \tilde{\mathbf{y}}_c\}} q_{pk}(\mathbf{x}_c). \tag{2.22}$$

In other words, each feature is nonzero only for a single output configuration $\tilde{\mathbf{y}}_c$, but as long as that constraint is met, then the feature value depends only on the input observation. Essentially, this means that we can think of our features as depending only on the input $\mathbf{x}_c$, but that we have a separate set of weights for each output configuration. This feature representation is also computationally efficient, because computing each q_{pk} may involve nontrivial text or image processing, and it need be evaluated only once for every feature that uses it. To avoid confusion, we refer to the functions $q_{pk}(\mathbf{x}_c)$ as *observation functions* rather than as features. Examples of observation functions are "word x_t is capitalized" and "word x_t ends in *ing*".

This representation can lead to a large number of features, which can have significant memory and time requirements. For example, to match state-of-the-art results on a standard natural language task, Sha and Pereira [108] use 3.8 million features. Many of these features always zero in the training data. In particular, some observation functions q_{pk} are nonzero for certain output configurations and zero for others. This point can be confusing: One might think that such features can have no effect on the likelihood, but actually putting a negative weight on them causes an assignment that does not appear in the training data to become less likely, which improves the likelihood. For this reason, including unsupported features typically results in better accuracy. In order to save memory, however, sometimes these *unsupported features*, that is, those which never occur in the training data, are removed from the model.

As a simple heuristic for getting some of the benefits of unsupported features with less memory, we have had success with an ad hoc technique for selecting a small set of unsupported features. The idea is to add unsupported features only for likely paths, as follows: first train a CRF without any unsupported features, stopping after a few iterations; then add unsupported features $f_{pk}(\mathbf{y}_c, \mathbf{x}_c)$ for cases where $\mathbf{x}_c$ occurs in the training data for some instance $\mathbf{x}^{(i)}$, and $p(\mathbf{y}_c|\mathbf{x}^{(i)}) > \epsilon$.

McCallum [71] presents a more principled method of feature induction for CRFs, in which the model begins with a number of base features, and the training procedure adds conjunctions of those features. Alternatively, one can use feature selection. A modern method for feature selection is L_1 regularization, which we discuss in Section 4.1.1. Lavergne et al. [56] find that in the most favorable cases L_1 finds models in which only 1% of the full feature set is non-zero, but with comparable performance to a dense feature setting. They also find it useful, after optimizing the L_1-regularized likelihood to find a set of nonzero features, to fine-tune the weights of the nonzero features only using an L_2-regularized objective.

Second, if the observations are categorical rather than ordinal, that is, if they are discrete but have no intrinsic order, it is important to convert them to binary features. For example, it makes sense to learn a linear weight on $f_k(y, x_t)$ when f_k is 1 if x_t is the word *dog* and

0 otherwise, but not when f_k is the integer index of word x_t in the text's vocabulary. Thus, in text applications, CRF features are typically binary; in other application areas, such as vision and speech, they are more commonly real-valued. For real-valued features, it can help to apply standard tricks such as normalizing the features to have mean 0 and standard deviation 1 or to bin the features to convert them to categorical values.

Third, in language applications, it is sometimes helpful to include redundant factors in the model. For example, in a linear-chain CRF, one may choose to include both edge factors $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ and variable factors $\Psi_t(y_t, \mathbf{x}_t)$. Although one could define the same family of distributions using only edge factors, the redundant node factors provide a kind of backoff, which is useful when the amount of data is small compared to the number of features. (When there are hundreds of thousands of features, many data sets are small!) It is important to use regularization (Section 4.1.1) when using redundant features because it is the penalty on large weights that encourages the weight to be spread across the overlapping features.

2.7 Notes on Terminology

Different parts of the theory of graphical models have been developed independently in many different areas, so many of the concepts in this chapter have different names in different areas. For example, undirected models are commonly also referred to *Markov random fields*, *Markov networks*, and *Gibbs distributions*. As mentioned, we reserve the term "graphical model" for a family of distributions defined by a graph structure; "random field" or "distribution" for a single probability distribution; and "network" as a term for the graph structure itself. This choice of terminology is not always consistent in the literature, partly because it is not ordinarily necessary to be precise in separating these concepts.

Similarly, directed graphical models are commonly known as *Bayesian networks*, but we have avoided this term because of its confusion with the area of Bayesian statistics. The term *generative model* is an important one that is commonly used in the literature, but is not usually given a precise definition.

3

Inference

Efficient inference is critical for CRFs, both during training and for predicting the labels on new inputs. The are two inference problems that arise. First, after we have trained the model, we often predict the labels of a new input $\mathbf{x}$ using the most likely labeling $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. Second, as will be seen in Chapter 4, estimation of the parameters typically requires that we compute the marginal distribution for each edge $p(y_t, y_{t-1}|\mathbf{x})$, and also the normalizing function $Z(\mathbf{x})$.

These two inference problems can be seen as fundamentally the same operation on two different semirings [1], that is, to change the marginalization problem to the maximization problem, we simply substitute max for plus. Although for discrete variables the marginals can be computed by brute-force summation, the time required to do this is exponential in the size of Y. Indeed, both inference problems are intractable for general graphs, because any propositional satisfiability problem can be easily represented as a factor graph.

In the case of linear-chain CRFs, both inference tasks can be performed efficiently and exactly by variants of the standard dynamic-programming algorithms for HMMs. We begin by presenting these algorithms—the forward-backward algorithm for computing marginal

distributions and Viterbi algorithm for computing the most probable assignment—in Section 3.1. These algorithms are a special case of the more general belief propagation algorithm for tree-structured graphical models (Section 3.2.2). For more complex models, approximate inference is necessary. In principle, we could run any approximate inference algorithm we want, and substitute the resulting approximate marginals for the exact marginals within the gradient (4.9). This can cause issues, however, because for many optimization procedures, such as BFGS, we require an approximation to the likelihood function as well. We discuss this issue in Section 4.4.

In one sense, the inference problem for a CRF is no different than that for any graphical model, so any inference algorithm for graphical models can be used, as described in several textbooks [67, 49]. However, there are two additional issues that need to be kept in mind in the context of CRFs. The first issue is that the inference subroutine is called repeatedly during parameter estimation (Section 4.1.1 explains why), which can be computationally expensive, so we may wish to trade off inference accuracy for computational efficiency. The second issue is that when approximate inference is used, there can be complex interactions between the inference procedure and the parameter estimation procedure. We postpone discussion of these issues to Chapter 4, when we discuss parameter estimation, but it is worth mentioning them here because they strongly influence the choice of inference algorithm.

3.1 Linear-Chain CRFs

In this section, we briefly review the inference algorithms for HMMs, the forward-backward and Viterbi algorithms, and describe how they can be applied to linear-chain CRFs. These standard inference algorithms are described in more detail by Rabiner [96]. Both of these algorithms are special cases of the belief propagation algorithm described in Section 3.2.2, but we discuss the special case of linear chains in detail both because it may help to make the earlier discussion more concrete, and because it is useful in practice.

First, we introduce notation which will simplify the forward-backward recursions. An HMM can be viewed as a factor graph

$p(\mathbf{y}, \mathbf{x}) = \prod_t \Psi_t(y_t, y_{t-1}, x_t)$ where $Z = 1$, and the factors are defined as:

$$\Psi_t(j, i, x) \stackrel{\text{def}}{=} p(y_t = j | y_{t-1} = i) p(x_t = x | y_t = j). \tag{3.1}$$

If the HMM is viewed as a weighted finite state machine, then $\Psi_t(j, i, x)$ is the weight on the transition from state i to state j when the current observation is x.

Now, we review the HMM forward algorithm, which is used to compute the probability $p(\mathbf{x})$ of the observations. The idea behind forward-backward is to first rewrite the naive summation $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$ using the distributive law:

$$p(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t) \tag{3.2}$$

$$= \sum_{y_{\mathrm{T}}} \sum_{y_{\mathrm{T}-1}} \Psi_{\mathrm{T}}(y_{\mathrm{T}}, y_{\mathrm{T}-1}, x_{\mathrm{T}}) \sum_{y_{\mathrm{T}-2}} \Psi_{\mathrm{T}-1}(y_{\mathrm{T}-1}, y_{\mathrm{T}-2}, x_{\mathrm{T}-1}) \sum_{y_{\mathrm{T}-3}} \cdots \tag{3.3}$$

Now we observe that each of the intermediate sums is reused many times during the computation of the outer sum, and so we can save an exponential amount of work by caching the inner sums.

This leads to defining a set of *forward variables* α_t, each of which is a vector of size M (where M is the number of states) which stores one of the intermediate sums. These are defined as:

$$\alpha_t(j) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle 1 \ldots t \rangle}, y_t = j) \tag{3.4}$$

$$= \sum_{\mathbf{y}_{\langle 1 \ldots t-1 \rangle}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{3.5}$$

where the summation over $\mathbf{y}_{\langle 1 \ldots t-1 \rangle}$ ranges over all assignments to the sequence of random variables $y_1, y_2, \ldots, y_{t-1}$. The alpha values can be computed by the recursion

$$\alpha_t(j) = \sum_{i \in S} \Psi_t(j, i, x_t) \alpha_{t-1}(i), \tag{3.6}$$

with initialization $\alpha_1(j) = \Psi_1(j, y_0, x_1)$. (Recall that y_0 is the fixed initial state of the HMM.) It is easy to see that $p(\mathbf{x}) = \sum_{y_{\mathrm{T}}} \alpha_{\mathrm{T}}(y_{\mathrm{T}})$

by repeatedly substituting the recursion (3.6) to obtain (3.3). A formal proof would use induction.

The backward recursion is exactly the same, except that in (3.3), we push in the summations in reverse order. This results in the definition

$$\beta_t(i) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle t+1\ldots \mathrm{T}\rangle}|y_t = i) \tag{3.7}$$

$$= \sum_{\mathbf{y}_{\langle t+1\ldots \mathrm{T}\rangle}} \prod_{t'=t+1}^{\mathrm{T}} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{3.8}$$

and the recursion

$$\beta_t(i) = \sum_{j\in S} \Psi_{t+1}(j, i, x_{t+1})\beta_{t+1}(j), \tag{3.9}$$

which is initialized $\beta_{\mathrm{T}}(i) = 1$. Analogously to the forward case, we can compute $p(\mathbf{x})$ using the backward variables as $p(\mathbf{x}) = \beta_0(y_0) \stackrel{\text{def}}{=} \sum_{y_1} \Psi_1(y_1, y_0, x_1)\beta_1(y_1)$.

By combining results from the forward and backward recursions, we can compute the marginal distributions $p(y_{t-1}, y_t|\mathbf{x})$ needed for the gradient (4.6). This can be seen from either the probabilistic or the factorization perspectives. First, taking a probabilistic viewpoint we can write

$$p(y_{t-1}, y_t|\mathbf{x}) = \frac{p(\mathbf{x}|y_{t-1}, y_t)p(y_t, y_{t-1})}{p(\mathbf{x})} \tag{3.10}$$

$$= \frac{p(\mathbf{x}_{\langle 1\ldots t-1\rangle}, y_{t-1})p(y_t|y_{t-1})p(x_t|y_t)p(\mathbf{x}_{\langle t+1\ldots \mathrm{T}\rangle}|y_t)}{p(\mathbf{x})} \tag{3.11}$$

$$\propto \alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x_t)\beta_t(y_t), \tag{3.12}$$

where in the second line we have used the fact that $\mathbf{x}_{\langle 1\ldots t-1\rangle}$ is independent from $\mathbf{x}_{\langle t+1\ldots \mathrm{T}\rangle}$ and from x_t given y_{t-1}, y_t. Equivalently, from the factorization perspective, we can apply the distributive law to obtain

we see that

$$p(y_{t-1}, y_t, \mathbf{x}) = \Psi_t(y_t, y_{t-1}, x_t) \left(\sum_{\mathbf{y}_{\langle 1 \ldots t-2 \rangle}} \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right) \left(\sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right), \quad (3.13)$$

which can be computed from the forward and backward recursions as

$$p(y_{t-1}, y_t, \mathbf{x}) = \alpha_{t-1}(y_{t-1}) \Psi_t(y_t, y_{t-1}, x_t) \beta_t(y_t). \quad (3.14)$$

Once we have $p(y_{t-1}, y_t, \mathbf{x})$, we can renormalize over y_t, y_{t-1} to obtain the desired marginal $p(y_{t-1}, y_t|\mathbf{x})$.

Finally, to compute the globally most probable assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, we observe that the trick in (3.3) still works if all the summations are replaced by maximization. This yields the Viterbi recursion:

$$\delta_t(j) = \max_{i \in S} \Psi_t(j, i, x_t) \delta_{t-1}(i) \quad (3.15)$$

Now that we have described the forward-backward and Viterbi algorithms for HMMs, the generalization to linear-chain CRFs is fairly straightforward. The forward-backward algorithm for linear-chain CRFs is identical to the HMM version, except that the transition weights $\Psi_t(j, i, x_t)$ are defined differently. We observe that the CRF model (2.16) can be rewritten as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}_t), \quad (3.16)$$

where we define

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp \left\{ \sum_k \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \quad (3.17)$$

With that definition, the forward recursion (3.6), the backward recursion (3.9), and the Viterbi recursion (3.15) can be used unchanged

for linear-chain CRFs. Instead of computing $p(\mathbf{x})$ as in an HMM, in a CRF the forward and backward recursions compute $Z(\mathbf{x})$.

We mention three more specialised inference tasks that can also be solved using direct analogues of the HMM algorithms. First, assignments to $\mathbf{y}$ can be sampled from the joint posterior $p(\mathbf{y}|\mathbf{x})$ using the forward algorithm combined with a backward sampling place, in exactly the same way as an HMM. Second, if instead of finding the single best assignment $\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, we wish to find the k assignments with highest probability, we can do this also using the standard algorithms from HMMs. Finally, sometimes it is useful to compute a marginal probability $p(y_t, y_{t+1}, \ldots y_{t+k}|\mathbf{x})$ over a possibly non-contiguous range of nodes. For example, this is useful for measuring the model's confidence in its predicted labeling over a segment of input. This marginal probability can be computed efficiently using constrained forward-backward, as described by Culotta and McCallum [25].

3.2 Inference in Graphical Models

Exact inference algorithms for general graphs exist. Although these algorithms require exponential time in the worst case, they can still be efficient for graphs that occur in practice. The most popular exact algorithm, the junction tree algorithm, successively clusters variables until the graph becomes a tree. Once an equivalent tree has been constructed, its marginals can be computed using exact inference algorithms that are specific to trees. However, for certain complex graphs, the junction tree algorithm is forced to make clusters which are very large, which is why the procedure still requires exponential time in the worst case. For more details on exact inference, see Koller and Friedman [49].

For this reason, an enormous amount of effort has been devoted to approximate inference algorithms. Two classes of approximate inference algorithms have received the most attention: Monte Carlo algorithms and variational algorithms. *Monte Carlo* algorithms are stochastic algorithms that attempt to approximately produce a sample from the distribution of interest. *Variational* algorithms are algorithms that convert the inference problem into an optimization problem, by attempting to find a simple distribution that most closely matches the intractable

distribution of interest. Generally, Monte Carlo algorithms are unbiased in the sense that they guaranteed to sample from the distribution of interest given enough computation time, although it is usually impossible in practice to know when that point has been reached. Variational algorithms, on the other hand, can be much faster, but they tend to be biased, by which we mean that they tend to have a source of error that is inherent to the approximation, and cannot be easily lessened by giving them more computation time. Despite this, variational algorithms can be useful for CRFs, because parameter estimation requires performing inference many times, and so a fast inference procedure is vital to efficient training.

In the remainder of this section, we outline two examples of approximate inference algorithms, one from each of these two categories. Too much work has been done on approximate inference for us to attempt to summarize it here. Rather, our aim is to highlight the general issues that arise when using approximate inference algorithms within CRF training. In this chapter, we focus on describing the inference algorithms themselves, whereas in Chapter 4 we discuss their application to CRFs.

3.2.1 Markov Chain Monte Carlo

Currently the most popular type of Monte Carlo method for complex models is *Markov Chain Monte Carlo* (MCMC) [101]. Rather than attempting to approximate a marginal distribution $p(y_s|\mathbf{x})$ directly, MCMC methods generate approximate samples from the joint distribution $p(\mathbf{y}|\mathbf{x})$. MCMC methods work by constructing a Markov chain, whose state space is the same as that of Y, in careful way so that when the chain is simulated for a long time, the distribution over states of the chain is approximately $p(y_s|\mathbf{x})$. Suppose that we want to approximate the expectation of some function $f(\mathbf{x}, \mathbf{y})$ that depends on. Given a sample $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^M$ from a Markov chain in an MCMC method, we can approximate this expectation as:

$$\sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) f(\mathbf{x}, \mathbf{y}) \approx \frac{1}{M} \sum_{j=1}^{M} f(\mathbf{x}, \mathbf{y}^j) \tag{3.18}$$

For example, in the context of CRFs, these approximate expectations can then be used to approximate the quantities required for learning, specifically the gradient (4.6).

A simple example of an MCMC method is Gibbs sampling. In each iteration of the Gibbs sampling algorithm, each variable is resampled individually, keeping all of the other variables fixed. Suppose that we already have a sample $\mathbf{y}^j$ from iteration j. Then to generate the next sample $\mathbf{y}^{j+1}$,

(1) Set $\mathbf{y}^{j+1} \leftarrow \mathbf{y}^j$.
(2) For each $s \in V$, resample component s. Sample $\mathbf{y}_s^{j+1}$ from the distribution $p(y_s|\mathbf{y}_{\setminus s}, \mathbf{x})$.
(3) Return the resulting value of $\mathbf{y}^{j+1}$.

This procedure defines a Markov chain that can be used to approximation expectations as in (3.18). In the case of general CRFs, then using the notation from Section 2.4, this conditional probability can be computed as

$$p(y_s|\mathbf{y}_{\setminus s}, \mathbf{x}) = \kappa \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p), \tag{3.19}$$

where κ is a normalizing constant. This is much easier to compute than the joint probability $p(\mathbf{y}|\mathbf{x})$, because computing κ requires a summation only over all possible values of y_s rather than assignments to the full vector $\mathbf{y}$.

A major advantage of Gibbs sampling is that it is simple to implement. Indeed, software packages such as BUGS can take a graphical model as input and automatically compile an appropriate Gibbs sampler [66]. The main disadvantage of Gibbs sampling is that it can work poorly if $p(\mathbf{y}|\mathbf{x})$ has strong dependencies, which is often the case in sequential data. By "works poorly" we mean that it may take many iterations before the distribution over samples from the Markov chain is close to the desired distribution $p(\mathbf{y}|\mathbf{x})$.

There is an enormous literature on MCMC algorithms. The textbook by Robert and Casella [101] provides an overview. However, MCMC algorithms are not commonly applied in the context of conditional random fields. Perhaps the main reason for this is that as we

have mentioned earlier, parameter estimation by maximum likelihood requires calculating marginals many times. In the most straightforward approach, one MCMC chain would be run for each training example for each parameter setting that is visited in the course of a gradient descent algorithm. Since MCMC chains can take thousands of iterations to converge, this can be computationally prohibitive. One can imagine ways of addressing this, such as not running the chain all the way to convergence (see Section 4.4.3).

3.2.2 Belief Propagation

An important variational inference algorithm is *belief propagation (BP)*, which we explain in this section. In addition, it is a direct generalization of the exact inference algorithms for linear-chain CRFs.

Suppose that G is a tree, and we wish to compute the marginal distribution of a variable s. The intuition behind BP is that each of the neighboring factors of s makes a multiplicative contribution to the marginal of s, called a *message*, and each of these messages can be computed separately because the graph is a tree. More formally, for every factor $a \in N(s)$, call V_a the set of variables that are "upstream" of a, that is, the set of variables v for which a is between s and v. In a similar fashion, call F_a the set of factors that are upstream of a, including a itself. But now because G is a tree, the sets $\{V_a\} \cup \{s\}$ form a partition of the variables in G. This means that we can split up the summation required for the marginal into a product of independent subproblems as:

$$p(y_s) \propto \sum_{\mathbf{y} \setminus y_s} \prod_a \Psi_a(\mathbf{y}_a) \tag{3.20}$$

$$= \prod_{a \in N(s)} \sum_{\mathbf{y}_{V_a}} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b) \tag{3.21}$$

Denote each factor in the above equation by m_{as}, that is,

$$m_{as}(x_s) = \sum_{\mathbf{y}_{V_a}} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b), \tag{3.22}$$

can be thought of as a *message* from the factor a to the variable s that summarizes the impact of the network upstream of a on the belief in s.

In a similar fashion, we can define messages from variables to factors as

$$m_{sA}(x_s) = \sum_{\mathbf{y}_{V_s}} \prod_{\Psi_b \in F_s} \Psi_b(\mathbf{y}_b). \tag{3.23}$$

Then, from (3.21), we have that the marginal $p(y_s)$ is proportional to the product of all the incoming messages to variable s. Similarly, factor marginals can be computed as

$$p(\mathbf{y}_a) \propto \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(\mathbf{y}_a). \tag{3.24}$$

Here we treat a as a set a variables denoting the scope of factor Ψ_a, as we will throughout. In addition, we will sometimes use the reverse notation $c \ni s$ to mean the set of all factors c that contain the variable s.

Naively computing the messages according to (3.22) is impractical, because the messages as we have defined them require summation over possibly many variables in the graph. Fortunately, the messages can also be written using a recursion that requires only local summation. The recursion is

$$\begin{aligned} m_{as}(x_s) &= \sum_{\mathbf{y}_a \setminus y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in a \setminus s} m_{ta}(x_t) \\ m_{sa}(x_s) &= \prod_{b \in N(s) \setminus a} m_{bs}(x_s) \end{aligned} \tag{3.25}$$

That this recursion matches the explicit definition of m can be seen by repeated substitution, and proven by induction. In a tree, it is possible to schedule these recursions such that the antecedent messages are always sent before their dependents, by first sending messages from the root, and so on. This is the algorithm known as *belief propagation* [88].

In addition to computing single-variable marginals, we will also wish to compute factor marginals $p(\mathbf{y}_a)$ and joint probabilites $p(\mathbf{y})$ for a given assignment $\mathbf{y}$. (Recall that the latter problem is difficult because it requires computing the partition function $\log Z$.) First, to compute marginals over factors—or over any connected set of variables, in fact—we can use the same decomposition of the marginal as for the single-

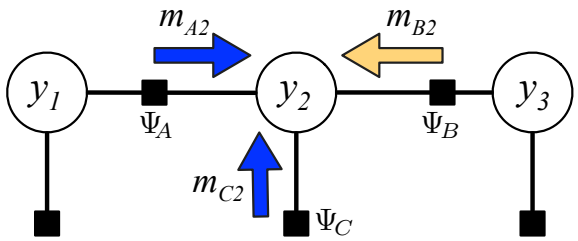

Fig. 3.1 Illustration of the correspondence between forward backward and belief propagation in linear chain graphs

variable case, and get

$$p(\mathbf{y}_a) = \kappa \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(y_s), \tag{3.26}$$

where κ is a normalization constant. In fact, a similar idea works for any connected set of variables—not just a set that happens to be the domain of some factor—although if the set is too large, then computing κ is impractical.

BP can also be used to compute the normalizing constant $Z(\mathbf{x})$. This can be done directly from the propagation algorithm, in an analogous way to the forward-backward algorithm in Section 3.1. Alternatively, there is another way to compute $Z(x)$ from only the beliefs at the end of the algorithm. In a tree structured distribution, it is always true that

$$p(\mathbf{y}) = \prod_{s \in V} p(y_s) \prod_a \frac{p(\mathbf{y}_a)}{\prod_{t \in a} p(y_t)} \tag{3.27}$$

For example, in a linear chain this amounts to

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t) \prod_{t=1}^{\mathrm{T}} \frac{p(y_t, y_{t-1})}{p(y_t)p(y_{t-1})}, \tag{3.28}$$

which, after cancelling and rearranging terms, is just another way to write the familiar equation $p(\mathbf{y}) = \prod_t p(y_t|y_{t-1})$. More generally, (3.27) can be derived using the junction tree theorem, by considering a junction tree with one cluster for each factor. Using this identity, we can compute $p(\mathbf{y})$ (or $\log Z$) from the per-variable and per-factor marginals.

If G is a tree, belief propagation computes the marginal distributions exactly. Indeed, if G is a linear chain, then BP reduces to the

forward-backward algorithm (Section 3.1). To see this, refer to Figure 3.1. The figure shows a three node linear chain along with the BP messages as we have described them in this section. To see the correspondence to forward backward, the forward message that we denoted α_2 in Section 3.1 corresponds to the product of the two messages m_{A2} and m_{C2} (the thick, dark blue arrows in the figure). The backward message β_2 corresponds to the message m_{B2} (the thick, light orange arrow in the figure).

If G is not a tree, the message updates (3.25) are no longer guaranteed to return the exact marginals, nor are they guaranteed even to converge, but we can still iterate them in an attempt to find a fixed point. This procedure is called *loopy belief propagation*. To emphasize the approximate nature of this procedure, we refer to the approximate marginals that result from loopy BP as *beliefs* rather than as marginals, and denote them by $q(y_s)$.

Surprisingly, loopy BP can be seen as a variational method for inference, meaning that there actually exists an objective function over beliefs that is approximately minimized by the iterative BP procedure. Several introductory papers [137, 131] describe this in more detail.

The general idea behind a variational algorithm is:

(1) Define a family of tractable distributions $\mathcal{Q}$ and an objective function $\mathcal{O}(q)$. The function $\mathcal{O}$ should be designed to measure how well a tractable distribution $q \in \mathcal{Q}$ approximates the distribution p of interest.
(2) Find the "closest" tractable distribution $q^* = \min_{q \in \mathcal{Q}} \mathcal{O}(q)$.
(3) Use the marginals of q^* to approximate those of p.

For example, suppose that we take $\mathcal{Q}$ be the set of all possible distributions over $\mathbf{y}$, and we choose the objective function

$$\mathcal{O}(q) = \mathrm{KL}(q \| p) - \log Z \tag{3.29}$$

$$= -H(q) - \sum_a q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a). \tag{3.30}$$

Then the solution to this variational problem is $q^* = p$ with optimal value $\mathcal{O}(q^*) = \log Z$. Solving this particular variational formulation is

thus equivalent to performing exact inference. Approximate inference techniques can be devised by changing the set $\mathcal{Q}$—for example, by requiring q to be fully factorized—or by using a different objective $\mathcal{O}$. For example, the mean field method arises by requiring q to be fully factorized, i.e., $q(\mathbf{y}) = \prod_s q_s(y_s)$ for some choice for q_s, and finding the factorized q that most closely matches p.

With that background on variational methods, let us see how belief propagation can be understood in this framework. We make two approximations. First, we approximate the entropy term $H(q)$ of (3.30), which as it stands is difficult to compute. If q were a tree-structured distribution, then its entropy could be written exactly as

$$H_{\text{Bethe}}(q) = \sum_a q(\mathbf{y}_a) \log q(\mathbf{y}_a) + \sum_i (1 - d_i) q(y_i) \log q(y_i). \tag{3.31}$$

This follows from substituting the junction-tree formulation (3.27) of the joint into the definition of entropy. If q is not a tree, then we can still take H_{Bethe} as an approximation to H to compute the exact variational objective $\mathcal{O}$. This yields the *Bethe free energy*:

$$\mathcal{O}_{\text{Bethe}}(q) = H_{\text{Bethe}}(q) - \sum_a q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a) \tag{3.32}$$

The objective $\mathcal{O}_{\text{Bethe}}$ depends on q only through its marginals, so rather than optimizing it over all probability distributions q, we can optimize over the space of all marginal vectors. Specifically, every distribution q has an associated *belief vector* $\mathbf{q}$, with elements $q_{a;y_a}$ for each factor a and assignment y_a, and elements $q_{i;y_i}$ for each variable i and assignment y_i. The space of all possible belief vectors has been called the *marginal polytope* [130]. However, for intractable models, the marginal polytope can have extremely complex structure.

This leads us to the second variational approximation made by loopy BP, namely that the objective $\mathcal{O}_{\text{Bethe}}$ is optimized instead over a relaxation of the marginal polytope. The relaxation is to require that the beliefs be only *locally consistent*, that is, that

$$\sum_{\mathbf{y}_a \setminus y_i} q_a(\mathbf{y}_a) = q_i(y_i) \qquad \forall a, i \in a \tag{3.33}$$

Under these constraints, Yedidia et al. [136] show that constrained stationary points of $\mathcal{O}_{\text{Bethe}}$ fixed points of loopy BP. So we can view the Bethe energy $\mathcal{O}_{\text{Bethe}}$ as an objective function that the loopy BP fixed-point operations attempt to optimize.

This variational perspective provides new insight into the method that would not be available if we thought of it solely from the message passing perspective. One of the most important insights is that it shows how to use loopy BP to approximate $\log Z$. Because we introduced $\min_q \mathcal{O}_{\text{Bethe}}(q)$ as an approximation to $\min_q \mathcal{O}(q)$, and we know that $\min_q \mathcal{O}(q) = \log Z$, then it seems reasonable to define $\log Z_{\text{Bethe}} = \min_q \mathcal{O}_{\text{Bethe}}(q)$ as an approximation to $\log Z$. This will be important when we discuss CRF parameter estimation using BP in Section 4.4.2.

3.3 Implementation Concerns

In this section, we mention a few implementation techniques that are important to practical inference in CRFs: sparsity and preventing numerical underflow.

First, it is often possible to exploit *sparsity* in the model to make inference more efficient. Two different types of sparsity are relevant: sparsity in the factor values, and sparsity in the features. First, about the factor values, recall that in the linear-chain case, each of the forward updates (3.6) and backward updates (3.9) requires $O(M^2)$ time, that is, quadratic time in the number of labels. Analogously, in general CRFs, an update of loopy BP in a model with pairwise factors requires $O(M^2)$ time. In some models, however, it is possible to implement inference more efficiently, because it is known a priori not all factor values (y_t, y_{t-1}) are feasible, that is, the factor $\Psi_t(y_t, y_{t+1}, \mathbf{x}_t)$ is 0 for many values y_t, y_{t+1}. In such cases, the computational cost of sending a message can be reduced by implementing the message-passing iterations using sparse matrix operations.

The second kind of sparsity that is useful is sparsity in the feature vectors. Recall from (2.20) that computing the factors $\Psi_c(\mathbf{x}_c, \mathbf{y}_c)$ requires computing a dot product between the parameter vector θ_p and and the vector of features $F_c = \{f_{pk}(y_c, \mathbf{x}_c)\}$. Often, many elements

of the vectors F_c are zero. For example, natural language applications often involve binary indicator variables on word identity. In this case, the time required to compute the factors Ψ_c can be greatly improved using a sparse vector representation. In a similar fashion, we can use sparsity improve the time required to compute the likelihood gradient, as we discuss in Chapter 4.

A related trick, that will also speed up forward backward, is to tie the parameters for certain subsets of transitions [20]. This has the effect of reducing the effective size of the model's transition matrix, lessening the effect of the quadratic dependence of the size of the label set.

A second implementation concern that arises in inference is avoiding numerical underflow. The probabilities involved in forward-backward and belief propagation are often too small to be represented within numerical precision (for example, in an HMM they decay toward 0 exponentially fast in T). There are two standard approaches to this common problem. One approach is to scale each of the vectors α_t and β_t to sum to 1, thereby magnifying small values. This scaling does not affect our ability to compute $Z(\mathbf{x})$ because it can be computed as $Z(\mathbf{x}) = p(\mathbf{y}'|\mathbf{x})^{-1} \prod_t (\Psi_t(y'_t, y'_{t+1}, \mathbf{x}_t))$ for an arbitrary assignment $\mathbf{y}'$, where $p(\mathbf{y}'|\mathbf{x})^{-1}$ is computed from the marginals using (3.27). But in fact, there is actually a more efficient method described by Rabiner [96] that involves saving each of the local scaling factors. In any case, the scaling trick can be used in forward-backward or loopy BP; in either case, it does not affect the final values of the beliefs.

A second approach to preventing underflow is to perform computations in the logarithmic domain, e.g., the forward recursion (3.6) becomes

$$\log \alpha_t(j) = \bigoplus_{i \in S} \big(\log \Psi_t(j, i, x_t) + \log \alpha_{t-1}(i) \big), \tag{3.34}$$

where $\oplus$ is the operator $a \oplus b = \log(e^a + e^b)$. At first, this does not seem much of an improvement, since numerical precision is lost when computing e^a and e^b. But $\oplus$ can be computed as

$$a \oplus b = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b}), \tag{3.35}$$

which can be much more numerically stable, particularly if we pick the version of the identity with the smaller exponent.

At first, it would seem that the normalization approach is preferable to the logarithmic approach, because the logarithmic approach requires $O(TM^2)$ calls to the special functions log and exp, which can be computationally expensive. This observation is correct for HMMs, but not for CRFs. In a CRF, even when the normalization approach is used, it is still necessary to call the exp function in order to compute $\Psi_t(y_t, y_{t+1}, \mathbf{x}_t)$, defined in (3.17). So in CRFs, special functions cannot be avoided. In the worst case, there are TM^2 of these Ψ_t values, so the normalization approach needs TM^2 calls to special functions just as the logarithmic domain approach does. However, there are some special cases in which the normalization approach can yield a speedup, such as when the transition features do not depend on the observations, so that there are only M^2 distinct Ψ_t values.

4

Parameter Estimation

In this chapter we discuss how to estimate the parameters $\theta = \{\theta_k\}$ of a conditional random field. In the simplest and typical case, we are provided with fully labeled independent data, but there has also been work in CRFs with latent variables and CRFs for relational learning.

CRFs are trained by *maximum likelihood*, that is, the parameters are chosen such that the training data has highest probability under the model. In principle, this can be done in a manner exactly analogous to logistic regression, which should not be surprising given the close relationship between these models that was described in Chapter 2. The main difference is computational: CRFs tend to have more parameters and more complex structure than a simple classifier, so training is correspondingly more expensive.

In tree structured CRFs, the maximum likelihood parameters can be found by a numerical optimization procedure that calls the inference algorithms of Section 3.1 as a subroutine. Crucially, the likelihood is a convex function of the parameters, which means that powerful optimization procedures are available that provably converge to the optimal solution. For general CRFs, on the other hand, maximum likelihood training is intractable. One way to deal with this problem is

to use approximate inference methods, as discussed in Chapter 3, but another way is to choose a different training criterion than maximum likelihood.

We begin by describing maximum likelihood training, both in the linear chain case (Section 4.1.1) and in the case of general graphical structures (Section 4.1.2), including the case of latent variables. Then we discuss training in general graphical structures, in which approximations are necessary. We also describe two general methods for speeding up parameter estimation that exploit iid structure in the data: stochastic gradient descent (Section 4.2) and multithreaded training (Section 4.3). In CRFs with general structure, typically approximate inference procedures must be used. The approximate training procedures build on the approximate algorithms for inference described in Chapter 3, but there can be complications in the interaction between approximate inference and learning. This is described in Section 4.4.

4.1 Maximum Likelihood

4.1.1 Linear-chain CRFs

In a linear-chain CRF, the maximum likelihood parameters can be determined using numerical optimization methods. We are given iid training data $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$, where each $\mathbf{x}^{(i)} = \{\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \ldots \mathbf{x}_T^{(i)}\}$ is a sequence of inputs, and each $\mathbf{y}^{(i)} = \{y_1^{(i)}, y_2^{(i)}, \ldots y_T^{(i)}\}$ is a sequence of the desired predictions.

Parameter estimation is typically performed by penalized maximum likelihood. Because we are modeling the conditional distribution, the following log likelihood, sometimes called the *conditional log likelihood*, is appropriate:

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}). \tag{4.1}$$

One way to understand the conditional likelihood $p(\mathbf{y}|\mathbf{x};\theta)$ is to imagine combining it with some arbitrary prior $p(\mathbf{x};\theta')$ to form a joint $p(\mathbf{y},\mathbf{x})$. Then when we optimize the joint log likelihood

$$\log p(\mathbf{y},\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x};\theta) + \log p(\mathbf{x};\theta'), \tag{4.2}$$

the two terms on the right-hand side are decoupled, that is, the value of θ' does not affect the optimization over θ. If we do not need to estimate $p(\mathbf{x})$, then we can simply drop the second term, which leaves (4.1).

After substituting in the CRF model (2.16) into the likelihood (4.1), we get the following expression:

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}), \tag{4.3}$$

It is often the case that we have a large number of parameters, e.g., several hundred thousand. As a measure to avoid overfitting, we use *regularization*, which is a penalty on weight vectors whose norm is too large. A common choice of penalty is based on the Euclidean norm of θ and on a *regularization parameter* $1/2\sigma^2$ that determines the strength of the penalty. Then the regularized log likelihood is

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\theta_k^2}{2\sigma^2}. \tag{4.4}$$

The parameter σ^2 is a free parameter which determines how much to penalize large weights. Intuitively, the idea is to reduce the potential for a small number of features to dominate the prediction. The notation for the regularizer is intended to suggest that regularization can also be viewed as performing maximum a posteriori (MAP) estimation of θ, if θ is assigned a Gaussian prior with mean 0 and covariance $\sigma^2 I$. Determining the best regularization parameter can require a computationally-intensive parameter sweep. Fortunately, often the accuracy of the final model is not sensitive to small changes in σ^2 (e.g., up to a factor of 10). The best value of σ^2 depends on the size of the training set; for medium-sized training sets, $\sigma^2 = 10$ is typical.

An alternative choice of regularization is to use the L_1 norm instead of the Euclidean norm, which corresponds to an exponential prior on parameters [37]. This results in the following penalized likelihood:

$$\ell'(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \alpha \sum_{k=1}^{K} |\theta_k|. \tag{4.5}$$

This regularizer tends to encourage sparsity in the learned parameters, meaning that most of the θ_k are 0. This can be useful for performing feature selection, and also has theoretical advantages [84]. In practice, models trained with the L_1 regularizer tend to be sparser but have roughly the same accuracy as models training using the L_2 regularizer [56]. A disadvantage of the L_1 regularizer is that it is not differentiable at 0, which complicates numerical parameter estimation somewhat [37, 3, 138].

In general, the function $\ell(\theta)$ cannot be maximized in closed form, so numerical optimization is used. The partial derivatives of (4.4) are

$$\frac{\partial \ell}{\partial \theta_k} = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y' | \mathbf{x}^{(i)}) - \frac{\theta_k}{\sigma^2}. \tag{4.6}$$

The first term is the expected value of f_k under the empirical distribution:

$$\tilde{p}(\mathbf{y}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}} \mathbf{1}_{\{\mathbf{x}=\mathbf{x}^{(i)}\}}. \tag{4.7}$$

The second term, which arises from the derivative of $\log Z(\mathbf{x})$, is the expectation of f_k under the model distribution $p(\mathbf{y}|\mathbf{x};\theta)\tilde{p}(\mathbf{x})$. Therefore, at the unregularized maximum likelihood solution, when the gradient is zero, these two expectations are equal. This pleasing interpretation is a standard result about maximum likelihood estimation in exponential families.

To compute the likelihood $\ell(\theta)$ and its derivative requires techniques from inference in graphical models. In the likelihood, inference is needed to compute the partition function $Z(\mathbf{x}^{(i)})$, which is a sum over all possible labellings. In the derivatives, inference is required to compute the marginal distributions $p(y, y'|\mathbf{x}^{(i)})$. Because both of these quantities depend on $\mathbf{x}^{(i)}$, we will need to run inference once for each training instance every time the likelihood is computed. This is the key computational difference between CRFs and generative Markov random fields. In linear-chain models, inference can be performed efficiently using the algorithms described in Section 3.1.

Now we discuss how to optimize $\ell(\theta)$. The function $\ell(\theta)$ is concave, which follows from the convexity of functions of the form $g(\mathbf{x}) =$

$\log \sum_i \exp x_i$. Convexity is extremely helpful for parameter estimation, because it means that every local optimum is also a global optimum. Adding regularization ensures that ℓ is strictly concave, which implies that it has exactly one global optimum.

Perhaps the simplest approach to optimize ℓ is steepest ascent along the gradient (4.6), but this requires too many iterations to be practical. Newton's method converges much faster because it takes into account the curvature of the likelihood, but it requires computing the Hessian, the matrix of all second derivatives. The size of the Hessian is quadratic in the number of parameters. Since practical applications often use tens of thousands or even millions of parameters, simply storing the full Hessian is not practical.

Instead, current techniques for optimizing (4.4) make approximate use of second-order information. Particularly successful have been quasi-Newton methods such as BFGS [6], which compute an approximation to the Hessian from only the first derivative of the objective function. A full $K \times K$ approximation to the Hessian still requires quadratic size, however, so a limited-memory version of BFGS is used, due to Byrd et al. [14]. Conjugate gradient is another optimization technique that also makes approximate use of second-order information and has been used successfully with CRFs. For a good introduction to both limited-memory BFGS and conjugate gradient, see Nocedal and Wright [87]. Either can be thought of as a black-box optimization routine that is a drop-in replacement for vanilla gradient ascent. When such second-order methods are used, gradient-based optimization is much faster than the original approaches based on iterative scaling in Lafferty et al. [54], as shown experimentally by several authors [108, 132, 68, 79]. Finally, trust region methods have recently been shown to perform well on multinomial logistic regression [63], and may work well for CRFs as well.

Finally, we discuss the computational cost of training linear chain models. As we will see in Section 3.1, the likelihood and gradient for a single training instance can be computed by forward-backward in time $O(TM^2)$, where M is the number of labels and T the length of the training instance. Because we need to run forward-backward for

each training instance, each computation of the likelihood and gradient requires $O(TM^2N)$ time, so that the total cost of training is $O(TM^2NG)$, where G the number of gradient computations required by the optimization procedure. Unfortunately, G depends on the data set and is difficult to predict in advance. For batch L-BFGS on linear-chain CRFs, it is often but not always under 100. For many data sets, this cost is reasonable, but if the number of states M is large, or the number of training sequences N is very large, then this can become expensive. Depending on the number of labels, training CRFs can take anywhere from a few minutes to a few days; see Section 4.5 for examples.

4.1.2 General CRFs

Parameter estimation for general CRFs is essentially the same as for linear-chains, except that computing the model expectations requires more general inference algorithms. First, we discuss the fully-observed case, in which the training and testing data are independent, and the training data is fully observed. In this case the conditional log likelihood, using the notation of Section 2.4, is

$$\ell(\theta) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \theta_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x}). \tag{4.8}$$

The equations in this section do not explicitly sum over training instances, because if a particular application happens to have iid training instances, they can be represented by disconnected components in the graph G.

The partial derivative of the log likelihood with respect to a parameter θ_{pk} associated with a clique template C_p is

$$\frac{\partial \ell}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) p(\mathbf{y}'_c | \mathbf{x}). \tag{4.9}$$

The function $\ell(\theta)$ has many of the same properties as in the linear-chain case. First, the zero-gradient conditions can be interpreted as requiring that the sufficient statistics $F_{pk}(\mathbf{x}, \mathbf{y}) = \sum_{\Psi_c} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)$ have the same

expectations under the empirical distribution and under the model distribution. Second, the function $\ell(\theta)$ is concave, and can be efficiently maximized by second-order techniques such as conjugate gradient and L-BFGS. Finally, regularization is used just as in the linear-chain case.

All of the discussion so far has assumed that the training data contains the true values of all the label variables in the model. In the latent variable case, on the other hand, the model contains variables that are observed at neither training nor test time. This situation is called a *hidden-state CRF (HCRF)* by Quattoni et al. [95] which was one of the first examples of latent variable CRFs. Quattoni et al. [94] present a more detailed description. For other early applications of HCRFs, see [120, 75]. It is more difficult to train CRFs with latent variables because the latent variables need to be marginalized out to compute the likelihood. Because of this difficultly, the original work on CRFs focused on fully-observed training data, but recently there has been increasing interest in HCRFs.

Suppose we have a conditional random field with inputs $\mathbf{x}$ in which the output variables $\mathbf{y}$ are observed in the training data, but we have additional variables $\mathbf{w}$ that are latent, so that the CRF has the form

$$p(\mathbf{y}, \mathbf{w}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{4.10}$$

A natural objective function to maximize during training is the marginal likelihood

$$\ell(\theta) = \log p(\mathbf{y}|\mathbf{x}) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x}). \tag{4.11}$$

The first question is how even to compute the marginal likelihood $\ell(\theta)$, because if there are many variables $\mathbf{w}$, the sum cannot be computed directly. The key is to realize that we need to compute $\log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})$ not for any possible assignment $\mathbf{y}$, but only for the particular assignment that occurs in the training data. This motivates taking the original CRF (4.10), and clamping the variables Y to their observed values in the training data, yielding a distribution over $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}) = \frac{1}{Z(\mathbf{y}, \mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p), \tag{4.12}$$

where the normalization factor is

$$Z(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{4.13}$$

This new normalization constant $Z(\mathbf{y}, \mathbf{x})$ can be computed by the same inference algorithm that we use to compute $Z(\mathbf{x})$. In fact, $Z(\mathbf{y}, \mathbf{x})$ is easier to compute, because it sums only over $\mathbf{w}$, while $Z(\mathbf{x})$ sums over both $\mathbf{w}$ and $\mathbf{y}$. Graphically, this amounts to saying that clamping the variables $\mathbf{y}$ in the graph G can simplify the structure among $\mathbf{w}$.

Once we have $Z(\mathbf{y}, \mathbf{x})$, the marginal likelihood can be computed as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p) = \frac{Z(\mathbf{y}, \mathbf{x})}{Z(\mathbf{x})}. \tag{4.14}$$

Now that we have a way to compute ℓ, we discuss how to maximize it with respect to θ. Maximizing $\ell(\theta)$ can be difficult because ℓ is no longer convex in general (log-sum-exp is convex, but the difference of two log-sum-exp functions might not be), so optimization procedures are typically guaranteed to find only local maxima. Whatever optimization technique is used, the model parameters must be carefully initialized in order to reach a good local maximum.

We discuss two different ways to maximize ℓ: directly using the gradient, as in Quattoni et al. [95]; and using EM, as in McCallum et al. [75]. (In addition, it is also natural to use stochastic gradient descent here; see Section 4.2.) To maximize ℓ directly, we need to calculate its gradient. The simplest way to do this is to use the following fact. For any function $f(\theta)$, we have

$$\frac{df}{d\theta} = f(\theta) \frac{d \log f}{d\theta}, \tag{4.15}$$

which can be seen by applying the chain rule to $\log f$ and rearranging. Applying this to the marginal likelihood $\ell(\theta) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})$ yields

$$\frac{\partial \ell}{\partial \theta_{pk}} = \frac{1}{\sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})} \sum_{\mathbf{w}} \frac{\partial}{\partial \theta_{pk}} \big[p(\mathbf{y}, \mathbf{w}|\mathbf{x}) \big] \tag{4.16}$$

$$= \sum_{\mathbf{w}} p(\mathbf{w}|\mathbf{y}, \mathbf{x}) \frac{\partial}{\partial \theta_{pk}} \big[\log p(\mathbf{y}, \mathbf{w}|\mathbf{x}) \big]. \tag{4.17}$$

This is the expectation of the fully-observed gradient, where the expectation is taken over $\mathbf{w}$. This expression simplifies to

$$\frac{\partial \ell}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c} p(\mathbf{w}'_c|\mathbf{y}, \mathbf{x}) f_k(\mathbf{y}_c, \mathbf{x}_c, \mathbf{w}'_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c, \mathbf{y}'_c} p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c) f_k(\mathbf{y}'_c, \mathbf{x}_c, \mathbf{w}'_c). \quad (4.18)$$

This gradient requires computing two different kinds of marginal probabilities. The first term contains a marginal probability $p(\mathbf{w}'_c|\mathbf{y}, \mathbf{x})$, which is exactly a marginal distribution of the clamped CRF (4.12). The second term contains a different marginal $p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c)$, which is the same marginal probability required in a fully-observed CRF. Once we have computed the gradient, ℓ can be maximized by standard techniques such as conjugate gradient. For BFGS, it has been our experience that the memory-based approximation to the Hessian can become confused by violations of convexity, such as occur in latent-variable CRFs. One practical trick in this situation is to reset the Hessian approximation when that happens.

Alternatively, ℓ can be optimized using expectation maximization (EM). At each iteration j in the EM algorithm, the current parameter vector $\theta^{(j)}$ is updated as follows. First, in the E-step, an auxiliary function $q(\mathbf{w})$ is computed as $q(\mathbf{w}) = p(\mathbf{w}|\mathbf{y}, \mathbf{x}; \theta^{(j)})$. Second, in the M-step, a new parameter vector $\theta^{(j+1)}$ is chosen as

$$\theta^{(j+1)} = \arg\max_{\theta'} \sum_{\mathbf{w}'} q(\mathbf{w}') \log p(\mathbf{y}, \mathbf{w}'|\mathbf{x}; \theta'). \quad (4.19)$$

The direct maximization algorithm and the EM algorithm are strikingly similar. This can be seen by substituting the definition of q into (4.19) and taking derivatives. The gradient is almost identical to the direct gradient (4.18). The only difference is that in EM, the distribution $p(\mathbf{w}|\mathbf{y}, \mathbf{x})$ is obtained from a previous, fixed parameter setting rather than from the argument of the maximization. We are unaware of any empirical comparison of EM to direct optimization for latent-variable CRFs.

4.2 Stochastic Gradient Methods

So far, all of the methods that we have discussed for optimizing the likelihood work in a *batch setting*, meaning that they do not make any change to the model parameters until they have scanned the entire training set. If the training data consist of a large number of iid samples, then this may seem wasteful. We may suspect that many different items in the training data provide similar information about the model parameters, so that it should be possible to update the parameters after seeing only a few examples, rather than sweeping through all of them.

Stochastic gradient descent (SGD) is a simple optimization method that is designed to exploit this insight. The basic idea is at every iteration, to pick a training instance at random, and take a small step in the direction given by the gradient for that instance only. In the batch setting, gradient descent is generally a poor optimization method, because the direction of steepest descent locally (that is, the negative gradient) can point in a very different direction than the optimum. So stochastic gradient methods involve an interesting tradeoff: the directions of the individual steps may be much better in L-BFGS than in SGD, but the SGD directions can be computed much faster.

In order to keep the notation simple, we present SGD only for the case of linear-chain CRFs, but it can be easily used with any graphical structure, as long as the training data are iid. The gradient of the likelihood for a single training instance $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ is

$$\frac{\partial \ell_i}{\partial \theta_k} = \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y' | \mathbf{x}^{(i)}) - \frac{\theta_k}{N\sigma^2}. \tag{4.20}$$

This is exactly the same as the full gradient (4.6), with two changes: the sum over training instances has been removed, and the regularization contains an additional factor of $1/N$. These ensure that the batch gradient equals the sum of the per-instance gradients, i.e., $\nabla \ell = \sum_{i=1}^{N} \nabla \ell_i$, where we use $\nabla \ell_i$ to denote the gradient for instance i.

At each iteration m of SGD, we randomly select a training instance $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$. Then compute the new parameter vector $\theta^{(m)}$ from the old

vector $\theta^{(m)}$ by

$$\theta^{(m)} = \theta^{(m-1)} - \alpha_m \nabla \ell_i(\theta^{(m-1)}), \tag{4.21}$$

where $\alpha_m > 0$ is a step size parameter that controls how far the parameters move in the direction of the gradient. If the step size is too large, then the parameters will swing too far in the direction of whatever training instance is sampled at each iteration. If α_m is too small, then training will proceed very slowly, to the extent that in extreme cases, the parameters may appear to have converged numerically when in fact they are far from the minimum.

We want α_m to decrease as m increases, so that the optimization algorithm converges to a single answer. The most common way to do this is to select a step size schedule of a form like $\alpha_m \sim 1/m$ or $\alpha_m \sim 1/\sqrt{m}$. These choices are motivated by the classic convergence results for stochastic approximation procedures [100, 47]. However, simply taking $\alpha_m = 1/m$ is usually bad, because then the first few step sizes are too large. Instead, a common trick is to use a schedule like

$$\alpha_m = \frac{1}{\sigma^2(m_0 + m)}, \tag{4.22}$$

where m_0 is a free parameter that needs to be set. A suggestion for setting this parameter, due to Leon Bottou [11], is to sample a small subset of the training data and run one pass of SGD over the subset with various fixed step sizes α. Pick the α^* such that the resulting likelihood on the subset after one pass is highest, and choose m_0 such that $\alpha_0 = \alpha^*$.

Stochastic gradient descent has also gone by the name of backpropagation in the neural network literature, and many tricks for tuning the method have been developed over the years [57]. Recently, there has been renewed interest in advanced online optimization methods [128, 24, 109, 36], which also update parameters in an online fashion, but in a more sophisticated way than simple SGD. Vishwanathan et al. [128] was the first application of stochastic gradient methods to CRFs.

The main disadvantage of stochastic gradient methods is that they do require tuning, unlike off-the-shelf solvers such as conjugate gradient and L-BFGS. Stochastic gradient methods are also not useful in relational settings in which the training data are not iid, or on small data

sets. On appropriate data sets, however, stochastic gradient methods can offer considerable speedups.

4.3 Parallelism

Stochastic gradient descent speeds up the gradient computation by computing it over fewer instances. An alternative way to speed up the gradient computation is to compute the gradient over multiple instances in parallel. Because the gradient (4.6) is a sum over training instances, it is easy to divide the computation into multiple threads, where each thread computes the gradient on a subset of training instances. If the CRF implementation is run on a multicore machine, then the threads will run in parallel, greatly speeding up the gradient computation. This is a characteristic shared by many common machine learning algorithms, as pointed out by Chu et al. [18].

In principle, one could also distribute the gradient computation across multiple machines, rather than multiple cores of the same machine, but the overhead involved in transferring large parameter vectors across the network can be an issue. A potentially promising way to avoid this is to update the parameter vectors asynchronously. An example of this idea is recent work on incorporating parallel computation into stochastic gradient methods [55].

4.4 Approximate Training

All of the training methods that we have described so far, including the stochastic and parallel gradient methods, assume that the graphical structure of the CRF is tractable, that is, that we can efficiently compute the partition function $Z(\mathbf{x})$ and the marginal distributions $p(\mathbf{y}_c|\mathbf{x})$. This is the case, for example, in linear chain and tree-structured CRFs. Early work on CRFs focused on these cases, both because of the tractability of inference, and because this choice is very natural for certain tasks such as sequence labeling tasks in NLP.

But more complex graphs are important in domains such as computer vision, where grid-structured graphs are natural, and for more global models of natural language [114, 30, 13]. When the graphical

structure is more complex, then the marginal distributions and the partition function cannot be computed tractably, and we must resort to approximations. As described in Chapter 3, there is a large literature on approximate inference algorithms. In the context of CRFs, however, there is a crucial additional consideration, which is that the approximate inference procedure is embedded within a larger optimization procedure for selecting the parameters.

There are two general ways to think about approximate training in CRFs [118]: One can either modify the likelihood, or approximate the marginal distributions directly. Modifying the likelihood typically means finding some substitute for $\ell(\theta)$ (such as the BP approximation (4.27)), which we will call a *surrogate likelihood* that is easier to compute but is still expected to favor good parameter setting. Then the surrogate likelihood can be optimized using a gradient-based method, in a similar way to the exact likelihood. Approximating the marginal distributions means using a generic inference algorithm to compute an approximation to the marginals $p(\mathbf{y}_c|\mathbf{x})$, substituting the approximate marginals for the exact marginals in the gradient (4.9), and performing some kind of gradient descent procedure using the resulting approximate gradients.

Although surrogate likelihood and approximate marginal methods are obviously closely related, they are distinct. Usually an surrogate likelihood method directly yields an approximate marginals method, because just as the derivatives of $\log Z(\mathbf{x})$ give the true marginal distributions, the derivatives of an approximation to $\log Z(\mathbf{x})$ can be viewed as an approximation to the marginal distributions. These approximate marginals are sometimes termed *pseudomarginals* [129]. However, the reverse direction does not always hold: for example, there are certain approximate marginal procedures that provably do not correspond to the derivative of any likelihood function [118, 112].

The main advantage of a surrogate likelihood method is that having an objective function can make it easier to understand the properties of the method, both to human analysts and to the optimization procedure. Advanced optimization engines such as conjugate gradient and BFGS require an objective function in order to operate. The advantage to the approximate marginals viewpoint, on the other hand, is that it is more

flexible. It is easy to incorporate arbitrary inference algorithms, including tricks such as early stopping of BP and MCMC. Also, approximate marginal methods fit well within a stochastic gradient framework.

There are aspects of the interaction between approximate inference and parameter estimation that are not completely understood. For example, Kulesza and Pereira [52] present an example of a situation in which the perceptron algorithm interacts in a pathological fashion with max-product belief propagation. Surrogate likelihood methods, by contrast, do not seem to display this sort of pathology, as Wainwright [129] point out for the case of convex surrogate likelihoods.

To make this discussion more concrete, in the rest of this section, we will discuss several examples of surrogate likelihood and approximate marginal methods. We discuss surrogate likelihood methods based on pseudolikelihood (Section 4.4.1) and belief propagation (Section 4.4.2) and approximate gradient methods based on belief propagation (Section 4.4.2) and MCMC (Section 4.4.3).

4.4.1 Pseudolikelihood

One of the earliest surrogate likelihoods is the pseudolikelihood [8]. The idea in pseudolikelihood is for the training objective to depend only on conditional distributions over single variables. Because the normalizing constants for these distributions depend only on single variables, they can be computed efficiently. In the context of CRFs, the pseudolikelihood is

$$\ell_{\text{PL}}(\theta) = \sum_{s \in V} \log p(y_s | \mathbf{y}_{N(s)}, \mathbf{x}; \theta) \tag{4.23}$$

Here the summation over s ranges over all output nodes in the graph, and $\mathbf{y}_{N(s)}$ are the values of the variables $N(s)$ that are neighbors of s. (As in (4.8), we do not include the sum over training instances explicitly.)

Intuitively, one way to understand pseudolikelihood is that it attempts to match the local conditional distributions $p(y_s | \mathbf{y}_{N(s)}, \mathbf{x}; \theta)$ according to the model to those of the training data, and because of the conditional independence assumptions of the model, the local conditional distributions are sufficient to specify the joint. (This is similar

to the motivation behind a Gibbs sampler.)

The parameters are estimated by maximizing the pseudolikelihood, i.e., the estimates are $\hat{\theta}_{\text{PL}} = \max_\theta \ell_{\text{PL}}(\theta)$. Typically, the maximization is carried out by a second order method such as limited-memory BFGS, but in principle parallel computation or stochastic gradient can be applied to the pseudolikelihood exactly in the same way as the full likelihood. Also, regularization can be used just as with maximum likelihood.

The motivation behind pseudolikelihood is computational efficiency. The pseudolikelihood can be computed and optimized without needing to compute $Z(\mathbf{x})$ or the marginal distributions. Although pseudolikelihood has sometimes proved effective in NLP [126], more commonly the performance of pseudolikelihood is poor [115], in an intuitively analogous way that a Gibbs sampler can mix slowly in sequential models. One can obtain better performance by performing a "blockwise" version of pseudolikelihood in which the local terms involve conditional probabilities of larger regions in the model. For example, in a linear-chain CRF, one could consider a per-edge pseudolikelihood:

$$\ell_{\text{EPL}}(\theta) = \sum_{t=1}^{T-1} \log p(y_t, y_{t+1} | y_{t-1}, y_{t+2}, \theta) \tag{4.24}$$

(Here we assume that the sequence is padded with dummy labels y_0 and y_{T+1} so that the edge cases are correct.) This blockwise version of pseudolikelihood is a special case of composite likelihood [64, 29], for which there are general theoretical results concerning asymptotic consistency and normality. Typically larger blocks lead to better parameter estimates, both in theory and in practice.

4.4.2 Belief Propagation

The loopy belief propagation algorithm (Section 3.2.2) can be used within approximate CRF training. This can be done within either the surrogate likelihood or the approximate gradient perspectives.

In the approximate gradient algorithm, at every iteration of training, we run loopy BP on the training input $\mathbf{x}$, yielding a set of approximate marginals $q(\mathbf{y}_c)$ for each clique in the model. Then we approxi-

mate the true gradient (4.9) by substituting in the BP marginals. This results in approximate partial derivatives

$$\frac{\partial \tilde{\ell}}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) q(\mathbf{y}'_c). \tag{4.25}$$

These can be used to update the current parameter setting as

$$\theta_{pk}^{(t+1)} = \theta_{pk}^{(t)} + \alpha \frac{\partial \tilde{\ell}}{\partial \theta_{pk}} \tag{4.26}$$

where $\alpha > 0$ is a step size parameter. The advantages of this setup are that it is extremely simple, and is especially useful within an outer stochastic gradient approximation.

More interestingly, however, it is also possible to use loopy BP within a surrogate likelihood setup. To do this, we need to develop some surrogate function for the true likelihood (4.8) which has the property that the gradient of the surrogate likelihood are exactly the approximate BP gradients (4.26). This may seem like a tall order, but fortunately it is possible using the Bethe free energy described in Section 3.2.2.

Remember from that section that loopy belief propagation can be viewed as an optimization algorithm, namely, one that minimizes the objective function $\mathcal{O}_{\text{Bethe}}(q)$ (3.32) over the set of all locally consistent belief vectors, and that the minimizing value $\min_q \mathcal{O}_{\text{Bethe}}(q)$ can be used as an approximation to the partition function. Substituting in that approximation to the true likelihood (4.8) gives us, for a fixed belief vector q, the approximate likelihood

$$\ell_{\text{Bethe}}(\theta, q) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \log \Psi_c(\mathbf{x}_c, \mathbf{y}_c) - \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} q(\mathbf{y}_c) \log \frac{q(\mathbf{y}_c)}{\Psi_c(\mathbf{x}_c, \mathbf{y}_c)} + \sum_{s \in Y} (1 - d_i) q(y_s) \log q(y_s). \tag{4.27}$$

Then approximate training can be viewed as the optimization problem $\max_\theta \min_q \ell_{\text{Bethe}}(\theta, q)$. This is a *saddlepoint problem*, in which we are maximizing with respect to one variable (to find the best parameters) and minimizing with respect to another (to solve the approximate

inference problem). One approach to solve saddlepoint problems is coordinate ascent, that is, to alternately minimize ℓ_{Bethe} with respect to q for fixed θ and take a gradient step to partially maximize ℓ_{Bethe} with respect to θ for fixed b. The first step (minimizing with respect to q) is just running the loopy BP algorithm. The key point is that for the second step (maximizing with respect to θ), the partial derivatives of (4.27) with respect to a weight θ_k is exactly (4.26), as desired.

Alternatively, there is a different surrogate likelihood that can also be used. This is

$$\hat{\ell}(\theta; q) = \log \left[\frac{\prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} q(\mathbf{y}_c)}{\prod_{s \in Y} q(y_s)^{d_s - 1}} \right], \tag{4.28}$$

In other words, instead of the true joint likelihood, we use the product over each clique's approximate belief, dividing by the node beliefs to avoid overcounting. The nice thing about this is that it is a direct generalisation of the true likelihood for tree-structured models, as can be seen by comparing (4.28) with (3.27). This surrogate likelihood can be justified using a dual version of Bethe energy that we have presented here [78, 81]. When BP has converged, for the resulting belief vector q, it can be shown that $\ell_{\text{Bethe}}(\theta, q) = \hat{\ell}(\theta, q)$. This equivalence does not hold in general for arbitrary values of q, e.g., if BP has not converged.

Another surrogate likelihood method that is related to BP is the *piecewise* estimator [117], in which the factors of the model are partitioned into tractable subgraphs that are trained independently. This idea can work surprisingly well (better than pseudolikelihood) if the local features are sufficiently informative. Sutton and Minka [118] discuss the close relationship between piecewise training and early stopping of belief propagation.

4.4.3 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) inference methods (Section 3.2.1) can be used within CRF training by setting up a Markov chain whose stationary distribution is $p(\mathbf{y}|\mathbf{x}; \theta)$, running the chain for a number of iterations, and using the resulting approximate marginals $\hat{p}(\mathbf{y}|\mathbf{x}; \theta)$ to approximate the true marginals in the gradient (4.9).

In practice, however, MCMC methods are not commonly used in the context of CRFs. There are two main reasons for this. First, MCMC methods typically require many iterations to reach convergence, and as we have emphasized, inference needs to be run for many different parameter settings over the course of training. Second, many MCMC methods, such as Metropolis-Hastings, require computing a ratio of normalising constants $Z_{\theta_1}(\mathbf{x})/Z_{\theta_2}(\mathbf{x})$ for two different parameters settings θ_1 and θ_2. This presents a severe difficulty for models in which computing $Z_\theta(\mathbf{x})$ is intractable.

One possibility to overcome these difficulties is contrastive divergence (CD) [44], in which the true marginals $p(y_c|\mathbf{x})$ in (4.9) are approximated by running an MCMC method for only a few iterations, where the initial state of the Markov chain (which is just an assignment to $\mathbf{y}$) is set to be the value of $\mathbf{y}$ in the training data. CD has been mostly applied to latent variable models such as restricted Boltzmann machines, it can also be applied to CRFs. We are unaware of much work in this direction.

Another possibility is a more recent method called SampleRank [135], whose objective is that the learned parameters score pairs of $\mathbf{y}$s such that their sorted ranking obeys a given supervised ranking (which is often specified in terms of a fixed scoring function on $\mathbf{y}$ that compares to true target values of $\mathbf{y}$). Approximate gradients may be calculated from pairs of successive states of the MCMC sampler. Like CD, SampleRank learns very quickly because it performs useful parameter updates on many individual MCMC steps. Experiments have shown the structured classification accuracy from SampleRank to be substantially higher than CD [135].

The discussion above concerns MCMC methods within an approximate gradient framework. In contrast, it is very difficult to use an MCMC inference method within an surrogate likelihood framework, because it is notoriously difficult to obtain a good approximation to $\log Z(\mathbf{x})$ given samples from an MCMC method.

Task	Parameters	Predicates	# Sequences	# Positions	Labels	Time (s)
NP chunking	248471	116731	8936	211727	3	958s
NER	187540	119265	946	204567	9	4866s
POS tagging	509951	127764	38219	912344	45	325500s

Table 4.1 Scale of typical CRF applications in natural language processing

4.5 Implementation Concerns

To make the discussion of efficient training methods more concrete, here we give some examples of data sets from NLP in which CRFs have been successful. The idea is to give a sense of the scales of problem to which CRFs have been applied, and of typical values of the number of the numbers of features and of training times.

We describe three example tasks to which CRFs have been applied. The first example task is noun-phrase (NP) chunking [104], in which the problem is to find base noun phrases in text, such as the phrases "He" and "the current account deficit" in the sentence *He reckons the current account deficit will narrow*. The second task is named identity recognition (NER) [125], The final task is part-of-speech tagging (POS), that is, labelling each word in a sentence with its part of speech. The NP chunking and POS data sets are derived from the WSJ Penn Treebank [70], while the NER data set consists of newswire articles from Reuters.

We will not go into detail about the features that we use, but they include the identity of the current and previous word, prefixes and suffixes, and (for the named-entity and chunking tasks) automatically generated part of speech tags and lists of common places and person names. We do not claim that the feature sets that we have used are optimal for these tasks, but still they should be useful for getting a sense of scale.

For each of these data sets, Table 4.1 shows (a) the number of parameters in the trained CRF model, (b) the size of the training set, in terms of the total number of sequences and number of words, (c) the number of possible labels for each sequence position, and (d) the training time. The training times range from minutes in the best case to days in the worst case. As can be expected from our previous discussion,

the factor that seems to most influence training time is the number of labels.

Obviously the exact training time will depend heavily on details of the implementation and hardware. For the examples in Table 4.1, we use the MALLET toolkit on machines with a 2.4 GHz Intel Xeon CPU, optimizing the likelihood using batch L-BFGS without using multithreaded or stochastic gradient training.

5

Related Work and Future Directions

In this section, we briefly place CRFs in the context of related lines of research, especially that of *structured prediction*, a general research area which is concerned with extending classification methods to complex objects. We also describe relationships both to neural networks and to a simpler sequence model called maximum entropy Markov models (MEMMs). Finally, we outline a few open areas for future work.

5.1 Related Work

5.1.1 Structured Prediction

Conditional random fields provide one method for extending the ideas behind classification to the prediction of more complex objects such as sequences and trees. This general area of research is called *structured prediction*. Essentially, logistic regression is to a CRF as classification is to structured prediction. Examples of the types of structured outputs that are considered include parse trees of natural language sentences [123, 31], alignments between sentences in different languages [124], and route plans in mobile robotics [97]. Detailed information about structured prediction methods is available in a recent collection of research

papers [4].

Structured prediction methods are essentially a combination of classification and graphical modeling, combining the ability to compactly model multivariate data with the ability to perform prediction using large sets of input features. The idea is, for an input $\mathbf{x}$, to define a discriminant function $F_{\mathbf{x}}(\mathbf{y})$, and predict $\mathbf{y}^* = \arg\max_{\mathbf{y}} F_{\mathbf{x}}(\mathbf{y})$. This function factorizes according to a set of local factors, just as in graphical models. But as in classification, each local factor is modeled a linear function of $\mathbf{x}$, although perhaps in some induced high-dimensional space. To understand the benefits of this approach, consider a hidden Markov model (Section 2.2.2) and a set of per-position classifiers, both with fixed parameters. In principle, the per-position classifiers predict an output y_s given all of $\mathbf{x}_0 \dots \mathbf{x}_T$.[1] In the HMM, on the other hand, to predict y_s it is statistically sufficient to know only the local input $\mathbf{x}_s$, the previous forward message $p(y_{s-1}, \mathbf{x}_0 \dots \mathbf{x}_{s-1})$, and the backward message $p(\mathbf{x}_{s+1} \dots \mathbf{x}_T | y_s)$. So the forward and backward messages serve as a summary of the rest of the input, a summary that is generally nonlinear in the observed features.

In principle, the same effect could be achieved using a per-position classifier if it were possible to define an extremely flexible set of nonlinear features that depend on the entire input sequence. But as we have seen the size of the input vector is extremely large. For example, in part-of-speech tagging, each vector $\mathbf{x}_s$ may have tens of thousands of components, so a classifier based on all of $\mathbf{x}$ would have many parameters. But using only $\mathbf{x}_s$ to predict y_s is also bad, because information from neighboring feature vectors is also useful in making predictions. Essentially the effect of a structured prediction method is that a confident prediction about one variable is able to influence nearby, possibly less confident predictions.

Several types of structured prediction algorithms have been studied. All such algorithms assume that the discriminant function $F_{\mathbf{x}}(\mathbf{y})$ over labels can be written as a sum of local functions $F_{\mathbf{x}}(\mathbf{y}) = \sum_a f_a(\mathbf{y}_a, \mathbf{x}, \theta)$. The task is to estimate the real-valued parameter vec-

[1] To be fair, in practice the classifier for y_s would probably depend only on a sliding window around $\mathbf{x}_s$, rather than all of $\mathbf{x}$.

tor θ given a training set $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$. The methods differ in how the parameters are selected.

Alternative structured prediction methods are based on maximizing over assignments rather than marginalizing. Perhaps the most popular of these methods has been *maximum-margin* methods that are so successful for univariate classification. Maximum margin methods have been generalized to the structured case [2, 122]. Both batch and online algorithms have been developed to maximize this objective function. The perceptron update can also be generalized to structured models [21]. The resulting algorithm is particularly appealing because it is little more difficult to implement than the algorithm for selecting $\mathbf{y}^*$. The online perceptron update can also be made margin-aware, yielding the MIRA algorithm [23], which may perform better than the perceptron update.

Another class of methods are search-based methods [27, 28] in which a heuristic search procedure over outputs is assumed, and learns a classifier that predicts the next step in the search. This has the advantage of fitting in nicely to many problems that are complex enough to require performing search. It is also able to incorporate arbitrary loss functions over predictions.

A general advantage of all of these maximization-based methods is that they do not require summation over all configurations for the partition function or for marginal distributions. There are certain combinatorial problems, such as matching and network flow problems, in which finding an optimal configuration is tractable, but summing over configurations is not (for an example of applying max-margin methods in such situations, see Taskar et al. [124]). For more complex problems, neither summation nor maximization is tractable, so this advantage is perhaps not as significant. Another advantage of these methods is that kernels can be naturally incorporated, in an analogous way as in support vector machines.

Finally, LeCun et al. [59] generalizes many prediction methods, including the ones listed above, under the rubric of *energy-based* methods, and presents interesting historical information about their use. They advocate changing the loss function to avoid probabilities altogether.

Perhaps the main advantage of probabilistic methods is that they can incorporate latent variables in a natural way, by marginalization. This can be useful, for example, in collective classification methods [121]. For examples of structured models with latent variables, see Quattoni et al. [95] and McCallum et al. [75]. A particularly powerful example of this is provided by Bayesian methods, in which the model parameters themselves are integrated out (Section 5.2.1).

The differences between the various structured prediction methods are not well understood. To date, there has been little careful comparison of these, especially CRFs and max-margin approaches, across different structures and domains, although see Keerthi and Sundararajan [46] for some experiments in this regard.[2] We take the view that the similarities between various structured prediction methods are more important than the differences. Careful selection of features has more effect on performance than the choice of structured prediction algorithm.

5.1.2 Neural Networks

There are close relationships between neural networks and conditional random fields, in that both can be viewed as discriminatively trained probabilistic models. Neural networks are perhaps best known for their use in classification, but they can also be used to predict multiple outputs, for example, by using a shared latent representation [15], or by modelling dependencies between outputs directly [58]. Although neural networks are typically trained using stochastic gradient descent (Section 4.2), in principle they can be trained using any of the other methods used for CRFs. The main difference between them is that neural networks represent the dependence between output variables using a shared latent representation, while structured methods learn these dependences as direct functions of the output variables.

Because of this, it is easy to make the mistake of thinking that CRFs are convex and neural networks are not. This is incorrect. A neural network without a hidden layer is a linear classifier that can be trained

[2] An earlier study [86] appears to have been flawed. See Keerthi and Sundararajan [46] for discussion.

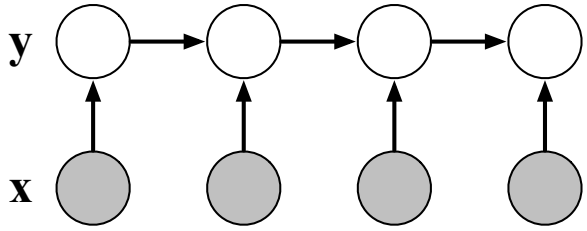

Fig. 5.1 Graphical model of a maximum entropy Markov model [74].

efficiently in a number of ways, while a CRF with latent variables has a complex non-convex likelihood (Section 2.4). The correct way of thinking is: In fully observed models, the likelihood is convex; in latent variable models it is not.

So the main new insight of structured prediction models compared to neural networks is: If you add connections among the nodes in the output layer, and if you have a good set of features, then sometimes you don't need a hidden layer to get good performance. If you can afford to leave out the hidden, then in practice you always want to do so, because this avoids all of the problems with local minima. For harder problems, however, one might expect that even after modeling output structure, incorporating hidden state will still yield additional benefit. Once hidden state is introduced into the model, whether it be a neural network or a structured model, it seems to be inevitable (at least given our current understanding of machine learning) that convexity will be lost.

5.1.3 MEMMs, Directed Models, and Label Bias

Linear-chain CRFs were originally introduced as an improvement to the *maximum-entropy Markov model* (MEMM) [74], which is essentially a Markov model in which the transition probabilities are given by logistic

regression. Formally, an MEMM is

$$p_{\text{MEMM}}(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1}, \mathbf{x}) \tag{5.1}$$

$$p(y_t|y_{t-1}, \mathbf{x}) = \frac{1}{Z_t(y_{t-1}, \mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\} \tag{5.2}$$

$$Z_t(y_{t-1}, \mathbf{x}) = \sum_{y'} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y', y_{t-1}, \mathbf{x}_t)\right\} \tag{5.3}$$

A similar idea can be extended to general directed graphs, in which the distribution $p(\mathbf{y}|\mathbf{x})$ is expressed by a Bayesian network in which each CPT is a logistic regression models with input $\mathbf{x}$ [102].

In the linear-chain case, notice that the MEMM works out to have the same form as the linear-chain CRF (4.3) with the exception that in a CRF $Z(\mathbf{x})$ is a sum over sequences, whereas in a MEMM the analogous term is $\prod_{t=1}^{T} Z_t(y_{t-1}, \mathbf{x})$. This difference has important consequences. Unlike in a CRFs, maximum likelihood training of MEMMs does not require performing inference, because Z_t is just a simple sum over the labels at a single position, rather than a sum over labels of an entire sequence. This is an example of the general phenomenon that training of directed models is less computationally demanding than undirected models.

There are theoretical difficulties with the MEMM model, however. MEMMs can exhibit the problems of label bias [54] and observation bias [48]. Originally, the label bias problem was described from an algorithmic perspective. Consider the backward recursion (3.9). In the case of an MEMM, this amounts to

$$\beta_t(i) = \sum_{j \in S} p(y_{t+1} = j | y_t = i, x_{t+1}) \beta_{t+1}(j). \tag{5.4}$$

Unfortunately, this sum is always 1, regardless of the value of the current label i. To see this, assume for the sake of induction that $\beta_{t+1}(j) = 1$ for all j. Then it is clear that the sum over j in (5.4) collapses, and $\beta_t(i) = 1$. What this means is that the future observations provide no information about the current state, which seems to lose a major advantage of sequence modelling.

Perhaps a more intuitive way to understand label bias is from the perspective of graphical models. Consider the graphical model of an MEMM, shown in Figure 5.1. By looking at the v-structures in the graph, we can read off the following independence assumptions: at all time steps t, the label y_t is marginally independent of the future observations $\mathbf{x}_{t+1}, \mathbf{x}_{t+2}$, etc. This independence assumption is usually strongly violated in sequence modeling, which explains why CRFs can have better performance than MEMMs. Also, this independence relation explains why $\beta_t(i)$ should always be 1. (In general, this correspondence between graph structure and inference algorithms is one of main conceptual advantages of graphical modelling.) To summarize this discussion, *label bias is simply a consequence of explaining away.*

There is a caveat here: We can always copy information from previous and future time steps into the feature vector $\mathbf{x}_t$, and this is common in practice. (The only constraint is that if we have too many features, then overfitting we become an issue.) This has the effect of adding arcs between (for example) $\mathbf{x}_{t+1}$. This explains why the performance gap between MEMMs and CRFs is not always as large as might be expected.

Finally, one might try a different way to combine the advantages of conditional training and directed models. One can imagine defining a directed model $p(\mathbf{y}, \mathbf{x})$, perhaps a generative model, and then training it by optimizing the resulting conditional likelihood $p(\mathbf{y}|\mathbf{x})$. In fact, this procedure has long been done in the speech community, where it is called maximum mutual information training. However, this does not have strong computational benefits over CRFs. The reason is that computing the conditional likelihood $p(\mathbf{y}|\mathbf{x})$ requires computing the marginal probability $p(\mathbf{x})$, which plays the same role as $Z(\mathbf{x})$ in the CRF likelihood. In fact, training is more complex in a directed model, because the model parameters are constrained to be probabilities—constraints which can actually make the optimization problem more difficult.

5.2 Frontier Areas

Finally, we describe a few open research areas that related to CRFs. In all of the cases below, the research question is a special case of a larger question for general graphical models, but there are special additional considerations in conditional models that make the problem more difficult.

5.2.1 Bayesian CRFs

Because of the large number of parameters in typical applications of CRFs, the models can be prone to overfitting. The standard way to control this is using regularization, as described in Section 4.1.1. One way that we motivated this procedure is as an approximation to a fully Bayesian procedure. That is, instead of predicting the labels of a testing instance $\mathbf{x}$ as $\mathbf{y}^* = \max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}; \hat{\theta})$, where $\hat{\theta}$ is a single parameter estimate, in a Bayesian method we would use the predictive distribution $\mathbf{y}^* = \max_{\mathbf{y}} \int p(\mathbf{y}|\mathbf{x}; \theta) p(\theta) \prod_{i=1}^{N} p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \theta) d\theta$. This integral over θ needs to be approximated, for example, by MCMC.

In general, it is difficult to formulate efficient Bayesian methods for undirected models; see [83, 82] for some of the few examples in this regard. A few papers have specially considered approximate inference algorithms for Bayesian CRFs [92, 133], but while these methods are interesting, they do not seem to be useful at the scale of current CRF applications (e.g., those in Table 4.1). Even for linear chain models, Bayesian methods are not commonly in use for CRFs, primarily due to the computational demands. If all we want is the benefits of model averaging, one may question whether simpler ensemble learning techniques, such as bagging, would give the same benefit. However, the Bayesian perspective does have other potential benefits, particularly when more complex, hierarchical priors are considered.

5.2.2 Semi-supervised CRFs

One practical difficulty in applying CRFs is that training requires obtaining true labels for potentially many sequences. This can be expensive because it is more time consuming for a human labeller to provide

labels for sequence labelling than for simple classification. For this reason, it would be very useful to have techniques that can obtain good accuracy given only a small amount of labeled data.

One strategy for achieving this goal is *semi-supervised learning*, in which in addition to some fully-labelled data $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, the data set is assumed to contain a large number of unlabelled instances $\{\mathbf{x}^{(j)}\}_{j=1}^{M}$, for which we observe only the inputs. However, unlike in generative models, it is less obvious how to incorporate unlabelled data into a conditional criterion, because the unlabelled data is a sample from the distribution $p(\mathbf{x})$, which in principle need have no relationship to the CRF $p(\mathbf{y}|\mathbf{x})$. In order to deal with this, several different types of regularization terms have been proposed that take the unlabelled data into account, including entropy regularization [39, 45], generalized expectation criteria [69], posterior regularization [32, 38], and measurement-based learning [62].

5.2.3 Structure Learning in CRFs

All of the methods described in this tutorial assume that the structure of the model has been decided in advance. It is natural to ask if we can learn the structure of the model as well. As in graphical models more generally, this is a difficult problem. In fact, Bradley and Guestrin [12] point out an interesting complication that is specific to conditional models. Typically, maximum likelihood structure learning can be performed efficiently if the model is restricted to be tree-structured, using the well-known Chow-Liu algorithm. The analogous algorithm in the conditional case is more difficult, however, because it requires estimating marginal distributions of the form $p(y_u, y_v|\mathbf{x}_{1:N})$, that is, we need to estimate the effects of the entire input on every pair of variables. It is difficult to estimate these distributions efficiently without knowing the structure of the model to begin with.

Acknowledgments

We thank Francine Chen, Benson Limketkai, Gregory Druck, Kedar Bellare, and Ray Mooney for useful comments on earlier versions of this tutorial. A previous version of this tutorial has appeared in Sutton and

McCallum [116], and as part of Charles Sutton's doctoral dissertation [113].

References

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2): 325–343, 2000.

[2] Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden Markov support vector machines. In *International Conference on Machine Learning (ICML)*, 2003.

[3] Galen Andrew and Jianfeng Gao. Scalable training of l_1-regularized log-linear models. In *International Conference on Machine Learning (ICML)*, 2007.

[4] Gökhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan, editors. *Predicting Structured Data*. MIT Press, 2007.

[5] Axel Bernal, Koby Crammer, Artemis Hatzigeorgiou, and Fernando Pereira. Global discriminative learning for higher-accuracy computational gene prediction. *PLoS Computational Biology*, 3 (3), 2007.

[6] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

[7] Julian Besag. Spatial interaction and the statistical analysis of

lattice systems. *Journal of the Royal Statistical Society. Series B*, 36(2):192–236, 1974.

[8] Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3: 993, 2003.

[10] Phil Blunsom and Trevor Cohn. Discriminative word alignment with conditional random fields. In *International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL)*, pages 65–72, 2006.

[11] Léon Bottou. Stochastic gradient descent examples on toy problems. 2010. URL `http://leon.bottou.org/projects/sgd`.

[12] Joseph K. Bradley and Carlos Guestrin. Learning tree conditional random fields. In *International Conference on Machine Learning (ICML 2010)*, 2010.

[13] Razvan Bunescu and Raymond J. Mooney. Collective information extraction with relational Markov networks. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, 2004.

[14] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.*, 63(2):129–156, 1994. ISSN 0025-5610.

[15] Rich Caruana. Multitask learning. *Machine Learning*, 28(1): 41–75, 1997. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1007379606734.

[16] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms using different performance metrics. Technical Report TR2005-1973, Cornell University, 2005. `http://www.niculescu-mizil.org/paper.php?p=comparison.tr.pdf`.

[17] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the Human Lan-*

guage Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP), 2005.

[18] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, 2007.

[19] Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL)*, pages 103–110, 2004.

[20] Trevor Cohn. Efficient inference in large conditional random fields. In *European Conference on Machine Learning (ECML)*, pages 606–613, Berlin, Germany, September 2006.

[21] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[22] Philip J. Cowans and Martin Szummer. A graphical model for simultaneous partitioning and labeling. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[23] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, Jan 2003.

[24] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 2006.

[25] Aron Culotta and Andrew McCallum. Confidence estimation for information extraction. In *Human Language Technology Conference (HLT)*, 2004.

[26] Aron Culotta, Ron Bekkerman, and Andrew McCallum. Extracting social networks and contact information from email and the web. In *First Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, 2004.

[27] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured

prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005. URL `http://pub.hal3.name/#daume05laso`.

[28] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning Journal*, 2009.

[29] Joshua Dillon and Guy Lebanon. Statistical and computational tradeoffs in stochastic composite likelihood. arXiv:1003.0691v1, 2010.

[30] Jenny Finkel, Trond Grenager, and Christopher D. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[31] Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Annual Meeting of the Association for Computational Linguistics (ACL/HLT)*, pages 959–967, 2008.

[32] Kuzman Ganchev, Joao Graca, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. Technical Report MS-CIS-09-16, University of Pennsylvania Department of Computer and Information Science, 2009.

[33] Alan E. Gelfand and Adrian F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409, 1990.

[34] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6: 721–741, 1984.

[35] Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Conference on Information and Knowledge Management (CIKM)*, 2005.

[36] Amir Globerson, Terry Koo, Xavier Carreras, and Michael Collins. Exponentiated gradient algorithms for log-linear structured prediction. In *International Conference on Machine Learning (ICML)*, 2007.

[37] Joshua Goodman. Exponential priors for maximum entropy mod-

els. In *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2004.

[38] Joao Graca, Kuzman Ganchev, Ben Taskar, and Fernando Pereira. Posterior vs parameter sparsity in latent variable models. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 664–672. 2009.

[39] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, 2004.

[40] Michelle L. Gregory and Yasemin Altun. Using conditional random fields to predict pitch accents in conversational speech. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 677–683, 2004. doi: http://dx.doi.org/10.3115/1218955.1219041.

[41] Asela Gunawardana, Milind Mahajan Alex Acero, and John C. Platt. Hidden conditional random fields for phone classification. In *International Conference on Speech Communication and Technology*, 2005.

[42] John M. Hammersley and Peter Clifford. Markov fields on finite graphs and lattices. 1971.

[43] Xuming He, Richard S. Zemel, and Miguel Á. Carreira-Perpiñián. Multiscale conditional random fields for image labelling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[44] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[45] F. Jiao, S. Wang, C. Lee, R. Greiner, and D Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL)*, 2006.

[46] S. Sathiya Keerthi and S. Sundararajan. CRF versus SVM-struct for sequence labeling. Technical report, Yahoo! Research, 2007. URL `http://www.keerthis.com/crf_`

`comparison_keerthi_07.pdf`.

[47] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23: 462–466, 1952.

[48] Dan Klein and Christopher D. Manning. Conditional structure versus conditional estimation in NLP models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[49] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[50] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[51] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[52] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Advances in Neural Information Processing Systems*, 2008.

[53] Sanjiv Kumar and Martial Hebert. Discriminative fields for modeling spatial dependencies in natural images. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS)16*. MIT Press, Cambridge, MA, 2003.

[54] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning (ICML)*, 2001.

[55] John Langford, Alex Smola, and Martin Zinkevich. Slow learners are fast. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2331–2339, 2009.

[56] T. Lavergne, O. Cappé, and F. Yvon. Practical very large scale crfs. In *Proc. 48th Annual Meeting Association for Computational Linguistics (ACL)*, pages 504–513, 2010.

[57] Yann Le Cun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998. URL `http://leon.bottou.org/papers/lecun-98x`.

[58] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[59] Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2007.

[60] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer-Verlag, 2001.

[61] P. Liang and M.I. Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *International Conference on Machine Learning (ICML)*, pages 584–591. ACM, 2008.

[62] P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *International Conference on Machine Learning (ICML)*, 2009.

[63] Chih-Jen Lin, Ruby Chiu-Hsing Weng, and Sathiya Keerthi. Trust region newton methods for large-scale logistic regression. In *Interational Conference on Machine Learning (ICML)*, 2007.

[64] Bruce G. Lindsay. Composite likelihood methods. *Contemporary Mathematics*, pages 221–239, 1988.

[65] Yan Liu, Jaime Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. Protein fold recognition using segmentation conditional random fields (SCRFs). *Journal of Computational Biology*, 13 (2):394–406, 2006.

[66] David J. Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. WinBUGS—a Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, 10(4):325–337, 2000.

[67] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[68] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In Dan Roth and Antal van den Bosch, editors, *Conference on Natural Language Learning (CoNLL)*, pages 49–55, 2002.

[69] Gideon Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proceedings of Association of Computational Linguistics*, 2008.

[70] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[71] Andrew McCallum. Efficiently inducing features of conditional random fields. In *Conference on Uncertainty in AI (UAI)*, 2003.

[72] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Seventh Conference on Natural Language Learning (CoNLL)*, 2003.

[73] Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 905–912. MIT Press, Cambridge, MA, 2005.

[74] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *International Conference on Machine Learning (ICML)*, pages 591–598. Morgan Kaufmann, San Francisco, CA, 2000.

[75] Andrew McCallum, Kedar Bellare, and Fernando Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *Conference on Uncertainty in AI (UAI)*, 2005.

[76] Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor

graphs. In *Advances on Neural Information Processing Systems (NIPS)*, 2009.

[77] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[78] Thomas P. Minka. The EP energy function and minimization schemes. Technical report, 2001. `http://research.microsoft.com/~minka/papers/ep/minka-ep-energy.pdf`.

[79] Thomas P. Minka. A comparsion of numerical optimizers for logistic regression. Technical report, 2003. `http://research.microsoft.com/~minka/papers/logreg/`.

[80] Tom Minka. Discriminative models, not discriminative training. Technical Report MSR-TR-2005-144, Microsoft Research, October 2005. `ftp://ftp.research.microsoft.com/pub/tr/TR-2005-144.pdf`.

[81] Tom Minka. Divergence measures and message passing. Technical Report MSR-TR-2005-173, Microsoft Research, 2005.

[82] Iain Murray. *Advances in Markov chain Monte Carlo methods*. PhD thesis, Gatsby computational neuroscience unit, University College London, 2007.

[83] Iain Murray, Zoubin Ghahramani, and David J. C. MacKay. MCMC for doubly-intractable distributions. In *Uncertainty in Artificial Intelligence (UAI)*, pages 359–366. AUAI Press, 2006.

[84] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *International Conference on Machine Learning (ICML)*, 2004.

[85] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848, Cambridge, MA, 2002. MIT Press.

[86] N. Nguyen and Y. Guo. Comparisons of sequence labeling algorithms and extensions. In *International Conference on Machine Learning (ICML)*, 2007.

[87] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*.

Springer-Verlag, New York, 1999. ISBN 0-387-98793-2.

[88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[89] Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using conditional random fields. In *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, 2004.

[90] Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of The 20th International Conference on Computational Linguistics (COLING)*, pages 562–568, 2004.

[91] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the ACM SIGIR*, 2003.

[92] Yuan Qi, Martin Szummer, and Thomas P. Minka. Bayesian conditional random fields. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[93] Yuan Qi, Martin Szummer, and Thomas P. Minka. Diagram structure recognition by Bayesian conditional random fields. In *International Conference on Computer Vision and Pattern Recognition*, 2005.

[94] A. Quattoni, S. Wang, L.P. Morency, M. Collins, and T. Darrell. Hidden-state conditional random fields. *IEEE PAMI*, 2007.

[95] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1097–1104. MIT Press, Cambridge, MA, 2005.

[96] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, 1989.

[97] Nathan Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In *International Conference on Machine Learning*, July 2006.

[98] Matthew Richardson and Pedro Domingos. Markov logic net-

works. *Machine Learning*, 62(1–2):107–136, 2006.

[99] Stefan Riezler, Tracy King, Ronald Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

[100] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[101] Christian Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2004.

[102] David Rosenberg, Dan Klein, and Ben Taskar. Mixture-of-parents maximum entropy Markov models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[103] Dan Roth and Wen-tau Yih. Integer linear programming inference for conditional random fields. In *International Conference on Machine Learning (ICML)*, pages 737–744, 2005.

[104] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, 2000. See `http://lcg-www.uia.ac.be/~erikt/research/np-chunking.html`.

[105] Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1185–1192. MIT Press, Cambridge, MA, 2005.

[106] Kengo Sato and Yasubumi Sakakibara. RNA secondary structural alignment with conditional random fields. *Bioinformatics*, 21:ii237–242, 2005. URL `http://bioinformatics.oxfordjournals.org/cgi/content/abstract/21/suppl_2/ii237`.

[107] Burr Settles. Abner: an open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[108] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Conference on Human Language Technology*

and North American Association for Computational Linguistics (HLT-NAACL), pages 213–220, 2003.

[109] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *International Conference on Machine Learning (ICML)*, 2007.

[110] P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 868–873, Pittsburgh, PA, 2005. AAAI Press.

[111] David H. Stern, Thore Graepel, and David J. C. MacKay. Modelling uncertainty in the game of go. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1353–1360. MIT Press, Cambridge, MA, 2005.

[112] Ilya Sutskever and Tijmen Tieleman. On the convergence properties of contrastive divergence. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[113] Charles Sutton. *Efficient Training Methods for Conditional Random Fields*. PhD thesis, University of Massachusetts, 2008.

[114] Charles Sutton and Andrew McCallum. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

[115] Charles Sutton and Andrew McCallum. Piecewise training of undirected models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[116] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[117] Charles Sutton and Andrew McCallum. Piecewise training for structured prediction. *Machine Learning*, 77(2–3):165–194, 2009.

[118] Charles Sutton and Tom Minka. Local training and belief propagation. Technical Report TR-2006-121, Microsoft Research, 2006.

[119] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic

models for labeling and segmenting sequence data. In *International Conference on Machine Learning (ICML)*, 2004.

[120] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8:693–723, March 2007. URL `publications/jmlr-sutton07a.pdf`.

[121] Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2002.

[122] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[123] Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. Max-margin parsing. In *Empirical Methods in Natural Language Processing (EMNLP04)*, 2004.

[124] Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 73–80, 2005.

[125] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada, 2003.

[126] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, 2003.

[127] Paul Viola and Mukund Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the ACM SIGIR*, 2005.

[128] S.V.N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin Murphy. Accelerated training of conditional random fields with stochastic meta-descent. In *International Conference on Machine Learning (ICML)*, pages 969–976, 2006.

[129] Martin J. Wainwright. Estimating the wrong Markov random field: Benefits in the computation-limited setting. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

[130] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. Technical Report Technical Report 649, UC Berkeley, Dept. of Statistics, September 2003.

[131] M.J. Wainwright and M.I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[132] Hanna Wallach. Efficient training of conditional random fields. M.Sc. thesis, University of Edinburgh, 2002.

[133] Max Welling and Sridevi Parise. Bayesian random fields: The Bethe-Laplace approximation. In *Uncertainty in Artificial Intelligence (UAI)*, 2006.

[134] Michael Wick, Khashayar Rohanimanesh, Andrew McCallum, and AnHai Doan. A discriminative approach to ontology alignment. In *International Workshop on New Trends in Information Integration (NTII)*, 2008.

[135] Michael Wick, Khashayar Rohanimanesh, Aron Culotta, and Andrew McCallum. Samplerank: Learning preferences from atomic gradients. In *Neural Information Processing Systems (NIPS) Workshop on Advances in Ranking*, 2009.

[136] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR2004-040, Mitsubishi Electric Research Laboratories, 2004.

[137] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, July 2005.

[138] Jin Yu, S.V.N. Vishwanathan, Simon Güunter, and Nicol N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. *Journal of Machine*

Learning Research, 11:1145–1200, Mar 2010.