Foundations and Trends® in
Machine Learning
Vol. 4, No. 4 (2011) 267–373

DOI: 10.1561/2200000013

now
the essence of knowledge

# An Introduction to Conditional Random Fields

By Charles Sutton and Andrew McCallum

## Contents

Foundations and Trends® in
Machine Learning
Vol. 4, No. 4 (2011) 267–373

DOI: 10.1561/2200000013

now
the essence of knowledge

# An Introduction to Conditional Random Fields

**Charles Sutton[1] and Andrew McCallum[2]**

[1] *School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK, csutton@inf.ed.ac.uk*
[2] *Department of Computer Science, University of Massachusetts, Amherst, MA, 01003, USA, mccallum@cs.umass.edu*

## Abstract

Many tasks involve predicting 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. They combine 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 survey describes *conditional random fields*, a popular probabilistic method for structured prediction. CRFs have seen wide application in many areas, including 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 survey is intended to be useful to practitioners in a wide variety of fields.

# 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 [49, 61, 69], estimating the score in a game of Go [130], segmenting genes in a strand of DNA [7], and syntactic parsing of natural-language text [144]. In such applications, we wish to predict an output 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, in English adjectives do not

usually follow nouns, and in computer vision, neighboring regions in an image tend to have similar labels. Another difficulty is that 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 this approach has advantages, it also has important limitations. Not only can the dimensionality of $\mathbf{x}$ be very large, but the features may have complex dependencies, so constructing a probability distribution over them is difficult. Modeling the dependencies among inputs can lead to intractable models, but ignoring them can lead to reduced performance.

A solution to this problem is a *discriminative* approach, similar to that taken in classifiers such as logistic regression. Here we model the conditional distribution $p(\mathbf{y}|\mathbf{x})$ directly, which is all that is needed for classification. This is the approach taken by conditional random fields (CRFs). CRFs are essentially a way of combining the advantages of discriminative classification and graphical modeling, combining the ability to compactly model multivariate outputs $\mathbf{y}$ with the ability to leverage a large number of input features $\mathbf{x}$ 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 regression 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 interest in applying CRFs to many different problems. Successful applications have included text processing [105, 124, 125], bioinformatics [76, 123], and computer vision [49, 61]. 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 independence assumption among entities, as in relational learning [142].

This survey describes modeling, inference, and parameter estimation using CRFs. We do not assume previous knowledge of graphical modeling, so this survey is intended to be useful to practitioners in a wide variety of fields. We begin by describing modeling issues in CRFs (Section 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 sections, we describe inference (Section 4) and learning (Section 5) in CRFs. In this context, *inference* refers both to the task of computing the marginal distributions of $p(\mathbf{y}|\mathbf{x})$ and to the related task of computing the maximum probability assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. With respect to *learning*, we will focus on the parameter estimation task, in which $p(\mathbf{y}|\mathbf{x})$ is determined by parameters that we will choose in order to best fit a set of training examples $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$. The inference and learning procedures are often closely coupled, because learning usually calls inference as a subroutine.

Finally, we discuss relationships between CRFs and other families of models, including other structured prediction methods, neural networks, and maximum entropy Markov models (Section 6).

## 1.1 Implementation Details

Throughout this survey, we strive to point out implementation details that are sometimes elided in the research literature. For example, we discuss issues relating to feature engineering (Section 2.5), avoiding numerical underflow during inference (Section 4.3), and the scalability of CRF training on some benchmark problems (Section 5.5).

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 section, we describe CRFs 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, 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 a CRF, both for the commonly-used case of linear chains (Section 2.3), and for general graphical structures (Section 2.4). Because the accuracy of a CRF is strongly dependent on the features that are used, we also describe some commonly used tricks for engineering features (Section 2.5). Finally, we present two examples of applications of CRFs (Section 2.6), a broader survey of typical application areas for CRFs (Section 2.7).

## 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 [89], computer vision [41], knowledge representation [103], Bayesian statistics [40], and natural-language processing [11, 63].

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 treatment of graphical modeling and approximate inference is available in a textbook by Koller and Friedman [57].

### 2.1.1 Undirected Models

We consider probability distributions over sets of random variables $Y$. We index the variables by integers $s \in 1, 2, \ldots |Y|$. Every variable $Y_s \in Y$ takes outcomes from a set $\mathcal{Y}$, which can be either continuous or discrete, although we consider only the discrete case in this survey. An arbitrary assignment to $Y$ is denoted by a vector $\mathbf{y}$. Given a variable $Y_s \in Y$, the notation $y_s$ denotes the value assigned to $Y_s$ by $\mathbf{y}$. The notation $\mathbf{1}_{\{y=y'\}}$ denotes an indicator function of $y$ which takes the value 1 when $y = y'$ 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 $Y_s$ 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 $\Psi_a(\mathbf{y}_a)$ where $a$ is an integer index that ranges from 1 to $A$, the number of factors. Each factor $\Psi_a$ depends only on a subset $Y_a \subseteq Y$ of the variables. The value $\Psi_a(\mathbf{y}_a)$ is a non-negative scalar that can be thought of as a measure of how compatible the values $\mathbf{y}_a$ are with each other. Assignments that have higher compatibility values will have higher probability. This factorization can allow us to represent $p$ much more efficiently, because the sets $Y_a$ may be much smaller than the full variable set $Y$.

An undirected graphical model is a family of probability distributions that each factorize according to a given collection of factors. Formally, given a collection of subsets $\{Y_a\}_{a=1}^{A}$ of $Y$, an *undirected graphical model* is the set of all distributions that can be written as

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{a=1}^{A} \Psi_a(\mathbf{y}_a), \tag{2.1}$$

for any choice of *factors* $\mathcal{F} = \{\Psi_a\}$ that have $\Psi_a(\mathbf{y}_a) \geq 0$ for all $\mathbf{y}_a$. (The factors are also called *local functions* or *compatibility functions*.) We will use the term *random field* to refer to a particular distribution among those defined by an undirected model.

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

$$Z = \sum_{\mathbf{y}} \prod_{a=1}^{A} \Psi_a(\mathbf{y}_a). \tag{2.2}$$

The quantity $Z$, considered as a function of the set $\mathcal{F}$ of factors, is also called the *partition function*. Notice that the summation in (2.2) is over the exponentially many possible assignments to $\mathbf{y}$. For this reason, computing $Z$ is intractable in general, but much work exists on how to approximate it (see Section 4).

The reason for the term "graphical model" is that the factorization factorization (2.1) can be represented compactly by means of a graph. A particularly natural formalism for this is provided by *factor graphs* [58]. A factor graph is a bipartite graph $G = (V, F, E)$ in which one set of nodes $V = \{1, 2, \ldots, |Y|\}$ indexes the random variables in the model, and the other set of nodes $F = \{1, 2, \ldots, A\}$ indexes the factors.

The semantics of the graph is that if a variable node $Y_s$ for $s \in V$ is connected to a factor node $\Psi_a$ for $a \in F$, then $Y_s$ is one of the arguments of $\Psi_a$. So a factor graph directly describes the way in which a distribution $p$ decomposes into a product of local functions.

We formally define the notion of whether a factor graph "describes" a given distribution or not. Let $N(a)$ be the neighbors of the factor with index $a$, i.e., a set of variable indices. Then:

---

**Definition 2.1.** A distribution $p(\mathbf{y})$ *factorizes according to a factor graph* $G$ if there exists a set of local functions $\Psi_a$ such that $p$ can be written as

$$p(\mathbf{y}) = Z^{-1} \prod_{a \in F} \Psi_a(\mathbf{y}_{N(a)}) \tag{2.3}$$

---

A factor graph $G$ describes an undirected model in the same way that a collection of subsets does. In (2.1), take the collection of subsets to be the set of neighborhoods $\{Y_{N(a)} | \forall a \in F\}$. The resulting undirected graphical model from the definition in (2.1) is exactly the set of all distributions that factorize according to $G$.

For example, Figure 2.1 shows an example of a factor graph over three random variables. In that figure, the circles are variable nodes, and the shaded boxes are factor nodes. We have labelled the nodes according to which variables or factors they index. This factor graph describes the set of all distributions $p$ over three variables that can be written as $p(y_1, y_2, y_3) = \Psi_1(y_1, y_2)\Psi_2(y_2, y_3)\Psi_3(y_1, y_3)$ for all $\mathbf{y} = (y_1, y_2, y_3)$.

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 a different

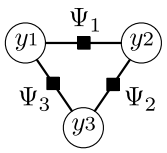


Fig. 2.1 An example of a factor graph over three variables.

undirected graph, called a *Markov network*, which directly represents conditional independence relationships in a multivariate distribution. Markov networks are graphs over random variables only, rather than random variables and factors. Now let $G$ be an undirected graph over integers $V = \{1, 2, \ldots, |Y|\}$ that index each random variable of interest. For a variable index $s \in V$, let $N(s)$ denote its neighbors in $G$. Then we say that a distribution $p$ is *Markov with respect to* $G$ if it satisfies the local Markov property: for any two variables $Y_s, Y_t \in Y$, the variable $Y_s$ is independent of $Y_t$ conditioned on its neighbors $Y_{N(s)}$. Intuitively, this means that $Y_{N(s)}$ on its own contains all of the information that is useful for predicting $Y_s$.

Given a factorization of a distribution $p$ as in (2.1), a corresponding 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 $p(y_s|\mathbf{y}_{N(s)})$ that follows from (2.1) is a function only of variables that appear in the Markov blanket.

A Markov network has an undesirable ambiguity from the factorization perspective. Consider the three-node Markov network in Figure 2.2 (left). Any distribution that factorizes as $p(y_1, y_2, y_3) \propto f(y_1, y_2, y_3)$ for some positive function $f$ is Markov with respect to this graph. However, we may wish to use a more restricted parameterization, where $p(y_1, y_2, y_3) \propto f(y_1, y_2)g(y_2, y_3)h(y_1, y_3)$. This second model family is a strict subset of the first, so in this smaller family we might not require as much data to accurately estimate the distribution. But the Markov network formalism cannot distinguish between these two

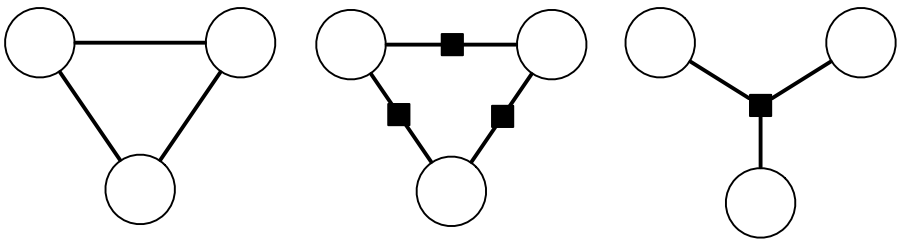

Fig. 2.2 A Markov network with an ambiguous factorization. Both of the factor graphs at right factorize according to the Markov network at left.

parameterizations. In contrast, 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 $G$ be a directed acyclic graph, in which $\pi(s)$ are the indices of the parents of $Y_s$ in $G$. A directed graphical model is a family of distributions that factorize as:

$$p(\mathbf{y}) = \prod_{s=1}^{S} p(y_s|\mathbf{y}_{\pi(s)}). \tag{2.4}$$

We refer to the distributions $p(y_s|\mathbf{y}_{\pi(s)})$ as *local conditional distributions*. Note that $\pi(s)$ can be empty for variables that have no parents. In this case $p(y_s|\mathbf{y}_{\pi(s)})$ should be understood as $p(y_s)$.

It can be shown by induction 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 (a) the factors equal conditional distributions over subsets of variables, and (b) the normalization constant $Z = 1$. 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 (2.7), which is depicted graphically in Figure 2.3 (left). In those figures, the shaded nodes indicate variables that are observed in some data set. This is a convention that we will use throughout this survey.

### 2.1.3 Inputs and Outputs

This survey considers the situation in which we know in advance which variables we will want to predict. The variables in the model will be partitioned into a set of *input variables* $X$ that we assume are always observed, and another set $Y$ of *output variables* that we wish to predict. For example, an assignment $\mathbf{x}$ might represent a vector of each word that occurs in a sentence, and $\mathbf{y}$ a vector of part of speech labels for each word.

We will be interested in building distributions over the combined set of variables $X \cup Y$, and we will extend the previous notation in order to accommodate this. For example, an undirected model over $X$ and $Y$ would be given by

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{a=1}^{A} \Psi_a(\mathbf{x}_a, \mathbf{y}_a), \tag{2.5}$$

where now each local function $\Psi_a$ depends on two subsets of variables $X_a \subseteq X$ and $Y_a \subseteq Y$. The normalization constant becomes

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{a \in F} \Psi_a(\mathbf{x}_a, \mathbf{y}_a), \tag{2.6}$$

which now involves summing over all assignments both to $\mathbf{x}$ and $\mathbf{y}$.

## 2.2 Generative versus Discriminative Models

In this section we discuss several examples of simple graphical models that have been used in 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 CRFs. We devote special attention to the hidden Markov model (HMM), because it is closely related to the linear-chain CRF.

One of the main purposes of this section is to contrast generative and discriminative models. Of the models that we will describe, two are generative (the naive Bayes and HMM) and one is discriminative (the logistic regression model). *Generative models* are models that describe how a label vector $\mathbf{y}$ can probabilistically "generate" a feature vector $\mathbf{x}$. *Discriminative models* work in the reverse direction, describing directly how to take a feature vector $\mathbf{x}$ and assign it a label $\mathbf{y}$. In principle, either type of model can be converted to the other type using Bayes's rule, but in practice the approaches are distinct, each with potential advantages as we describe in Section 2.2.3.

### 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 model of the form:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^{K} p(x_k|y). \tag{2.7}$$

This model can be described by the directed model shown in Figure 2.3 (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 2.3 (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). This classifier can be 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.[1] This leads to the conditional distribution:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\theta_y + \sum_{j=1}^{K} \theta_{y,j} x_j\right\}, \tag{2.8}$$

where $Z(\mathbf{x}) = \sum_y \exp\{\theta_y + \sum_{j=1}^{K} \theta_{y,j} x_j\}$ is a normalizing constant, and $\theta_y$ is a bias weight that acts like $\log p(y)$ in naive Bayes. Rather than using one weight vector per class, as in (2.8), 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

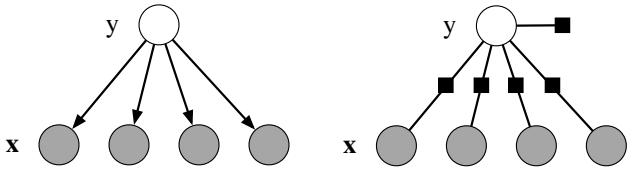


Fig. 2.3 The naive Bayes classifier, as a directed model (left), and as a factor graph (right).

[1] By log in this survey we will always mean the natural logarithm.

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 $\theta_k$ to index its corresponding weight $\theta_{y',j}$. Using this notational trick, the logistic regression model becomes:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y, \mathbf{x})\right\}. \tag{2.9}$$

We introduce this notation because it mirrors the notation for CRFs 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 in the graphical model are arranged in a sequence. To motivate this 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 an entity, and to classify each entity by type (person, organization, location, and so on). The challenge of this problem is that many named entity strings 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) [111]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^{\mathrm{T}}$ by assuming that there is an underlying sequence of *states* $Y = \{y_t\}_{t=1}^{\mathrm{T}}$. We let $S$ be the finite set of possible states and $O$ the finite set of possible observations, i.e., $x_t \in O$ and $y_t \in S$ for all $t$. 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

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{\mathrm{T}} p(y_t|y_{t-1}) p(x_t|y_t). \tag{2.10}$$

To simplify notation in the above equation, we create a "dummy" initial state $y_0$ which is clamped to 0 and begins every state sequence. This allows us to write the initial state distribution $p(y_1)$ as $p(y_1|y_0)$.

HMMs have been used for many sequence labeling tasks in natural-language processing such as part-of-speech tagging, named-entity recognition, and information extraction.

### 2.2.3 Comparison

Both generative models and discriminative models describe distributions over $(\mathbf{y}, \mathbf{x})$, but they work in different directions. A *generative model*, such as the naive Bayes classifier and the HMM, is a family of joint distributions that factorizes as $p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$, that is, it describes how to sample, or "generate," values for features given the label. A *discriminative model*, such as the logistic regression model, is

a family of conditional distributions $p(\mathbf{y}|\mathbf{x})$, that is, the classification rule is modeled directly. In principle, a discriminative model could also be used to obtain a joint distribution $p(\mathbf{y},\mathbf{x})$ by supplying a marginal distribution $p(\mathbf{x})$ over the inputs, but this is rarely needed.

The main conceptual difference between discriminative and generative models 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, a naive application of 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 (2.7). This is a family of joint distributions whose conditionals all take the "logistic regression form" (2.9). But there are many other joint models, some with complex dependencies among $\mathbf{x}$, whose conditional distributions also have the form (2.9). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(\mathbf{x})$. Discriminative models, such as CRFs, make conditional independence assumptions among $\mathbf{y}$, and assumptions about how the $\mathbf{y}$ can depend on $\mathbf{x}$, but do not make conditonal independence assumptions among $\mathbf{x}$. This point 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. The first choice is to enhance the model to represent dependencies among the inputs, e.g., by adding directed edges among each $\mathbf{x}_t$. But this 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 choice is to make simplifying independence assumptions, such as the naive Bayes assumption. For example, an HMM model with a naive Bayes assumption would have the form $p(\mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} p(y_t|y_{t-1}) \prod_{k=1}^{K} p(x_{tk}|y_t)$. This idea can sometimes 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 [19].

Furthermore, naive Bayes can produce poor probability estimates. As an illustrative example, imagine training naive Bayes on a two class problem in which all the features are repeated, that is, given an original feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_K)$, we transform it to $\mathbf{x}' = (x_1, x_1, x_2, x_2, \ldots, x_K, x_K)$ and then run naive Bayes. Even though no new information has been added to the data, this transformation will increase the confidence of the probability estimates, by which we mean that the naive Bayes estimates of $p(y|\mathbf{x}')$ will tend to be farther from 0.5 than those of $p(y|\mathbf{x})$.

Assumptions like naive Bayes can be especially problematic when we generalize to sequence models, because inference should combine evidence from different parts of the model. If probability estimates of the label at each sequence position are overconfident, it can 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 (2.7) defines the same family of distributions as the logistic regression model (2.9), if we interpret it generatively as

$$p(y, \mathbf{x}) = \frac{\exp\{\sum_k \theta_k f_k(y, \mathbf{x})\}}{\sum_{\tilde{y}, \tilde{\mathbf{x}}} \exp\{\sum_k \theta_k f_k(\tilde{y}, \tilde{\mathbf{x}})\}}. \tag{2.11}$$

This means that if the naive Bayes model (2.7) 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 (2.11), 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 [98], naive Bayes and logistic regression form a *generative-discriminative pair*. For a recent theoretical perspective on generative and discriminative models, see Liang and Jordan [72].

In principle, it may not be clear why these approaches should be so different, because we can always convert between the two methods using Bayes rule. For example, in the naive Bayes model, it is easy to convert the joint $p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$ into a conditional distribution $p(\mathbf{y}|\mathbf{x})$. Indeed, this conditional has the same form as the logistic regression model (2.9). And if we managed to obtain a "true" generative model for the data, that is, a distribution $p^*(\mathbf{y}, \mathbf{x}) = p^*(\mathbf{y})p^*(\mathbf{x}|\mathbf{y})$ from which the data were actually sampled, then we could simply compute the true $p^*(\mathbf{y}|\mathbf{x})$, which is exactly the target of the discriminative approach. But it is precisely because we never have the true distribution that the two approaches are different in practice. Estimating $p(\mathbf{y})p(\mathbf{x}|\mathbf{y})$ first, and then computing the resulting $p(\mathbf{y}|\mathbf{x})$ (the generative approach) yields a different estimate than estimating $p(\mathbf{y}|\mathbf{x})$ directly. In other words, generative and discriminative models both have the aim of estimating $p(\mathbf{y}|\mathbf{x})$, but they get there in different ways.

One perspective for gaining insight into the difference between generative and discriminative modeling is due to Minka [93]. Suppose we have a generative model $p_g$ with parameters $\theta$. By definition, this takes the form:

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

But we could also rewrite $p_g$ using the chain rule of probability as

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta)p_g(\mathbf{y}|\mathbf{x}; \theta), \tag{2.13}$$

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, $p(\mathbf{x}) = p_c(\mathbf{x};\theta') = \sum_{\mathbf{y}} p_g(\mathbf{y},\mathbf{x}|\theta')$, where $\theta'$ will typically be distinct from the $\theta$ in (2.13). 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

$$p_c(\mathbf{y},\mathbf{x}) = p_c(\mathbf{x};\theta')p_c(\mathbf{y}|\mathbf{x};\theta). \tag{2.14}$$

By comparing (2.13) with (2.14), 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 $\theta$ in (2.13) 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, the added degree of freedom brings about an increased risk of overfitting the training data, and generalizing worse on unseen data.

Although so far we have been criticizing generative models, they do have advantages as well. 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, in some cases 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 [98] 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. However, this need not always be the case, and both undirected generative models, such as the Markov random field (2.32), and directed discriminative models, such as the MEMM (6.2), are sometimes used. It can also be useful to depict discriminative models by directed graphs in which the $\mathbf{x}$ precede the $\mathbf{y}$.

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 HMM, and this analogue is a particular special case of CRF, as we explain in the next section. This analogy between naive Bayes, logistic regression, generative models, and CRFs is depicted in Figure 2.4.

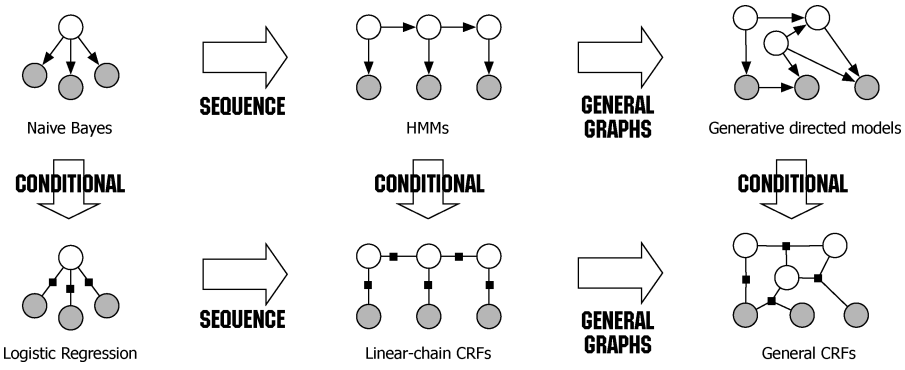


Fig. 2.4 Diagram of the relationship between naive Bayes, logistic regression, HMMs, linear-chain CRFs, generative models, and general CRFs.

## 2.3 Linear-chain CRFs

To motivate our introduction of linear-chain CRFs, 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 CRF with a particular choice of feature functions.

First, we rewrite the HMM joint (2.10) in a form that is more amenable to generalization. This is

$$p(\mathbf{y},\mathbf{x}) = \frac{1}{Z}\prod_{t=1}^{T}\exp\left\{\sum_{i,j\in S}\theta_{ij}\mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{y_{t-1}=j\}} + \sum_{i\in S}\sum_{o\in O}\mu_{oi}\mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{x_t=o\}}\right\}, \tag{2.15}$$

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.[2] If we had not added $Z$ in the (2.15), then some choices of $\theta$ would not yield proper distributions over $(\mathbf{y},\mathbf{x})$, e.g., set all the parameters to 1.

Now the interesting point is that (2.15) describes (almost) exactly the class (2.10) of HMMs. Every homogeneous HMM can be written in the form (2.15) by setting

$$\begin{aligned}\theta_{ij} &= \log p(y' = i|y = j)\\ \mu_{oi} &= \log p(x = o|y = i)\\ Z &= 1\end{aligned}$$

The other direction is true as well, namely, every distribution that factorizes as in (2.15) is an HMM.[3] (This can be shown by constructing the corresponding HMM using the forward–backward algorithm of Section 4.1.) So despite the added flexibility in the parameterization, we have not added any distributions to the family.

We can write (2.15) more compactly by introducing the concept of *feature functions*, just as we did for logistic regression in (2.9). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (2.15), there needs to be one feature $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$

[2] Not all choices of $\theta$ 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 usually over a finite set.

[3] But not necessarily a *homogeneous* HMM, which is an annoying technicality.

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:

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

Again, equation (2.16) defines exactly the same family of distributions as (2.15), and therefore as the original HMM equation (2.10).

The last step is to write the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that results from the HMM (2.16). This is

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y},\mathbf{x})}{\sum_{\mathbf{y}'}p(\mathbf{y}',\mathbf{x})} = \frac{\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t,y_{t-1},x_t)\right\}}{\sum_{\mathbf{y}'}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y'_t,y'_{t-1},x_t)\right\}}. \tag{2.17}$$

This conditional distribution (2.17) 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. This leads to the general definition of linear-chain CRFs:

---

**Definition 2.2.** Let $Y,X$ be random vectors, $\theta = \{\theta_k\} \in \Re^K$ be a parameter vector, and $\mathcal{F} = \{f_k(y,y',\mathbf{x}_t)\}_{k=1}^{K}$ 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:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t,y_{t-1},\mathbf{x}_t)\right\}, \tag{2.18}$$

where $Z(\mathbf{x})$ is an input-dependent normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t,y_{t-1},\mathbf{x}_t)\right\}. \tag{2.19}$$

---

Notice that a linear chain CRF can be described as a factor graph over $\mathbf{x}$ and $\mathbf{y}$, i.e.,

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}_t) \tag{2.20}$$

where each local function $\Psi_t$ has the special log-linear form:

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \tag{2.21}$$

This will be useful when we move to general CRFs in the next section.

Typically we will learn the parameter vector $\theta$ from data, as described in Section 5.

Previously we have 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 2.5. Other types of linear-chain CRFs are also useful, however. For example, 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 2.6.

In fact, since CRFs do not represent dependencies among the variables $\mathbf{x}_1, \ldots \mathbf{x}_T$, we can allow the factors $\Psi_t$ to depend on the entire observation vector $\mathbf{x}$ without breaking the linear graphical structure — allowing us to treat $\mathbf{x}$ as a single monolithic variable. As a result, the feature functions can be written $f_k(y_t, y_{t-1}, \mathbf{x})$ and have the freedom to examine all the input variables $\mathbf{x}$ together. This fact applies generally to CRFs and is not specific to linear chains. A linear-chain CRF

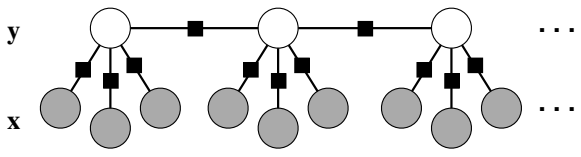


Fig. 2.5 Graphical model of the HMM-like linear-chain CRF from equation (2.17).

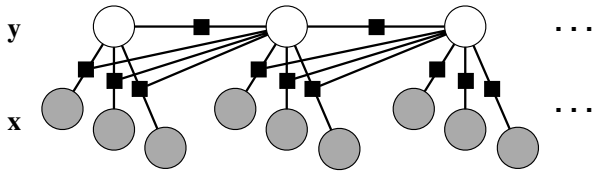


Fig. 2.6 Graphical model of a linear-chain CRF in which the transition factors depend on the current observation.

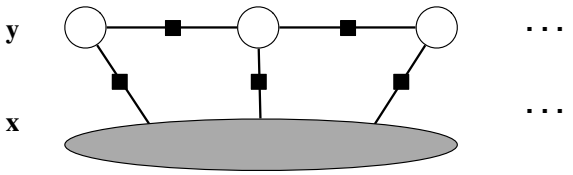


Fig. 2.7 Graphical model of a linear-chain CRF in which the transition factors depend on all of the observations.

with this structure in shown graphically in Figure 2.7. In this figure we show $\mathbf{x} = (\mathbf{x}_1, \ldots \mathbf{x}_T)$ as a single large observed node on which all of the factors depend, rather than showing each of the $\mathbf{x}_1, \ldots \mathbf{x}_T$ as individual nodes.

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 the forward–backward algorithm, as we explain in Section 4.1.

## 2.4 General CRFs

Now we generalize the previous discussion from a linear-chain to a general graph, matching the definition of a CRF originally given in

Lafferty et al. [63]. Conceptually the generalization is straightforward. We simply move from using a linear-chain factor graph to a more general factor graph.

---

**Definition 2.3.** Let $G$ be a factor graph over $X$ and $Y$. Then $(X, Y)$ is a *conditional random field* if for any value $\mathbf{x}$ of $X$, 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 $G$, then the conditional distribution for a CRF is

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{a=1}^{A} \Psi_a(\mathbf{y}_a, \mathbf{x}_a). \tag{2.22}$$

The difference between this equation and the general definition (2.1) of an undirected graphical model is that now the normalization constant $Z(\mathbf{x})$ is a function of the input $\mathbf{x}$. Because conditioning tends to simplify a graphical model, $Z(\mathbf{x})$ may be computable whereas $Z$ might not have been.

As we have done for HMMs and linear chain CRFs, it is often useful to require that $\log \Psi_a$ be linear over a prespecified set of feature functions, that is,

$$\Psi_a(\mathbf{y}_a, \mathbf{x}_a) = \exp\left\{ \sum_{k=1}^{K(A)} \theta_{ak} f_{ak}(\mathbf{y}_a, \mathbf{x}_a) \right\}, \tag{2.23}$$

where both the feature functions $f_{ak}$ and the weights $\theta_{ak}$ are indexed by the factor index $a$ to emphasize that each factor has its own set of weights. In general, each factor is permitted to have a different set of feature functions as well. Notice that if $\mathbf{x}$ and $\mathbf{y}$ are discrete, then the log-linear assumption (2.23) is no additional restriction, because we could choose $f_{ak}$ to be indicator functions for each possible assignment $(\mathbf{y}_a, \mathbf{x}_a)$, similarly to the way in which we converted an HMM into a linear-chain CRF.

Putting together (2.22) and (2.23), the conditional distribution for a CRF with log-linear factors can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\Psi_A \in F} \exp\left\{ \sum_{k=1}^{K(A)} \theta_{ak} f_{ak}(\mathbf{y}_a, \mathbf{x}_a) \right\}. \tag{2.24}$$

In addition, most applied models rely extensively on parameter tying. For example, in the linear-chain case, typically 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*, which is a set of factors sharing a set of feature functions $\{f_{pk}(\mathbf{x}_c, \mathbf{y}_c)\}_{k=1}^{K(p)}$ and a corresponding set of parameters $\theta_p \in \Re^{K(p)}$. A CRF that uses clique templates can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p), \tag{2.25}$$

where each templated factor is parameterized as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = \exp\left\{ \sum_{k=1}^{K(p)} \theta_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \right\}, \tag{2.26}$$

and the normalization function is

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p). \tag{2.27}$$

This notion of clique template specifies both repeated structure and parameter tying in the model. For example, in a linear-chain CRF, 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 $\Psi_t$ to have a separate set of parameters for each $t$, similar to a non-homogeneous HMM, this would be accomplished using $T$ templates, by taking $\mathcal{C} = \{C_t\}_{t=1}^{\mathrm{T}}$, where $C_t = \{\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)\}$.

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, which we will mention only briefly. For example, *dynamic conditional*

*random fields* [140] are sequence models which allow multiple labels at each time step, rather than a single label, in a manner analogous to dynamic Bayesian networks. Second, *relational Markov networks* [142] are a type of general CRF in which the graphical structure and parameter tying are determined by an SQL-like syntax. *Markov logic networks* [113, 128] 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* [87] 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. The notion of clique template that we present in this survey is inspired by those in Taskar et al. [142], Sutton et al. [140], Richardson and Domingos [113], and McCallum et al. [87].

## 2.5 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. The main tradeoff is the classic one, that using larger features sets can lead to better prediction accuracy, because the final decision boundary can be more flexible, but on the other hand larger feature sets require more memory to store all the corresponding parameters, and could have worse prediction accuracy due to overfitting.

**Label-observation features.** First, when the label variables are discrete, the features $f_{pk}$ of a clique template $C_p$ are ordinarily chosen to have a particular form:

$$f_{pk}(\mathbf{y}_c, \mathbf{x}_c) = \mathbf{1}_{\{\mathbf{y}_c = \tilde{\mathbf{y}}_c\}} q_{pk}(\mathbf{x}_c). \tag{2.28}$$

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. We call features that have this form *label-observation features*. 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*."

**Unsupported Features.** The use of label-observation features can result in a large number of parameters. For example, in the first large-scale application of CRFs, Sha and Pereira [125] use 3.8 million binary features in their best model. Many of these features never occur in the training data; they are always zero. The reason this happens is that some observation functions occur only with a small set of labels. For example, in a task of identifying named entities, the feature "word $x_t$ is *with* and label $y_t$ is CITY-NAME" is unlikely to ever have a value of 1 in the training data. We call these *unsupported features*. Perhaps surprisingly, these features can be useful. This is because they can be given a negative weight that prevents the spurious label (e.g. CITY-NAME) from being assigned high probability. (Decreasing the score of label sequences that do not occur in the training data will tend to make the label sequence that does occur more likely, so the parameter estimation procedures that we describe later do in fact assign negative weights to such features.) Including unsupported features typically results in slight improvements in accuracy, at the cost of greatly increasing the number of parameters in 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, which could be called the "unsupported features trick." The idea is that many unsupported features are not useful because the model is unlikely to make mistakes that cause them to be active. For example, the "with" feature above is unlikely to be useful, because "with" is a common word that will be strongly associated with the OTHER (not a named entity) label.

To reduce the number of parameters, we would like to include only those unsupported features that will correct likely mistakes. A simple way to do this is: First train a CRF without any unsupported features, stopping after a few iterations so that the model is not fully trained. Then add unsupported features for all cliques in which the model does not already have the correct answer with high confidence. In the example above, we would add the "with" feature if we found a training instance $i$ and sequence position $t$ in which $x_t^{(i)}$ is *with* and $y_t^{(i)}$ is not CITY-NAME and $p(y_t = \text{CITY-NAME}|\mathbf{x}_t^{(i)})$ is larger than some $\epsilon > 0$.

**Edge-Observation and Node-Observation Features.** In order to reduce the number of features in the model, we can choose to use label-observation features for certain clique templates but not for others. The two most common types of label-observation features are *edge-observation features* and *node-observation features*. Consider a linear chain CRF that has $M$ observation functions $\{q_m(\mathbf{x})\}$, for $m \in \{1, 2, \ldots M\}$. If edge-observation features are used, then the factor for every transition can depend on all of the observation functions, so that we can use features like "word $x_t$ is *New*, label $y_t$ is LOCATION and label $y_{t-1}$ is LOCATION." This can lead to a large number of parameters in the model, which can have disadvantages of memory usage and propensity to overfitting. One way to reduce the number of parameters is to use node-observation features instead. With this style of features, the transition factors can no longer depend on the observation functions. So we would be allowed features like "label $y_t$ is LOCATION and label $y_{t-1}$ is LOCATION" and "word $x_t$ is *New* and label $y_t$ is LOCATION," but we would not be able to use a feature that depends on $x_t$, $y_t$, and $y_{t-1}$ all at once. Both edge-observation features and node-observation features are described formally in Table 2.1. As usual, which of these two choices is preferable depends on the individual problem, such as the number of observation functions and the size of the data set.

**Boundary Labels.** A final choice is how to handle labels on the boundaries, e.g., at the start or end of a sequence or at the edge of an image. Sometimes boundary labels will have different characteristics than other labels. For example, in the middle of a sentence capitalization usually indicates a proper noun, but not at the beginning

Table 2.1. Edge-observation features versus node-observation features.

| | |
|---|---|
| Edge-observation features: | |
| $f(y_t, y_{t-1}, \mathbf{x}_t) = q_m(\mathbf{x}_t)\mathbf{1}_{\{y_t=y\}}\mathbf{1}_{\{y_{t-1}=y'\}}$ | $\forall y, y' \in \mathcal{Y}, \forall m$ |
| $f(y_t, \mathbf{x}_t) = q_m(\mathbf{x}_t)\mathbf{1}_{\{\mathbf{y}_t=y\}}$ | $\forall y \in \mathcal{Y}, \forall m$ |
| Node-observation features: | |
| $f(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=y\}}\mathbf{1}_{\{y_{t-1}=y'\}}$ | $\forall y, y' \in \mathcal{Y}$ |
| $f(y_t, \mathbf{x}_t) = q_m(\mathbf{x}_t)\mathbf{1}_{\{y_t=y\}}$ | $\forall y \in \mathcal{Y}, \forall m$ |

of the sentence. A simple way to represent this is to begin the label sequence of each sentence with a special START label, which allows learning any special characteristics of boundary labels. For example, if edge-observation features are also used, then a feature like "$y_{t-1}$ = START and $y_t$ = PERSON and word $x_t$ is capitalized," can represent the fact that capitalization is not a reliable indicator at the beginning of a sentence.

**Feature Induction.** The "unsupported features trick" described above is a poor man's version of feature induction. McCallum [83] 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 5.1.1. Lavergne et al. [65] find that in the most favorable cases $L_1$ finds models in which only 1% of the full feature set is nonzero, 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.

**Categorical Features.** 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 and then binary features as above.

**Features from Different Time Steps.** Although our notation $f(y_t, y_{t-1}, \mathbf{x}_t)$ for features obscures this, it is usually important for the features to depend on information not only from the nearest label but also from neighboring labels as well. An example of such a feature is "word $x_{t+2}$ is *Times* and label $y_t$ is ORGANIZATION," which could be useful for identifying the name of the New York Times newspaper. It can be useful to include conjunctions of features from neighboring time steps as well, e.g., "words $x_{t+1}$ and $x_{t+2}$ are *York Times*."

**Features as Backoff.** 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 benefit similar to that of backoff in language modelling, 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 5.1.1) when using redundant features because it is the penalty on large weights that encourages the weight to be spread across the overlapping features.

**Features as Model Combination.** Another interesting type of feature can be the results of simpler methods for the same task. For example, if one already has a simple rule-based system for a task (e.g., with rules like "any string of digits between 1900 and 2100 is a year"), the predictions of that system can be used as an observation function for a CRF. Another example is gazetteer features, which are observation functions based on predefined lists of terms, e.g., "$q(x_t) = 1$ if $x_t$ appears in a list of city names obtained from Wikipedia."

A more complex example is to use the output of a generative model as input to the discriminative model. For example, one could use a feature like $f_t(y, \mathbf{x}) = p_{\text{HMM}}(y_t = y|\mathbf{x})$, where $p_{\text{HMM}}$ denotes the marginal

probability of label $y_t = y$ from an HMM trained on a similar data set. It is probably not a good idea to train the HMM and the CRF-with-HMM-feature on the same data set, because the HMM is expected to perform very well on its own training set, which could perhaps cause the CRF to rely on it too much. This technique can be useful if the goal is to improve on a previous system for the same task. Bernal et al.[7] is a good example of doing this in the concept of identifying genes in DNA sequences.

A related idea is to cluster the inputs $\mathbf{x}_t$, e.g., cluster all of the words in the corpus by whatever method you like, and then use the cluster label for word $x_t$ as an additional feature. This sort of feature was used to good effect by Miller et al. [90].

**Input-Dependent Structure.** In a general CRF, it is sometimes useful to allow the structure of the graph for $p(\mathbf{y}|\mathbf{x})$ depend on the input $\mathbf{x}$. A simple example of this is the "skip-chain CRF" [37, 117, 133] which has been used in text applications. The idea behind the skip-chain CRF is that whenever the same word appears twice in the same sentence, we would like to encourage both occurrences of the word to have the same label. So we add an edge between the labels of the corresponding words. This results in a graphical structure over $\mathbf{y}$ that depends on the inputs $\mathbf{x}$.

## 2.6 Examples

In this section, we present more details about two example applications of CRFs. The first is a linear-chain CRF for a natural language text, and the second is a grid-structured CRF for computer vision.

### 2.6.1 Named-Entity Recognition

*Named-entity recognition* is the problem of segmenting and classifying proper names, such as names of people and organization, in text. Since we have been using this task as a running example throughout the section, now we will describe a CRF for this problem in more detail.

An *entity* is an individual person, place, or thing in the world, while a *mention* is a phrase of text that refers to an entity using a proper name. The problem of named-entity recognition is in part one

of segmentation because mentions in English are often multi-word, e.g., *The New York Times*, *the White House*. The exact entity types that are of interest vary across different settings of the problem. For example, in the CoNLL 2003 shared task [121], entities are people, locations, organizations, and miscellaneous entities. In biomedical domains [55, 51], on the other hand, the interest is in representing information from the molecular biology literature, so entities can include genes, proteins, and cell lines. For this example, we will consider the CoNLL 2003 shared task, because it is a good representative example of early applications of CRFs.

The data of the CoNLL 2003 shared task consists of news articles from English and German. The English articles are newswire articles from Reuters taken from between 1996 and 1997. The English data consists of a training set of 946 news articles comprising 203,621 tokens, a development set of 216 articles and 51,362 tokens, and a test set of 231 articles comprising 46,435 tokens. Each of the articles have been manually annotated to indicate the locations and types of all of the named entities. An example sentence is *U.N. official Ekeus heads for Baghdad*; in this sentence, the token *U.N.* is labeled as a organization, *Ekeus* a person, and *Baghdad* a location.

In order to represent this as a sequence labeling problem, we need to convert the entity annotations into a sequence of labels. This labelling needs to handle the fact that multi-word phrases (like *the New York Times*) can refer to a single entity. There is a standard trick for this: we use one type of label (B-???) for the first word of a mention, another type of label (I-???) for any subsequent words in the mention, and finally a label O for words that do not reference any named entity. This label scheme, which is called BIO notation, has the advantages that it can segment two different entities of the same type that occur side-by-side, e.g., *Alice gave Bob Charlie's ball*, and that the B-??? labels can have different weights than the corresponding I-??? labels. There are other schemes apart from BIO notation for converting the segmentation problem to an entity labelling problem, but we will not discuss them here.

In the CoNLL 2003 data set, there are four types of entities: people (PER), organizations (ORG), locations (LOC), and other types of

entities (MISC). So we need 9 labels

$$\mathcal{Y} = \{\text{B-PER}, \text{I-PER}, \text{B-LOC}, \text{I-LOC}, \text{B-ORG}, \text{I-ORG}, \text{B-MISC}, \text{I-MISC}, \text{O}\}$$

With this labeling, our example sentence looks like:

| $t$ | $y_t$ | $\mathbf{x}_t$ |
|---|---|---|
| 0 | B-ORG | U.N. |
| 1 | O | official |
| 2 | B-PER | Ekeus |
| 3 | O | heads |
| 4 | O | for |
| 5 | B-LOC | Baghdad |

To define a linear chain CRF for this problem, we need to choose the set $\mathcal{F}$ of feature functions $f_k(y_t, y_{t-1}, \mathbf{x}_t)$. The simplest choice is the HMM-like feature set that we have already described. In this feature set there are two kinds of features. The first kind we will call *label–label* features, which are:

$$f_{ij}^{\text{LL}}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{y_{t-1}=j\}}\forall i, j \in \mathcal{Y}. \tag{2.29}$$

For this problem, there are 9 different labels, so there are 81 label–label features. The second kind are *label-word* features, which are

$$f_{iv}^{\text{LW}}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{\mathbf{x}_t=v\}}\forall i \in \mathcal{Y}, v \in \mathcal{V}, \tag{2.30}$$

where $\mathcal{V}$ is the set of all unique words that appear in the corpus. For the CoNLL 2003 English data set, there are 21,249 such words, so there are 191,241 label-word features. Most of these features will not be very useful, e.g., they will correspond to words that occur only once in the training set. To see the relationship to (2.18), our full set of features is $\mathcal{F} = \{f_{ij}^{\text{LL}} | \forall i, j \in \mathcal{Y}\} \cup \{f_{iv}^{\text{LW}} | \forall i \in \mathcal{Y}, v \in \mathcal{V}\}$.

A practical NER system will use much more complex features than this. For example, when predicting the label of the word *Baghdad*, it can be helpful to know that the previous word was *for*. To represent this in our notation, we augment each of the vectors $\mathbf{x}_t$ to include the neighboring words, i.e., each vector $\mathbf{x}_t = (x_{t0}, x_{t1}, x_{t2})$, where $x_{t1}$ is the identity of the word at position $t$, and $x_{t0}$ and $x_{t2}$ are the preceding and succeeding words, respectively. The beginning and end of the sequence

are padded with special ⟨START⟩ and ⟨END⟩ tokens. Now our example sentence becomes

| $t$ | $y_t$ | $\mathbf{x}_t$ |
|---|---|---|
| 0 | B-ORG | (⟨START⟩, U.N., official) |
| 1 | O | (U.N., official, Ekeus) |
| 2 | B-PER | (official, Ekeus, heads) |
| 3 | O | (Ekeus, heads, for) |
| 4 | O | (heads, for, Baghdad) |
| 5 | B-LOC | (for, Baghdad, ⟨END⟩) |

To construct a CRF, we retain the label–label features as before, but now we have three different kinds of label-word features:

$$f_{iv}^{\mathrm{LW}0}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{x_{t0}=v\}} \quad \forall i \in \mathcal{Y}, v \in \mathcal{V}$$
$$f_{iv}^{\mathrm{LW}1}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{x_{t1}=v\}} \quad \forall i \in \mathcal{Y}, v \in \mathcal{V}$$
$$f_{iv}^{\mathrm{LW}2}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{x_{t2}=v\}} \quad \forall i \in \mathcal{Y}, v \in \mathcal{V}.$$

However, we wish to use still more features than this, which will depend mostly on the word $x_t$. So we will add label-observation features, as decribed in Section 2.5. To define these, we will define a series of observation functions $q_b(x)$ that take a single word $x$ as input. For each observation function $q_b$, the corresponding label-observation features $f_{ib}^{\mathrm{LO}}$ have the form:

$$f_{ib}^{\mathrm{LO}}(y_t, y_{t-1}, \mathbf{x}_t) = \mathbf{1}_{\{y_t=i\}} q_b(\mathbf{x}_t) \quad \forall i \in \mathcal{Y}. \tag{2.31}$$

When McCallum and Li [86] applied a CRF to this data set, they used a large set of observation functions, some of which are listed in Table 2.2. All of these are binary functions. The example sentence with this third feature set is given in Table 2.3. To display the feature set, we list the observation functions which are nonzero for each word.

These features are typical of those that are used in CRFs for text applications, but they should not be taken as the best possible feature set for this problem. For this particular task, Chieu and Ng [20] had an especially good set of features, during the competition was the best result obtained by a single model (i.e., without using ensemble methods).

Table 2.2. A subset of observation functions $q_s(\mathbf{x}, t)$ for the CoNLL 2003 Engilsh named-entity data, used by Mccallum and Li [86].

| | | |
|---|---|---|
| W=$v$ | $w_t = v$ | $\forall v \in \mathcal{V}$ |
| T=$j$ | part-of-speech tag for $w_t$ is $j$ (as determined by an automatic tagger) | $\forall$POS tags $j$ |
| P=I−$j$ | $w_t$ is part of a phrase with syntactic type $j$ (as determined by an automatic chunker) | |
| Capitalized | $w_t$ matches `[A-Z][a-z]+` | |
| Allcaps | $w_t$ matches `[A-Z][A-Z]+` | |
| EndsInDot | $w_t$ matches `[^\.]+.*\.` | |
| | $w_t$ contains a dash | |
| | $w_t$ matches `[A-Z]+[a-z]+[A-Z]+[a-z]` | |
| Acro | $w_t$ matches `[A-Z][A-Z\\.]*\\.[A-Z\\.]*` | |
| Stopword | $w_t$ appears in a hand-built list of stop words | |
| CountryCapital | $w_t$ appears in list of capitals of countries | |
| ⋮ | many other lexicons and regular expressions | |
| $q_k(\mathbf{x}, t+\delta)$ for all $k$ and $\delta \in [-1, 1]$ | | |

Table 2.3. The example sentence *U.N. official Ekeus heads for Baghdad* converted into a sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_5$ of feature vectors, using the set of observation functions defined in Table 2.2. Each row lists the names of the observation functions that are nonzero.

| $t$ | $y_t$ | Active observation functions |
|---|---|---|
| 1 | B-ORG | P=I-NP@1 W=⟨START⟩@-1 INITCAP P=I-NP T=NNP T=NN@1 ACRO ENDSINDOT W=official@1 W=U.N. |
| 2 | O | P=I-NP@1 INITCAP@1 P=I-NP T=JJ@1 CAPITALIZED@1 T=NNP@-1 P=I-NP@-1 INITCAP@-1 T=NN ENDSINDOT@-1 ACRO@-1 W=official W=U.N.@-1W=Ekeus@1 |
| 3 | B-PER | P=I-NP@1 INITCAP P=I-NP P=I-NP@-1 CAPITALIZED T=JJ T=NN@-1 T=NNS@1 W=official@-1 W=heads@1 W=Ekeus |
| 4 | O | P=I-NP P=I-NP@-1 INITCAP@-1 STOPWORD@1 T=JJ@-1 CAPITALIZED@-1 T=IN@1 P=I-PP@1 T=NNS W=for@1 W=heads W=Ekeus@-1 |
| 5 | O | T=NNP@1 P=I-NP@1 INITCAP@1 LOC@1 CAPITALIZED@1 P=I-NP@-1 STOPWORD COUNTRYCAPITAL@1 P=I-PP T=IN T=NNS@-1 W=for W=Baghdad@1 W=heads@-1 |
| 6 | B-LOC | INITCAP P=I-NP T=NNP CAPITALIZED STOPWORD@-1 T=.@1 P=O@1 PUNC@1 W=⟨END⟩@1 COUNTRYCAPITAL T=IN@-1 P=I-PP@-1 W=for@-1 W=Baghdad |

### 2.6.2 Image Labelling

Many different CRF topologies have been used for computer vision. As an example application, we may wish to classify areas of a given input image according to whether they are foreground or background, they are a manmade structure or not [61, 62], or whether they are sky, water, vegetation, etc. [49].

More formally, let the vector $\mathbf{x} = (x_1, x_2, \ldots x_T)$ represent an image of size $\sqrt{T} \times \sqrt{T}$, represented as a single vector. That is, $\mathbf{x}_{1:\sqrt{T}}$ gives the pixels in row 1, $\mathbf{x}_{\sqrt{T}+1:2\sqrt{T}}$ those in row 2, etc. Each individual $x_i$ represents the value of an individual pixel in the image. To keep the example simple, think of the image as being in black and white, so that $x_i$ will be a real number between 0 and 256 that gives the intensity of the pixel at location $i$. (The ideas here extend in a simple way to color images.) The goal is to predict a vector $\mathbf{y} = (y_1, y_2, \ldots, y_T)$, where each $y_i$ gives the label of site $i$, e.g., $+1$ if the pixel is a manmade structure, $-1$ otherwise.

There is an enormous variety of image features that have been proposed in the computer vision literature. As a simple example, given a pixel at location $i$, we can compute a histogram of pixel intensities in a $5 \times 5$ box of pixels centered at $i$, and then include the count of each bin in the histogram as features. Typically more complex features are used, for example, features that use gradients of the image, texton features [127], and SIFT features [77]. Importantly, these features do not depend on the pixel $x_i$ alone; most interesting features depend on a region of pixels, or even the entire image.

A basic characteristic of images is that neighboring pixels tend to have the same label. One way to incorporate this idea into our model is to place a prior distribution on $\mathbf{y}$ that encourages "smooth" predictions. The most common such prior in computer vision is a grid-structured undirected graphical model that is typically called a *Markov random field* [10]. An MRF is a generative undirected model with two types of factors: one type that associates each label $y_i$ with its corresponding pixel $x_i$, and another type that encourages neighboring labels $y_i$ and $y_j$ to agree.

More formally, let $\mathcal{N}$ define the neighborhood relationship among pixels, that is, $(i,j) \in \mathcal{N}$ if any only if $x_i$ and $x_j$ are neighboring pixels in the image. Often $\mathcal{N}$ will be chosen to form a $\sqrt{T} \times \sqrt{T}$ grid. An MRF is a generative model

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{(i,j)\in\mathcal{N}} \Psi(y_i, y_j)$$

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}) \prod_{i=1}^{T} p(x_i|y_i). \tag{2.32}$$

Here $\Psi$ is the factor that encourages smoothness. One common choice is $\Psi(y_i, y_j) = 1$ if $y_i = y_j$ and $\alpha$ otherwise, where $\alpha$ is a parameter that can be learned from data and typically $\alpha < 1$. The motivation behind this choice of $\Psi(y_i, y_j)$ is that if $\alpha < 1$, then efficient inference algorithms are available to maximize $\log p(\mathbf{y}, \mathbf{x})$. The distribution $p(x_i|y_i)$ is a class-conditional distribution over pixel values, for example, a mixture of Gaussians over $x_i$.

A disadvantage of this MRF is that it is difficult to incorporate features over regions of pixels, of the kind discussed earlier, because then $p(\mathbf{x}|\mathbf{y})$ would have complex structure. A conditional model provides a way to address this difficulty.

The CRF that we will describe for this task will be very similar to the MRF, except that it will allow the factors on both the individual sites and the edges to depend on arbitrary features of the image. Let $q(x_i)$ be a vector of features based on a region of the image around $x_i$, for example, using color histograms or image gradients as mentioned above. In addition we will want a feature vector $\nu(x_i, x_j)$ that depends on pairs of sites $x_i$ and $x_j$, so that the model can take into account the similarities and differences between $x_i$ and $x_j$. One way to do this is to define $\nu(x_i, x_j)$ to be the set of all cross-products between features in $q(x_i)$ and $q(x_j)$, i.e., to compute $\nu(x_i, x_j)$, first compute a matrix via the outer product $q(x_i)q(x_j)^\top$ and flattened into into a vector.

We have called the functions $q$ and $\nu$ *features* because this is the terminology most commonly used in computer vision. But in this survey we have been using features that depend both on the inputs $\mathbf{x}$ and the labels $\mathbf{y}$. So we will use $q$ and $\nu$ as observation functions to define

label-observation features in a CRF. The resulting label-observation features are

$$\begin{aligned} f_m(y_i, x_i) &= \mathbf{1}_{\{y_i=m\}} q(x_i) \quad \forall m \in \{0,1\} \\ g_{m,m'}(y_i, y_j, x_i, x_j) &= \mathbf{1}_{\{y_i=m\}} \mathbf{1}_{\{y_j=m'\}} \nu(x_i, x_j) \quad \forall m, m' \in \{0,1\} \\ f(y_i, x_i) &= \begin{pmatrix} f_0(y_i, x_i) \\ f_1(y_i, x_i) \end{pmatrix} \\ g(y_i, y_j, x_i, x_j) &= \begin{pmatrix} g_{00}(y_i, y_j, x_i, x_j) \\ g_{01}(y_i, y_j, x_i, x_j) \\ g_{10}(y_i, y_j, x_i, x_j) \\ g_{11}(y_i, y_j, x_i, x_j) \end{pmatrix} \end{aligned}$$

Using label-observation features has the effect of allowing the model to have a separate set of weights for each label.

To make the example concrete, here is a specific choice for $g$ and $\nu$ that has been used in some prominent applications [14, 119]. Consider the pairwise factor $\Psi(y_i, y_j)$ in the MRF (2.32). Although it is good that $\Psi$ encourages agreement, the way in which it does so is inflexible. If the pixels $x_i$ and $x_j$ have different labels, we expect them to have different intensities in black and white, because different objects tend to have different hues. So if we see a label boundary drawn between two pixels with sharply different intensities, we should be less suprised than if we see if the boundary drawn between identical-looking pixels. Unfortunately, $\Psi$ imposes the same dissimilarity penalty in both cases, because the potential does not depend on the pixel values. To fix this problem, the following choice of features has been proposed [14, 119]

$$\begin{aligned} \nu(x_i, x_j) &= \exp\left\{-\beta(x_i - x_j)^2\right\} \\ g(y_i, y_j, x_i, x_j) &= \mathbf{1}_{\{y_i \neq y_j\}} \nu(x_i, x_j). \end{aligned} \tag{2.33}$$

Putting this all together, the CRF model is

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{ \sum_{i=1}^{T} \theta^\top f(y_i, x_i) + \sum_{(i,j)\in\mathcal{N}} \lambda^\top g(y_i, y_j, x_i, x_j) \right\}, \tag{2.34}$$

where $\alpha \in \Re$, $\theta \in \Re^K$, and $\lambda \in \Re^{K^2}$ are the parameters of the model. The first two terms are analogous to the two types of factors in the

MRF. The first term represents the impact of the local evidence around $x_i$ to label $y_i$. Using the choice of $g$ described in (2.33), the second term encourages neighboring labels to be similar, in a manner that depends on the difference in pixel intensity.

Notice that this is an instance of the general CRF definition given in (2.25), where we have three clique templates, one for each of the three terms in (2.34).

The difference between (2.34) and (2.32) is analogous to the difference between the linear-chain CRF models in Figures 2.6 and 2.5: The pairwise factors now depend on features of the images, rather than simply on the label identity. As an aside, notice that, just as in the case of sequences, the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that results from the MRF model (2.32) is a particular kind of CRF, in that it will have the form (2.34) with $\lambda = 0$.

This simple CRF model can be improved in various ways. First, the feature functions $q$ and $\nu$ can be more complex, for example, taking image shape and texture into account [127], or depending on global characteristics of the image rather than on a local region. Additionally, one could use a more complex graphical structure among the labels than a grid. For example, one can define factors that depend on regions of the labels [49, 56]. For a more in-depth survey about how CRFs and other structured prediction models can be used in computer vision, see Nowozin and Lampert [101].

## 2.7 Applications of CRFs

In addition to the example domains in the previous section, CRFs have been applied to a large variety of other domains, including text processing, computer vision, and bioinformatics. One of the first large-scale applications of CRFs was by Sha and Pereira [125], 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 [86], feature induction for NER [83], shallow parsing [125, 138], identifying protein names in biology abstracts [124], segmenting addresses in Web pages [29], information integration [156], finding semantic roles in text [118],

prediction of pitch accents [47], phone classification in speech processing [48], identifying the sources of opinions [21], word alignment in machine translation [12], citation extraction from research papers [105], extraction of information from tables in text documents [106], Chinese word segmentation [104], Japanese morphological analysis [59], and many others.

In bioinformatics, CRFs have been applied to RNA structural alignment [123] and protein structure prediction [76]. Semi-Markov CRFs [122] are a way to allow more flexible feature functions. In a linear-chain CRF, feature functions $f()$ are restricted to depend only on successive pairs of labels. In a semi-Markov CRF, on the other hand, a feature function can depend on a entire segment of the labelling, that is, a sequence of successive labels that have the same value. This can be useful for certain tasks in information extraction and especially bioinformatics, where for example, one might want features that depend on the length of the segment.

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. [140] 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 [42] present a CRF that learns dependencies between the categories, resulting in improved classification performance. Finally, the skip-chain CRF [133] is a general CRF that represents long-distance dependencies in information extraction.

Another application of general CRFs that has used a different structure been in the problem of proper-noun coreference, that is, of determining which mentions in a document, such as *Mr. President* and *he*, refer to the same underlying entity. McCallum and Wellner [88] learn a distance metric between mentions using a fully-connected CRF in which inference corresponds to graph partitioning. A similar model has been used to segment handwritten characters and diagrams [26, 108].

As mentioned in Section 2.6.2, CRFs have been widely used in computer vision, for example, grid-shaped CRFs for labeling and

segmenting images [49, 61, 127]. Although image labelling is a common application of CRFs in vision, there are other types of probabilistic structure in the vision problem that are interesting to model. One type of structure is the relationships between parts of an object. For example, Quattoni et al. [109] use a tree-structured CRF in which the hope is that the latent variables will recognize characteristic parts of an object; this was an early example of hidden-variable CRFs. An especially successful example of a discriminative models with variables that correspond to different parts of an object is Felzenszwalb et al. [36].

Another type of probabilistic structure in computer vision occurs because similar objects appear in different images. For example, Deselaers et al. [33] present a model that simultaneously performs localization of objects in multiple images. They define a CRF in which each random variable specifies the boundaries of a bounding box that contains the object, i.e., each random variable corresponds to an entire image.

An excellent survey on structured prediction for computer vision, including CRFs, is Nowozin and Lampert [101].

Vision is one of more common application areas for CRFs with more loopy graphical structure. That said, general CRFs have also been used for global models of natural language [16, 37, 133].

In some applications of CRFs, efficient dynamic programs exist even though the graphical model is difficult to specify. For example, McCallum et al. [84] 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 [23, 38, 114, 148].

## 2.8 Notes on Terminology

Different aspects of the theory of graphical models have been developed independently in different research areas, so many of the concepts in this section 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. (We gave our definition of it in Section 2.2.).

# 3

# Overview of Algorithms

The next two sections will discuss inference and parameter estimation for CRFs. *Parameter estimation* is the problem of finding a set of parameters $\theta$ so that the resulting distribution $p(\mathbf{y}|\mathbf{x},\theta)$ best fits a set of training examples $\mathcal{D} = \{\mathbf{x}^{(i)},\mathbf{y}^{(i)}\}_{i=1}^{N}$ for which both the inputs and outputs are known. Intuitively, what we want to accomplish during parameter estimation is that if we look at any of the training inputs $\mathbf{x}^{(i)}$, the model's distribution over outputs $p(\mathbf{y}|\mathbf{x}^{(i)},\theta)$ should "look like" the true output $\mathbf{y}^{(i)}$ from the training data.

One way to quantify this intuition is to consider the feature functions that are used to define the model. Consider a linear-chain CRF. For each feature $f_k(y_t, y_{t-1}, \mathbf{x}_t)$, we would like the total value of $f_k$ that occurs in the data to equal the total value of $f_k$ that we would get by randomly selecting an input sequence $\mathbf{x}$ from the training data following by sampling $\mathbf{y}$ from the conditional model $p(\mathbf{y}|\mathbf{x},\theta)$. Formally this requirement yields for all $f_k$ that

$$\sum_{i=1}^{N}\sum_{t=1}^{T} f_k\big(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}\big) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{y,y'} f_k(y,y',\mathbf{x}_t^{(i)})\, p(y_t = y, y_{t-1} = y'|\mathbf{x}^{(i)}).$$

Remarkably, this system of equations can be obtained as the gradient of an objective function over parameters. This connection is useful because once we have an objective function we can optimize it using standard numerical techniques. The objective function that has this property is the likelihood

$$\begin{aligned}\ell(\theta) &= p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \theta) \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K}\theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N}\log Z(\mathbf{x}^{(i)}),\end{aligned}$$

which is the probability of training data under the model, viewed as a function of the parameter vector.

A standard way of training CRFs is maximum likelihood, in which we seek the parameters $\hat{\theta}_{\text{ML}} = \sup_\theta \ell(\theta)$. The intuition behind maximum likelihood is that $\hat{\theta}_{\text{ML}}$ is the parameter setting under which the observed data is most likely. To connect this with the discussion about matching expectations, take the partial derivative of the likelihood with respect to some parameter $\theta_k$ and set it to zero. This exactly yields the matching conditions on the feature expectations that we began with.

Although we have illustrated the idea behind maximum likelihood using linear chain CRFs the same ideas carry over to general CRFs. For general CRFs, instead of marginals $p(y_t, y_{t-1}|\mathbf{x}, \theta)$ over neighboring variables in a chain, computing the likelihood gradient requires marginals $p(\mathbf{y}_a|\mathbf{x}, \theta)$ over sets of variables $Y_a$ in a general graphical model.

Computing the marginal distributions that are required for parameter estimation can be computationally challenging. This is the task of *probabilistic inference*. In general, the goal of inference is to compute predictions over the output $\mathbf{y}$ from a given $\mathbf{x}$ for a single fixed value of $\theta$. There are two specific inference problems that we will focus on:

- Computing marginal distributions $p(\mathbf{y}_a|\mathbf{x}, \theta)$ over a subset $Y_a$ of output variabels. Usually the domain $Y_a$ of the marginals consists either of a single variable or of a set of neighboring variables that share a factor. The algorithms for this problem will usually also compute as a byproduct the normalization function $Z(\mathbf{x})$ which appears in the likelihood.

- Computing the labeling $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}, \theta)$, which is the single most likely labeling of a new input $\mathbf{x}$.

The marginals $p(\mathbf{y}_a|\mathbf{x}, \theta)$ and the normalization function $Z(\mathbf{x})$ are used for parameter estimation. Some parameter estimation methods, like maximum likelihood when optimized by limited memory BFGS, require both the marginals and the normalization function. Other parameter estimation methods, like stochastic gradient descent, require only the marginals. The Viterbi assignment $\mathbf{y}^*$ is used for assigning a sequence of labels to a new input that was not seen during training.

These inferential tasks can be solved using standard techniques from graphical models. In tree-structured models, these quantities can be computed exactly, while in more general models we typically resort to approximations.

The next two sections discuss inference and parameter estimation both for linear-chain and general CRFs. First, in Section 4, we discuss inference methods for CRFs, including exact methods for tree-structured CRFs and approximate methods for more general CRFs. In some sense, because a CRF is a type of undirected graphical model, this largely recaps inference methods for standard graphical models, but we focus on methods that are most appropriate for CRFs. Then, in Section 5, we discuss parameter estimation. Although the maximum likelihood procedure is conceptually simple, it can require expensive computations. We describe both the standard maximum likelihood procedure, ways to combine maximum likelihood with approximate inference, and other approximate training methods that can improve scalability in both the number of training instances and in the complexity of the graphical model structure of the CRF.

# 4

# 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 Section 5, estimation of the parameters typically requires that we compute the marginal distribution over subsets of labels, such as over node marginals $p(y_t|\mathbf{x})$ and edge marginals $p(y_t, y_{t-1}|\mathbf{x})$. 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 maximization for addition.

For discrete variables the marginals could be computed by brute-force summation, but the time required to do so 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 4.1. These algorithms are a special case of the more general belief propagation algorithm for tree-structured graphical models (Section 4.2.2). For more complex models, approximate inference is necessary.

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 [57, 79]. 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 5.1.1 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 Section 5, when we discuss parameter estimation, but it is worth mentioning them here because they strongly influence the choice of inference algorithm.

## 4.1 Linear-Chain CRFs

In this section, we briefly review the standard inference algorithms for HMMs, the forward–backward and Viterbi algorithms, and describe how they can be applied to linear-chain CRFs. A survey on these algorithms in the HMM setting is provided by Rabiner [111]. Both of these algorithms are special cases of the belief propagation algorithm described in Section 4.2.2, but we discuss the special case of linear chains in detail both because it may help to make the later 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:

$$\Psi_t(j, i, x) \stackrel{\text{def}}{=} p(y_t = j | y_{t-1} = i) p(x_t = x | y_t = j). \tag{4.1}$$

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:

$$p(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t) \tag{4.2}$$

$$= \sum_{y_{\mathrm{T}}} \sum_{y_{\mathrm{T}-1}} \Psi_{\mathrm{T}}(y_{\mathrm{T}}, y_{\mathrm{T}-1}, x_{\mathrm{T}}) \sum_{y_{\mathrm{T}-2}} \Psi_{\mathrm{T}-1}(y_{\mathrm{T}-1}, y_{\mathrm{T}-2}, x_{\mathrm{T}-1}) \sum_{y_{\mathrm{T}-3}} \cdots \tag{4.3}$$

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* $\alpha_t$, each of which is a vector of size $M$ (where $M$ is the number of states) which represents the intermediate sums. These are defined as:

$$\alpha_t(j) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle 1 \ldots t \rangle}, y_t = j) \tag{4.4}$$

$$= \sum_{\mathbf{y}_{\langle 1 \ldots t-1 \rangle}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{4.5}$$

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

$$\alpha_t(j) = \sum_{i \in S} \Psi_t(j, i, x_t) \alpha_{t-1}(i), \tag{4.6}$$

with initialization $\alpha_1(j) = \Psi_1(j, y_0, x_1)$. (Recall from (2.10) 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 (4.6) to obtain (4.3). A formal proof would use induction.

The backward recursion is exactly the same, except that in (4.3), we push in the summations in reverse order. This results in the definition

$$\beta_t(i) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle t+1 \ldots \mathrm{T} \rangle} | y_t = i) \tag{4.7}$$

$$= \sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{\mathrm{T}} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{4.8}$$

and the recursion

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}(j, i, x_{t+1}) \beta_{t+1}(j), \tag{4.9}$$

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

To compute the marginal distributions $p(y_{t-1}, y_t|\mathbf{x})$, which will prove necessary for parameter estimation, we combine the results of the forward and backward recursions. This can be seen from either the probabilistic or the factorization perspectives. First, taking a probabilistic viewpoint we can write

$$p(y_{t-1}, y_t|\mathbf{x}) = \frac{p(\mathbf{x}|y_{t-1}, y_t)p(y_{t-1}, y_t)}{p(\mathbf{x})} \tag{4.10}$$

$$= \frac{p(\mathbf{x}_{\langle 1 \ldots t-1 \rangle}, y_{t-1})p(y_t|y_{t-1})p(x_t|y_t)p(\mathbf{x}_{\langle t+1 \ldots \mathrm{T} \rangle}|y_t)}{p(\mathbf{x})} \tag{4.11}$$

$$= \frac{1}{p(\mathbf{x})} \alpha_{t-1}(y_{t-1}) \Psi_t(y_t, y_{t-1}, x_t) \beta_t(y_t), \tag{4.12}$$

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

$$\begin{aligned} p(y_{t-1}, y_t|\mathbf{x}) = \frac{1}{p(\mathbf{x})} & \Psi_t(y_t, y_{t-1}, x_t) \\ & \times \left( \sum_{\mathbf{y}_{\langle 1 \ldots t-2 \rangle}} \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right) \\ & \times \left( \sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right). \end{aligned} \tag{4.13}$$

Then, by substituting the definitions of $\alpha$ and $\beta$, we obtain the same result as before, namely

$$p(y_{t-1}, y_t|\mathbf{x}) = \frac{1}{p(\mathbf{x})} \alpha_{t-1}(y_{t-1}) \Psi_t(y_t, y_{t-1}, x_t) \beta_t(y_t). \tag{4.14}$$

The factor of $1/p(\mathbf{x})$ acts as a normalizing constant for this distribution. We can compute it by using $p(\mathbf{x}) = \beta_0(y_0)$ or $p(\mathbf{x}) = \sum_{i \in S} \alpha_{\mathrm{T}}(i)$.

Putting this together, the *forward–backward* algorithm is: First compute $\alpha_t$ for all $t$ using (4.6), then compute $\beta_t$ for all $t$ using (4.9), and then return the marginal distributions computed from (4.14).

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 (4.3) still works if all the summations are replaced by maximization. This yields the Viterbi recursion. The analogs of the $\alpha$ variables in forward backward are

$$\delta_t(j) \stackrel{\text{def}}{=} \max_{\mathbf{y}_{\langle 1 \ldots t-1 \rangle}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}). \tag{4.15}$$

And these can be computed by the analogous recursion

$$\delta_t(j) = \max_{i \in S} \Psi_t(j, i, x_t) \delta_{t-1}(i). \tag{4.16}$$

Once the $\delta$ variables have been computed, the maximizing assignment can be computed by a backwards recursion

$$\begin{aligned} y_{\mathrm{T}}^* &= \arg\max_{i \in S} \delta_{\mathrm{T}}(i) \\ y_t^* &= \arg\max_{i \in S} \Psi_t(y_{t+1}^*, i, x_{t+1}) \delta_t(i) \quad \text{for } t < T \end{aligned}$$

The recursions for $\delta_t$ and $y_t^*$ together comprise the *Viterbi algorithm*.

Now that we have described the forward–backward and Viterbi algorithms for HMMs, the generalization to linear-chain CRFs is 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 (2.18) can be rewritten as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}_t), \tag{4.17}$$

where we define

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp\left\{ \sum_k \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \tag{4.18}$$

With that definition, the algorithms for the forward recursion (4.6), the backward recursion (4.9), and the Viterbi recursion (4.16) can be used unchanged for linear-chain CRFs. Only the interpretation is slightly different. In a CRF we no longer have the probabilistic interpretation that $\alpha_t(j) = p(\mathbf{x}_{\langle 1\ldots t\rangle}, y_t = j)$ that we have for HMMs. Instead we define the $\alpha$, $\beta$, and $\delta$ variables using the factorization viewpoints, i.e., we define $\alpha$ as in (4.5), $\beta$ as in (4.8), and $\delta$ as in (4.15). Also the results of the forward and backward recursions are $Z(\mathbf{x})$ instead of $p(\mathbf{x})$, that is, $Z(\mathbf{x}) = \beta_0(y_0)$ and $Z(\mathbf{x}) = \sum_{i \in S} \alpha_{\mathrm{T}}(i)$.

For the marginal distributions, equation (4.14) remains true with the change that $Z(\mathbf{x})$ replaces $p(\mathbf{x})$, that is,

$$p(y_{t-1}, y_t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x_t)\beta_t(y_t). \tag{4.19}$$

$$p(y_t|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_t(y_t)\beta_t(y_t). \tag{4.20}$$

We mention three more specialized inference tasks that can also be solved using direct analogues of the HMM algorithms. First, if we wish to sample independent draws of $\mathbf{y}$ from the posterior $p(\mathbf{y}|\mathbf{x})$, we can use the forward algorithm combined with a backward sampling pass, exactly as in 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 apply standard algorithms from HMMs [129]. Finally, sometimes it is useful to compute a marginal probability $p(\mathbf{y}_S|\mathbf{x})$ over a (possibly non-contiguous) set of nodes with indices $S \subset [1, 2, \ldots T]$. 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 [30].

## 4.2 Inference in Graphical Models

There are a number of exact inference algorithms for general graphical models. 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 groups 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 [57].

Because of the complexity of exact inference, 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 approximation that most closely matches the intractable marginals of interest. Generally, Monte Carlo algorithms are unbiased in the sense that they are 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. For a good reference on MCMC, see Robert and Casella [116], and for variational approaches, see Wainwright and Jordan [150].

The material in this section is in no way specific to CRFs, but holds for any distribution that factorizes according to some factor graph, whether it be a joint distribution $p(\mathbf{y})$ or a conditional distribution $p(\mathbf{y}|\mathbf{x})$ like a CRF. To emphasize this, and to lighten the notation, in this section we will drop the dependence on $\mathbf{x}$ and talk about inference for joint distributions $p(\mathbf{y})$ that factorize according to some factor graph $G = (V, F)$, i.e.,

$$p(\mathbf{y}) = Z^{-1} \prod_{a \in F} \Psi_a(\mathbf{y}_a).$$

To carry this discussion to CRFs, simply replace $\Psi_a(\mathbf{y}_a)$ in the above equation with $\Psi_a(\mathbf{y}_a, \mathbf{x}_a)$, and likewise modify $Z$ and $p(\mathbf{y})$ to depend on $\mathbf{x}$. This point is not just notational but also affects the design of implementations: inference algorithms can be implemented for generic factor graphs in such a way that the inference procedure does not know if it is dealing with an undirected joint distribution $p(\mathbf{y})$, a CRF $p(\mathbf{y}|\mathbf{x})$, or even a directed graphical model.

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 section, we focus on describing the inference algorithms themselves, whereas in Section 5 we discuss their application to CRFs.

### 4.2.1 Markov Chain Monte Carlo

Currently the most popular type of Monte Carlo method for complex models is *Markov Chain Monte Carlo* (MCMC) [116]. Rather than attempting to approximate a marginal distribution $p(y_s)$ directly, MCMC methods generate approximate samples from the joint distribution $p(\mathbf{y})$. MCMC methods work by constructing a Markov chain, whose state space is the same as that of $Y$, in a careful way so that when the chain is simulated for a long time, the distribution over states of the chain is approximately $p(y_s)$. Suppose that we want to approximate the expectation of some function $f(\mathbf{y})$ over the distribution $p(\mathbf{y})$. 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:

$$\sum_{\mathbf{y}} p(\mathbf{y}) f(\mathbf{y}) \approx \frac{1}{M} \sum_{j=1}^{M} f(\mathbf{y}^j). \tag{4.21}$$

For example, we will see in the next section that expectations of this form are used during CRF training.

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 $Y_s$. Sample $\mathbf{y}_s^{j+1}$ from the distribution $p(y_s|\mathbf{y}_{\setminus s}, \mathbf{x})$.
(3) Return the resulting value of $\mathbf{y}^{j+1}$.

Recall from Section 2.1.1 that the summation $\sum_{\mathbf{y}\setminus y_s}$ to indicate a summation over all possible assignments $\mathbf{y}$ whose value for variable $Y_s$ is equal to $y_s$.

The above procedure defines a Markov chain that can be used to approximate expectations as in (4.21). In the case of a general factor graph, the conditional probability can be computed as

$$p(y_s|\mathbf{y}_{\setminus s}) = \kappa \prod_{a \in F} \Psi_a(\mathbf{y}_a), \tag{4.22}$$

where $\kappa$ is a normalizing constant. (In the following, $\kappa$ will denote a generic normalizing constant which need not be the same across equations.) The normalizer $\kappa$ from (4.22) is much easier to compute than the joint probability $p(\mathbf{y}|\mathbf{x})$, because computing $\kappa$ 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 [78]. The main disadvantage of Gibbs sampling is that it can work poorly if $p(\mathbf{y})$ 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 $p(\mathbf{y})$.

There is an enormous literature on MCMC algorithms. The textbook by Robert and Casella [116] provides an overview. However, MCMC algorithms are not commonly applied in the context of CRFs. 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 5.4.3).

### 4.2.2 Belief Propagation

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

Suppose that the factor graph $G = (V, F)$ is a tree, and we wish to compute the marginal distribution of a variable $Y_s$. The intuition behind BP is that each of the neighboring factors of $Y_s$ makes a multiplicative contribution to its marginal, called a *message*, and each of these messages can be computed separately because the graph is a tree. More formally, for every factor index $a \in N(s)$, denote by $G_a = (V_a, F_a)$ the subgraph of $G$ that contains $Y_s$, $\Psi_a$, and the entire subgraph of $G$ that is "upstream" of $\Psi_a$. By upstream we mean that $V_a$ contains all of the variables that $\Psi_a$ separates from $Y_s$, and $F_a$ all of the factors. This is depicted in Figure 4.1. All of the sets $V_a \backslash Y_s$ for all $a \in N(s)$ are mutually disjoint because $G$ is a tree, and similarly for the $F_a$. This means that we can split up the summation required for the marginal into a product of independent subproblems as:

$$p(y_s) \propto \sum_{\mathbf{y} \backslash y_s} \prod_{a \in F} \Psi_a(\mathbf{y}_a) \tag{4.23}$$

$$= \sum_{\mathbf{y} \backslash y_s} \prod_{a \in N(s)} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b) \tag{4.24}$$

$$= \prod_{a \in N(s)} \sum_{\mathbf{y}_{V_a} \backslash y_s} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b). \tag{4.25}$$

Although the notation somewhat obscures this, notice that the variable $y_s$ is contained in all of the $y_a$, so that it appears on both sides of (4.25).

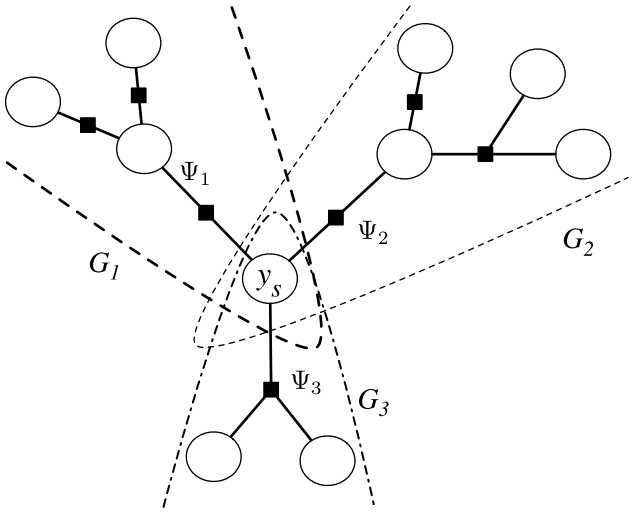


Fig. 4.1 Illustration of how marginal distributions factorize for tree-structured graphs. This factorization is exploited by the belief propagation algorithm (Section 4.2.2).

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

$$m_{as}(y_s) = \sum_{\mathbf{y}_{V_a} \setminus y_s} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b). \tag{4.26}$$

Each $m_{as}$ is just the marginal distribution over the variable $y_s$ for the subgraph $G_a$. The marginal distribution of $y_s$ in the full graph $G$ is the product of the marginals in all the subgraphs. Metaphorically, each $m_{as}(y_s)$ can be thought of as a *message* from the factor $a$ to the variable $Y_s$ that summarizes the impact of the network upstream of $a$ on the marginal probability over $Y_s$. In a similar fashion, we can define messages from variables to factors as

$$m_{sa}(y_s) = \sum_{\mathbf{y}_{V_s}} \prod_{\Psi_b \in F_s} \Psi_b(\mathbf{y}_b). \tag{4.27}$$

Then, from (4.25), we have that the marginal $p(y_s)$ is proportional to the product of all the incoming messages to variable $Y_s$. Similarly, factor marginals can be computed as

$$p(\mathbf{y}_a) \propto \Psi_a(\mathbf{y}_a) \prod_{s \in N(a)} m_{sa}(\mathbf{y}_a). \tag{4.28}$$

Naively computing the messages according to (4.26) is impractical, because the messages as we have defined them require summation over all possible assignments to $y_{V_a}$, and some of the sets $V_a$ will be large. Fortunately, the messages can also be written using a recursion that requires only local summation. The recursion is

$$\begin{aligned} m_{as}(y_s) &= \sum_{\mathbf{y}_a \setminus y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in a \setminus s} m_{ta}(y_t) \\ m_{sa}(y_s) &= \prod_{b \in N(s) \setminus a} m_{bs}(y_s). \end{aligned} \tag{4.29}$$

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 *belief propagation* algorithm [103].

In addition to computing single-variable marginals, we will also wish to compute factor marginals $p(\mathbf{y}_a)$ and joint probabilities $p(\mathbf{y})$ 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 we can use the same decomposition of the marginal as for the single-variable case, and get

$$p(\mathbf{y}_a) = \kappa \Psi_a(\mathbf{y}_a) \prod_{s \in N(a)} m_{sa}(y_s), \tag{4.30}$$

where $\kappa$ 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 $\kappa$ is impractical.

BP can also be used to compute the normalizing constant $Z$. This can be done directly from the propagation algorithm, in an analogous way to the forward–backward algorithm in Section 4.1. Alternatively, there is another way to compute $Z$ from only the approximate marginals at the end of the algorithm. In a tree structured distribution $p(\mathbf{y})$, it can be shown that the joint distribution always

factorizes as

$$p(\mathbf{y}) = \prod_{s \in V} p(y_s) \prod_a \frac{p(\mathbf{y}_a)}{\prod_{t \in a} p(y_t)}. \tag{4.31}$$

For example, in a linear chain this amounts to

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t) \prod_{t=1}^{\mathrm{T}} \frac{p(y_t, y_{t-1})}{p(y_t) p(y_{t-1})}, \tag{4.32}$$

which, after cancelling and rearranging terms, is just another way to write the familiar equation $p(\mathbf{y}) = \prod_t p(y_t|y_{t-1})$. Using this identity, we can compute $p(\mathbf{y})$ for any assignment $p(\mathbf{y})$ from the per-variable and per-factor marginals. This also gives us $Z = p(\mathbf{y})^{-1} \prod_{a \in F} \Psi_a(\mathbf{y}_a)$.

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 4.1). To see this, refer to Figure 4.2. 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 $\alpha_2$ in Section 4.1 corresponds to the product of the two messages $m_{A2}$ and $m_{C2}$ (the dark grey arrows in the figure). The backward message $\beta_2$ corresponds to the message $m_{B2}$ (the light grey arrow in the figure). Indeed, the decomposition of the marginal distribution $p(\mathbf{y}_a)$ in (4.30) generalizes that for the linear chain case in (4.14).

If $G$ is not a tree, the message updates (4.29) 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

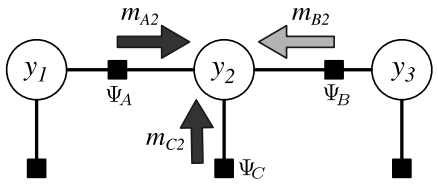


Fig. 4.2 Illustration of the correspondence between forward backward and belief propagation in linear chain graphs. See text for details.

marginals that result from loopy BP as *beliefs* rather than as marginals, and denote them by $q(y_s)$.

There is still the question of what schedule we use to iterate the message updates. In the tree structured case, any propagation schedule will converge to the correct marginal distributions, but this is not true in the loopy case: rather, the schedule that is used to update messages can affect not only the final answer from loopy BP but also whether the algorithm converges at all. A simple choice that works well in practice is to order the message updates randomly, for example, to shuffle the factors via a random permutation, and then for each factor in turn, send all of its outgoing and incoming messages via (4.29). However, more sophisticated schedules can also be effective [35, 135, 152].

Surprisingly, loopy BP can be seen as a variational method for inference, meaning that there exists an objective function over beliefs that is approximately minimized by the iterative BP procedure. We give an overview of this argument below; for more details, see several introductory papers [150, 158].

The general idea behind a variational algorithm is:

(1) Define a family of tractable approximations $\mathcal{Q}$ and an objective function $\mathcal{O}(q)$ for $q \in \mathcal{Q}$. Each $q$ could be either a distribution whose marginals are easy to compute, like a tree-structured distribution, or it could simply be a set of approximate marginal distributions. If the latter strategy is used, then the approximate marginals are often called *pseudomarginals*, because they need not correspond to the marginals of any joint distribution over $\mathbf{y}$. The function $\mathcal{O}$ should be designed to measure how well a tractable $q \in \mathcal{Q}$ approximates $p$.
(2) Find the "closest" approximation $q^* = \min_{q \in \mathcal{Q}} \mathcal{O}(q)$.
(3) Use $q^*$ to approximate the marginals 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

$$\mathcal{O}(q) = \mathrm{KL}(q \| p) - \log Z \tag{4.33}$$

$$= -H(q) - \sum_a \sum_{\mathbf{y}_a} q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a), \tag{4.34}$$

Once we minimize this with respect to $q$ to obtain $q^*$, we can use the marginals of $q^*$ to approximate those of $p$. Indeed, the solution to this variational problem is $q^* = p$ with optimal value $\mathcal{O}(q^*) = -\log Z$. So solving this particular variational formulation is 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 maximizes $\mathcal{O}(q)$ as given by (4.34).

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 (4.34), which as it stands is difficult to compute. If $q$ were a tree-structured distribution, then its entropy could be written exactly as

$$H_{\text{Bethe}}(q) = -\sum_a \sum_{\mathbf{y}_a} q(\mathbf{y}_a) \log q(\mathbf{y}_a) + \sum_i \sum_{y_i} (d_i - 1) q(y_i) \log q(y_i), \tag{4.35}$$

where $d_i$ is the degree of $i$, that is, the number of factors that depend on $y_i$. This follows from substituting the tree-structured factorization (4.31) of the joint into the definition of entropy. If $q$ is not a tree, then we can still take $H_{\text{Bethe}}$ as an approximation to $H$ to compute the exact variational objective $\mathcal{O}$. This yields the *Bethe free energy*:

$$\mathcal{O}_{\text{Bethe}}(q) = -H_{\text{Bethe}}(q) - \sum_a \sum_{\mathbf{y}_a} q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a) \tag{4.36}$$

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* [150]. 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 minimized instead over a

relaxation of the marginal polytope. The relaxation is to require that the beliefs be only *locally consistent*, that is, that

$$\sum_{\mathbf{y}_a \setminus y_i} q_a(\mathbf{y}_a) = q_i(y_i) \quad \forall a, i \in a. \tag{4.37}$$

As a technical point, if a set of putative marginal distributions satisfies (4.37), this does not imply that they are globally consistent, i.e., that there exists a single joint $q(\mathbf{y})$ that has those marginals. For this reason, the distributions $q_a(\mathbf{y}_a)$ are also called *pseudomarginals*.

Yedidia et al. [157] show that constrained stationary points of $\mathcal{O}_{\text{BETHE}}$ under the constraints (4.37) are 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 5.4.2.

## 4.3 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 (4.6) and backward updates (4.9) requires $O(M^2)$ time, that is, quadratic time in the number of labels $M$. 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 always 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 (2.26) that computing the factors $\Psi_c(\mathbf{x}_c, \mathbf{y}_c)$ requires computing a dot product between the parameter vector $\theta_p$ and and the vector of features $\mathbf{f}_c = \{f_{pk}(y_c, \mathbf{x}_c) \,|\, \forall p, \forall k\}$. Often, many elements of the vectors $\mathbf{f}_c$ 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 $\Psi_c$ can be greatly improved using a sparse vector representation. In a similar fashion, we can use sparsity to improve the time required to compute the likelihood gradient, as we discuss in Section 5.

A related trick, that will also speed up forward backward, is to tie the parameters for certain subsets of transitions [24]. 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, i.e., $\alpha_t$ and $m_{sa}$ are often too small to be represented within numerical precision (for example, in an HMM $\alpha_t$ decays toward 0 exponentially fast in $t$). There are two standard approaches to this common problem. One approach is to scale each of the vectors $\alpha_t$ and $\beta_t$ 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 (4.31). But in fact, there is actually a more efficient method described by Rabiner [111] 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 (4.6)

becomes

$$\log \alpha_t(j) = \bigoplus_{i \in S} \big( \log \Psi_t(j, i, x_t) + \log \alpha_{t-1}(i) \big), \tag{4.38}$$

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

$$a \oplus b = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b}), \tag{4.39}$$

which can be much more numerically stable 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 (4.18). So in CRFs, special functions cannot be avoided. In the worst case, there are $TM^2$ of these $\Psi_t$ 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 $\Psi_t$ values.

# 5

# Parameter Estimation

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

One way to train CRFs is 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 Section 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 4.1 as a subroutine. The inference algorithms are used to compute both the value of the likelihood and its gradient. 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.

We begin by describing maximum likelihood training, both in the linear chain case (Section 5.1.1) and in the case of general graphical structures (Section 5.1.2), including the case of latent variables. We also describe two general methods for speeding up parameter estimation that exploit iid structure in the data: stochastic gradient descent (Section 5.2) and multithreaded training (Section 5.3).

For CRFs with general structures, exact maximum likelihood training is intractable, so typically approximate procedures must be used. At a high level, there are two strategies to dealing with this problem. The first strategy is to approximate the likelihood with another function that is easy to compute, which we call a *surrogate likelihood*, and to optimize the surrogate function numerically. The second strategy is an *approximate marginal* strategy that "plugs in" an approximate inference algorithm to compute approximate marginal distributions wherever the maximum likelihood training algorithm demands exact marginal distributions. Some care must be used here, because there can be interesting complications in the interaction between approximate inference and learning. We discuss these issues in Section 5.4.

## 5.1 Maximum Likelihood

### 5.1.1 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. To simplify the notation, we have assumed that every training sequence $\mathbf{x}^{(i)}$ has the same length $T$. Usually, every sequence will have a different length — in other words, $T$ will depend on $i$. The discussion below can be extended to cover this situation in a straightforward fashion.

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:

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\theta). \tag{5.1}$$

To compute the maximum likelihood estimate, we maximize $\ell(\theta)$, that is, the estimate is $\hat{\theta}_{\text{ML}} = \sup_\theta \ell(\theta)$.

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 considering the joint log likelihood

$$\log p(\mathbf{y},\mathbf{x};\theta) = \log p(\mathbf{y}|\mathbf{x};\theta) + \log p(\mathbf{x};\theta'), \tag{5.2}$$

notice that the term $p(\mathbf{x};\theta')$ does not depend on the parameters $\theta$ of the conditional distribution. If we do not need to estimate $p(\mathbf{x})$, then when computing the maximum likelihood estimate of $\theta$, we can simply drop the second term from the maximization, which leaves (5.1).

After substituting in the CRF model (2.18) into the likelihood (5.1), we get the following expression:

$$\ell(\theta) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}), \tag{5.3}$$

It is often the case that we have a large number of parameters, e.g., several hundred thousand. To reduce 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 $\theta$ and on a *regularization parameter* $1/2\sigma^2$ that determines the strength of the penalty. Then the regularized log likelihood is

$$\ell(\theta) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\theta_k^2}{2\sigma^2}. \tag{5.4}$$

The parameter $\sigma^2$ 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 $\theta$, if $\theta$ 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 $\sigma^2$ (e.g., up to a factor of 10). The best value of $\sigma^2$ depends on the size of the training set; for training sets such as those described in Section 5.5, we have often used a value like $\sigma^2 = 10$.

An alternative choice of regularization is to use the $L_1$ norm instead of the Euclidean norm, which corresponds to an double exponential prior on parameters [44]. This results in the following penalized likelihood:

$$\begin{aligned}\ell'(\theta) = &\sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K}\theta_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) \\ &- \sum_{i=1}^{N}\log Z(\mathbf{x}^{(i)}) - \alpha\sum_{k=1}^{K}|\theta_k|, \end{aligned} \tag{5.5}$$

where $\alpha$ is a regularization parameter that needs to be tuned, analogous to $\sigma^2$ for the $L_2$ regularizer. This regularizer tends to encourage sparsity in the learned parameters, meaning that most of the $\theta_k$ are 0. This can be useful for performing feature selection, and also has theoretical advantages [97]. 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 [65]. A disadvantage of the $L_1$ regularizer is that it is not differentiable at 0, which somewhat complicates numerical optimization [3, 44, 160].

In general, the function $\ell(\theta)$ cannot be maximized in closed form, so numerical optimization is used. The partial derivatives of (5.4) are

$$\begin{aligned}\frac{\partial \ell}{\partial \theta_k} = &\sum_{i=1}^{N}\sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) \\ &- \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y' | \mathbf{x}^{(i)}) - \frac{\theta_k}{\sigma^2}. \end{aligned} \tag{5.6}$$

The first term can be interpreted as the expected value of $f_k$ under the *empirical distribution* $\tilde{p}$, which is defined as

$$\tilde{p}(\mathbf{y}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}} \mathbf{1}_{\{\mathbf{x}=\mathbf{x}^{(i)}\}}. \tag{5.7}$$

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 is where we require the inference techniques that we introduced in Section 4. First, in the likelihood, inference is needed to compute the partition function $Z(\mathbf{x}^{(i)})$, which is a sum over all possible labellings. Second, 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 a difference from generative models, such as the undirected generative models of Section 2.1.1. Undirected generative models can also be trained by maximum likelihood, but for those models $Z$ depends only on the parameters, not on both the parameters and the input. The requirement to run inference $N$ times for each likelihood computation is the motivation behind stochastic gradient ascent methods (Section 5.2).

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. In addition, if a strictly concave regularizer is used, such as the $L_2$ regularizer, then the objective function becomes strictly concave, which implies that it has exactly one global optimum.

Perhaps the simplest approach to optimize $\ell$ is steepest ascent along the gradient (5.6), 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, techniques for optimizing (5.4) make approximate use of second-order information. Particularly successful have been quasi-Newton methods such as BFGS [8], 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. [17]. 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 [100]. 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. [63], as shown experimentally by several authors [80, 92, 125, 153]. Finally, trust region methods have recently been shown to perform well on multinomial logistic regression [74], and may work well for CRFs as well.

All of these optimization algorithms — steepest descent, Newton's method, quasi-Newton methods, conjugate gradient, and trust region methods — are standard techniques for numerically optimizing nonlinear functions. We apply them "off the shelf" to optimize the regularized likelihood of a CRF. Typically these algorithms require the ability to calculate both the value and the gradient of the function that they optimize. In our case the value is the likelihood (5.4) and the first-order derivatives are given by (5.6). This is why in Section 4 we discussed how to compute the partition function $Z(\mathbf{x})$ in addition to the marginals.

Finally, we discuss the computational cost of training linear chain models. As we will see in Section 4.1, 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 5.5 for examples.

### 5.1.2 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

$$\ell(\theta) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \theta_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x}). \tag{5.8}$$

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 $\theta_{pk}$ associated with a clique template $C_p$ is

$$\frac{\partial \ell}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) p(\mathbf{y}'_c | \mathbf{x}). \tag{5.9}$$

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. *Latent variables* are variables that are observed at neither training nor test time. CRFs with latent variables have been called *hidden-state CRFs (HCRFs)* in Quattoni et al. [109, 110], which was one of the first examples of latent variable CRFs. For other early applications of HCRFs, see [84, 138]. It is more difficult to train CRFs with latent variables because the latent variables need to be marginalized out to compute the likelihood.

Suppose we have a CRF 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:

$$p(\mathbf{y}, \mathbf{w}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{5.10}$$

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

$$\ell(\theta) = \log p(\mathbf{y}|\mathbf{x}) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x}). \tag{5.11}$$

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 (5.10), and clamping the variables $Y$ to their observed values in the training data, yielding a distribution over $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}) = \frac{1}{Z(\mathbf{y}, \mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p), \tag{5.12}$$

where the normalization factor is

$$Z(\mathbf{y},\mathbf{x}) = \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{5.13}$$

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

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p) = \frac{Z(\mathbf{y},\mathbf{x})}{Z(\mathbf{x})}. \tag{5.14}$$

Now that we have a way to compute $\ell$, we discuss how to maximize it with respect to $\theta$. Maximizing $\ell(\theta)$ can be difficult because $\ell$ 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 $\ell$: directly using the gradient, as in Quattoni et al. [109]; and using EM, as in McCallum et al. [84]. (In addition, it is also natural to use stochastic gradient descent here; see Section 5.2.) To maximize $\ell$ 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

$$\frac{df}{d\theta} = f(\theta) \frac{d \log f}{d\theta}, \tag{5.15}$$

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

$$\frac{\partial \ell}{\partial \theta_{pk}} = \frac{1}{\sum_{\mathbf{w}} p(\mathbf{y},\mathbf{w}|\mathbf{x})} \sum_{\mathbf{w}} \frac{\partial}{\partial \theta_{pk}} [p(\mathbf{y},\mathbf{w}|\mathbf{x})] \tag{5.16}$$

$$= \sum_{\mathbf{w}} p(\mathbf{w}|\mathbf{y},\mathbf{x}) \frac{\partial}{\partial \theta_{pk}} [\log p(\mathbf{y},\mathbf{w}|\mathbf{x})]. \tag{5.17}$$

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

$$\frac{\partial \ell}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c} p(\mathbf{w}'_c|\mathbf{y},\mathbf{x}) f_k(\mathbf{y}_c, \mathbf{x}_c, \mathbf{w}'_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c, \mathbf{y}'_c} p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c) f_k(\mathbf{y}'_c, \mathbf{x}_c, \mathbf{w}'_c). \tag{5.18}$$

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 (5.12). 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, $\ell$ 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, $\ell$ 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

$$\theta^{(j+1)} = \arg\max_{\theta'} \sum_{\mathbf{w}'} q(\mathbf{w}') \log p(\mathbf{y}, \mathbf{w}'|\mathbf{x};\theta'). \tag{5.19}$$

The direct maximization algorithm and the EM algorithm are strikingly similar. This can be seen by substituting the definition of $q$ into (5.19) and taking derivatives. The gradient is almost identical to the direct gradient (5.18). 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.

## 5.2 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

$$\frac{\partial \ell_i}{\partial \theta_k} = \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y' | \mathbf{x}^{(i)}) - \frac{\theta_k}{N\sigma^2}. \tag{5.20}$$

This is exactly the same as the full gradient (5.6), 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

$$\theta^{(m)} = \theta^{(m-1)} + \alpha_m \nabla \ell_i(\theta^{(m-1)}), \tag{5.21}$$

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 $\alpha_m$ 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 $\alpha_m$ to decrease as $m$ increases, so that the optimization algorithm converges to a single answer. Classic convergence results for stochastic approximation procedures [54, 115] provide the minimal requirements of $\sum_m \alpha_m = \infty$ and $\sum_m \alpha_m^2 < \infty$, that is, the $\alpha$ should go to 0 for large $m$ but not too quickly. The most common way to meet these requirements is to select a step size schedule of a form like $\alpha_m \sim 1/m$ or $\alpha_m \sim 1/\sqrt{m}$. 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

$$\alpha_m = \frac{1}{\sigma^2(m_0 + m)}, \tag{5.22}$$

where $m_0$ is a free parameter that needs to be set. A suggestion for setting this parameter, which is used in the `crfsgd` package of Leon Bottou [13], is to sample a small subset of the training data and run one pass of SGD over the subset with various fixed step sizes $\alpha$. Pick the $\alpha^*$ 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 [66]. Recently, there has been renewed interest in advanced online optimization methods [27, 43, 126, 149], which also update parameters in an online fashion, but in a more sophisticated way than simple SGD. Vishwanathan et al. [149] was the first application of stochastic gradient methods to CRFs.

The main disadvantage of stochastic gradient methods is that they 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.

## 5.3 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 (5.6) 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. [22].

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 [64].

## 5.4 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.

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 Section 4, 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 strategies for approximate training in CRFs [139]: a *surrogate likelihood* strategy in which we modify the objective function that is used for training, and an *approximate marginals* strategy in which we approximate the gradient. The first strategy involves finding a substitute for $\ell(\theta)$ (such as the BP approximation (5.27)), which we will call a *surrogate likelihood* that is easier to compute but is still expected to favor good parameter settings. Then the surrogate likelihood can be optimized using a gradient-based method, in a similar way to the exact likelihood. Second, an approximate marginals strategy 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 (5.9), and performing a gradient descent procedure using the resulting approximate gradients.

Although surrogate likelihood and approximate marginal methods are obviously closely related, they are distinct. Usually a 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* [151]. 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 [131, 139].

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 [60] 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 [151] points 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 5.4.1) and belief propagation (Section 5.4.2) and approximate gradient methods based on belief propagation (Section 5.4.2) and MCMC (Section 5.4.3).

### 5.4.1 Pseudolikelihood

One of the earliest surrogate likelihoods is the pseudolikelihood [9]. 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

$$\ell_{\text{PL}}(\theta) = \sum_{s \in V} \log p(y_s | \mathbf{y}_{N(s)}, \mathbf{x}; \theta). \tag{5.23}$$

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 $Y_s$. (As in (5.8), 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 $p(y_s|\mathbf{y}_{N(s)}, \mathbf{x}; \theta)$ of the model distribution to the training data, and because of the conditional independence assumptions of the model, the local conditional

distributions are sufficient to specify a joint distribution. (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.

There is no intention that the value of pseudolikelihood function $\ell_{\text{PL}}$ be a close approximation to the value of the true likelihood. Rather, the idea is for the maximum pseudolikelihood estimate $\hat{\theta}_{\text{PL}}$ to match the maximum likelihood estimate $\hat{\theta}_{\text{ML}}$. That is, the two functions are not intended to coincide but the two maxima are. Under certain conditions, most notably that the model family is correct, it can be shown that pseudolikelihood is asymptotically correct, i.e., that it will recover the true parameters in the limit of an infinite amount of data.

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 [146], more commonly the performance of pseudolikelihood is poor [134], in an intuitively analogous way that a Gibbs sampler can mix slowly in sequential models. In vision problems, a common criticism of pseudolikelihood is that it "places too much emphasis on the edge potentials" [149]. An intuitive explanation for this is that a model trained by pseudolikelihood can learn to rely on the true values of the neighboring variables, but these are not available at test time.

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:

$$\ell_{\text{EPL}}(\theta) = \sum_{t=1}^{T-1} \log p(y_t, y_{t+1} | y_{t-1}, y_{t+2}, \theta). \tag{5.24}$$

(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 [34, 75]. In composite likelihood, each of the individual terms in the likelihood predicts not just a single variable or a pair of variable, but a block of variables of arbitrary size that the user can select. Composite likelihood generalizes both standard pseudolikelihood and the "blockwise" pseudolikelihood. There are general theoretical results concerning asymptotic consistency and normality of composite likelihood estimators. Typically larger blocks lead to better parameter estimates, both in theory and in practice. This allows a tradeoff between training time and the quality of the resulting parameters.

Finally, the piecewise training method of Sutton and McCallum [134, 137] is related to composite likelihood, but is perhaps better understood as a belief propagation method, so we defer it to the next section.

### 5.4.2 Belief Propagation

The loopy belief propagation algorithm (Section 4.2.2) 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 approximate the true gradient (5.9) by substituting in the BP marginals. This results in approximate partial derivatives

$$\frac{\partial \tilde{\ell}}{\partial \theta_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) q(\mathbf{y}'_c). \tag{5.25}$$

These can be used to update the current parameter setting as

$$\theta_{pk}^{(t+1)} = \theta_{pk}^{(t)} + \alpha \frac{\partial \tilde{\ell}}{\partial \theta_{pk}}, \tag{5.26}$$

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 (5.8) which has the property that the gradient of the surrogate likelihood are exactly the approximate BP gradients (5.26). This may seem like a tall order, but fortunately it is possible using the Bethe free energy described in Section 4.2.2.

Remember from that section that loopy belief propagation can be viewed as an optimization algorithm, namely, one that minimizes the objective function $\mathcal{O}_{\text{Bethe}}(q)$ (4.36) 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 (5.8) gives us, for a fixed belief vector $q$, the approximate likelihood

$$\begin{aligned}\ell_{\text{Bethe}}(\theta, q) = &\sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \log \Psi_c(\mathbf{x}_c, \mathbf{y}_c) \\ &- \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} q(\mathbf{y}_c) \log \frac{q(\mathbf{y}_c)}{\Psi_c(\mathbf{x}_c, \mathbf{y}_c)} \\ &+ \sum_{s \in Y} (1 - d_i) q(y_s) \log q(y_s). \end{aligned} \tag{5.27}$$

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 $\ell_{\text{Bethe}}$ with respect to $q$ for fixed $\theta$ and take a gradient step to partially maximize $\ell_{\text{Bethe}}$ with respect to $\theta$ 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 $\theta$), the partial derivatives of (5.27) with respect to a weight $\theta_k$ is exactly (5.26), as desired.

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

$$\hat{\ell}(\theta; q) = \log \left[ \frac{\prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} q(\mathbf{y}_c)}{\prod_{s \in Y} q(y_s)^{d_s - 1}} \right], \tag{5.28}$$

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 generalization of the true likelihood for tree-structured models, as can be seen by comparing (5.28) with (4.31). This surrogate likelihood can be justified using a dual version of Bethe energy that we have presented here [91, 94]. 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 [137], 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 [139] discuss the close relationship between piecewise training and early stopping of belief propagation.

### 5.4.3 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) inference methods (Section 4.2.1) can be used within CRF training in an approximate marginals framework. Once we have chosen a Markov chain whose stationary distribution is $p(\mathbf{y}|\mathbf{x}; \theta)$, the algorithm is to the chain for a number of iterations and use the resulting approximate marginals $\hat{p}(\mathbf{y}|\mathbf{x}; \theta)$ to approximate the true marginals in the gradient (5.9).

In practice, however, MCMC methods have been less commonly used in the context of CRFs, because 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.

One possibility to overcome this difficulty is contrastive divergence (CD) [50], in which the true marginals $p(y_c|\mathbf{x})$ in (5.9) 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. While in principle CD can be applied to CRFs, we are unaware of much work that does this.

Another possibility is a more recent method called SampleRank [155], 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 performs parameter updates on individual MCMC steps rather than waiting for the chain to converge. Experiments have shown that models trained using SampleRank can have substantially better accuracy than CD [155].

In contrast to the approximate marginals framework that we have been discussing, it is very difficult to use an MCMC inference method within a surrogate likelihood framework, because it is notoriously difficult to obtain a good approximation to $\log Z(\mathbf{x})$ given samples from an MCMC method.

## 5.5 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, including typical values for the number of features and typical training times.

We describe three example tasks to which CRFs have been applied. The first example task is noun-phrase (NP) chunking [120], 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) [121], the task of identifying proper names in text, such as person names and company names. The final task is part-of-speech

Table 5.1. Scale of typical CRF applications in natural language processing.

| Task | Parameters | Observation Functions | # 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 |

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 [82], 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. The feature set has a similar flavor to that described in the named entity example in Section 2.6. We do not claim that the feature sets that we have used are optimal for these tasks, but still they should be useful for understanding typical scale.

For each of these data sets, Table 5.1 shows the number of parameters in the trained CRF model, the size of the training set, in terms of the total number of sequences and number of words, the number of possible labels for each sequence position, and 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 5.1, 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.

# 6

# 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.

## 6.1 Related Work

### 6.1.1 Structured Prediction

Classification methods provide established, powerful methods for predicting discrete outcomes. But in the applications that we have been considering in this survey, we wish to predict more complex objects, such as parse trees of natural language sentences [38, 144], alignments between sentences in different languages [145], and route plans in mobile robotics [112]. Each of these complex objects have internal structure, such as the tree structure of a parse, and we should be able to use this structure in order to represent our predictor more efficiently.

This general problem is called *structured prediction*. Just as the CRF likelihood generalizes logistic regression to predict arbitrary structures, the field of structured prediction generalizes the classification problem to the problem of predicting structured objects.

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. CRFs provide one way to do this, generalizing logistic regression, but other standard classification methods can be generalized to the structured prediction case as well. Detailed information about structured prediction methods is available in a recent collection of research papers [5]. In this section, we give an outline and pointers to some of these methods.

In order to explain more formally what a "structure" is, we review the general setup of classification. Many classification techniques can be interpreted as learning a discriminant function $F(y, \mathbf{x})$ over outputs in a finite set $y \in \mathcal{Y}$, for example, $\mathcal{Y} = \{1, 2, \dots C\}$. Given a test input $\mathbf{x}$, we predict the output and predict $y^* = \arg\max_y F(y, \mathbf{x})$. For example, often the discriminant function is linear in some basis expansion $\phi(\mathbf{x})$, e.g., $F(y, \mathbf{x}) = \theta_y^T \phi(\mathbf{x})$ for a set of weight vectors $\theta_y$.

Structured prediction follows this framework with one essential difference. In structured prediction, the set of possible outputs $\mathcal{Y}$ is extremely large, for example, the set of all ways to label the words in a sentence with named-entity labels, as in Section 2.6.1. Clearly in this situation it is not feasible to have a separate weight vector $\theta_y$ for each $y \in \mathcal{Y}$, i.e., for each of the possible ways to label a sequence. So in structured prediction methods we add the restriction that the discriminant function decomposes according to the output structure. Formally, in the structured case we write the output as a random vector $\mathbf{y} = (y_1, y_2, \dots y_T)$, and we require the discriminant to decompose according to a set of *parts* $Y_a$, each of which is simply a subset of the variables in $\mathbf{y}$. We index the parts by $a \in \{1, 2, \dots A\}$. Then the main assumption is that the discriminant decomposes as

$$F(\mathbf{y}, \mathbf{x}) = \sum_{a=1}^{A} F_a(\mathbf{y}_a, \mathbf{x}). \tag{6.1}$$

This discussion should be familiar from our discussion of undirected models general CRFs. For example, the log probability of a general CRF can written in this form, because for the purposes of prediction at test time we can ignore $Z(\mathbf{x})$. Structured prediction methods all share the property that the discriminant function decomposes according to a set of parts. They differ in how the parameters of the discriminant function are estimated from data.

Many types of structured prediction algorithms have been proposed. The CRF likelihood depend on summation over possible outputs, to obtain partition function $Z(\mathbf{x})$ and to obtain marginal distributions $p(y_i|\mathbf{x})$. Most other structured prediction methods that do not use the likelihood focus on *maximization* rather than summation. For example, *maximum-margin methods* that are so successful for univariate classification have been generalized to the structured case, yielding the structured SVM [2, 147] and the maximum margin Markov network [143]. The perceptron update can also be generalized to structured models [25]. 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 [28].

Another class of structured methods are search-based methods [31, 32] 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 (i.e., ones that do not decompose in the same way that the discriminant function does). Finally, LeCun et al. [68] generalizes many prediction methods, including the ones listed above, under the rubric of *energy-based* methods.

A general advantage of maximization- and search-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 example, see Taskar et al. [145]. For more complex problems, neither summation nor maximization is tractable,

so this advantage is perhaps not as significant, although even in this case maximization makes it easier to apply approximate search methods such as beam search and pruning. For example, see Pal et al. [102] for an example of the difficulties that can arise when trying to naively incorporate beam search within CRF training.

It is worth noting that although maximization-based training methods do not use the likelihood during parameter estimation, the resulting parameters could still be interpreted as defining a factorized conditional probability distribution $p(\mathbf{y}|\mathbf{x})$, and hence a CRF by Definition 2.3.

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 [142]. For examples of structured models with latent variables, see Quattoni et al. [109] and McCallum et al. [84]. A particularly powerful example of this is provided by Bayesian methods, in which the model parameters themselves are integrated out (Section 6.2.1). That having been said, recent work has proposed methods of incorporating latent variables into SVMs and structured SVMs [36, 159].

The differences in prediction accuracy between the various structured prediction methods are not well understood. To date, there has been little careful experimental comparison of structured prediction methods across different domains, although see Keerthi and Sundararajan [53] for some experiments in this regard.[1] 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.

### 6.1.2 Neural Networks

There are close relationships between neural networks and CRFs, 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

[1] An earlier study [99] appears to have been flawed. See Keerthi and Sundararajan [53] for discussion.

example, by using a shared latent representation [18], or by modeling dependencies between outputs directly [67]. Although neural networks are typically trained using stochastic gradient descent (Section 5.2), 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 inaccurate. A neural network without a hidden layer is a linear classifier that can be trained efficiently in a number of ways, while a CRF with latent variables has a 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 layer, 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 states will still yield additional benefit. Once hidden states are 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.

There is another sense in which even without hidden states, a CRF is like a neural network. For concreteness consider a linear-chain CRF $p(y_1, y_2|\mathbf{x})$ over a sequence of length 2. The CRF is a linear model, by which we mean that for any two label assignments $\mathbf{y} = (y_1, y_2)$ and $\mathbf{y}' = (y_1', y_2')$, the log odds $\log(p(\mathbf{y}|\mathbf{x})/p(\mathbf{y}'|\mathbf{x}))$ are a linear function of the parameters. However, the marginal distributions behave nonlinearly. In other words, $\log(p(y_1|\mathbf{x})/p(y_1'|\mathbf{x}))$ is *not* a linear function of the parameters. This is because the variable $y_2$ acts like a hidden variable when computing the marginal distribution over $y_1$. This viewpoint

has been exploited to approximate CRFs using per-position classifiers with an expanded feature set [71].

### 6.1.3 MEMMs, Directed Models, and Label Bias

Linear-chain CRFs were originally introduced as an improvement to the *maximum-entropy Markov model* (MEMM) [85], which is essentially a Markov model in which the transition probabilities are given by logistic regression. Formally, an MEMM is

$$p_{\text{MEMM}}(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1},\mathbf{x}) \tag{6.2}$$

$$p(y_t|y_{t-1},\mathbf{x}) = \frac{1}{Z_t(y_{t-1},\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\} \tag{6.3}$$

$$Z_t(y_{t-1},\mathbf{x}) = \sum_{y'} \exp\left\{\sum_{k=1}^{K} \theta_k f_k(y', y_{t-1}, \mathbf{x}_t)\right\}. \tag{6.4}$$

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 local conditional distribution is a logistic regression model with input $\mathbf{x}$ [117].

In the linear-chain case, notice that the MEMM works out to have the same form as the linear-chain CRF (5.3) 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 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 problem of label bias [63]. The label bias problem is essentially that future observations cannot affect the posterior distribution over earlier states. To understand the label bias problem, consider the backward recursion (4.9). In the case of an MEMM, this

amounts to

$$\beta_t(i) = \sum_{j \in S} p(y_{t+1} = j | y_t = i, x_{t+1}) \beta_{t+1}(j). \tag{6.5}$$

Unfortunately, this sum is always 1, regardless of the value of the current label $i$. What this means is that the future observations provide no information about the current state, which seems to lose a major advantage of sequence modeling. 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 (6.5) collapses, and $\beta_t(i) = 1$.

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 6.1. 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 modeling.) 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. This has the effect of adding arcs between (for example) $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. This explains why the performance gap in practice between MEMMs and CRFs is not always as large as might be expected.

This graphical modeling view on label bias highlights an important point. Label bias is not caused by a model being directed or

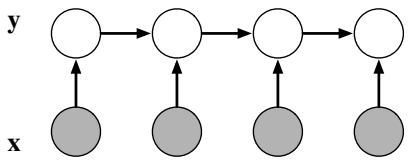


Fig. 6.1 Graphical model of a maximum entropy Markov model [85].

undirected. It is caused by the structure of the particular directed model that is used in the MEMM. This point is forcefully corroborated by Berg-Kirkpatrick et al. [6], which presents impressive experimental results on various unsupervised learning tasks by using directed models whose local conditional distributions have a log linear structure like the MEMM does, but which avoid label bias because they have a generative graphical structure, rather than the v-structures of Figure 6.1.

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 a long history in the speech community, where it is called maximum mutual information training [4]. Naively, this might seem to offer a simpler training algorithm than CRFs do, because directed models are usually easier to train than undirected models. But in fact, this approach does not obviate the need to perform inference during training. 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 can be 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.

## 6.2 Frontier Areas

Finally, we describe a few open research areas that are 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.

### 6.2.1 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 5.1.1. 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 $\mathbf{y}^* = \max_{\mathbf{y}} \int p(\mathbf{y}|\mathbf{x};\theta)p(\theta|\mathbf{x}^{(1)},\mathbf{y}^{(1)},\ldots,\mathbf{x}^{(N)},\mathbf{y}^{(N)})d\theta$. This integral over $\theta$ needs to be approximated, for example, by MCMC.

In general, it is difficult to formulate efficient Bayesian methods for undirected models; see [95, 96] for some of the few examples in this regard. A few papers have specially considered approximate inference algorithms for Bayesian CRFs [107, 154, 161], 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 5.1). 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 [141]. However, the Bayesian perspective does have other potential benefits, particularly when more complex, hierarchical priors are considered.

### 6.2.2 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 [46, 52], generalized

expectation criteria [81], discriminative methods [70], posterior regularization [39, 45], and measurement-based learning [73].

### 6.2.3 Structure Learning in CRFs

All of the methods described in this survey 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 [15] point out an interesting complication that is specific to conditional models. For a generative model $p(\mathbf{x})$, maximum likelihood structure learning can be performed efficiently if the model is restricted to be tree-structured, using the well-known Chow-Liu algorithm. In the conditional case, when we wish to estimate the structure of $p(\mathbf{y}|\mathbf{x})$, the analogous algorithm is more difficult, because it requires estimating marginal distributions of the form $p(y_u, y_v|\mathbf{x})$, that is, we need to estimate the effects of the entire input vector on every pair of output variables. It is difficult to estimate these distributions without knowing the structure of the model to begin with.

# Acknowledgments

We thank Kedar Bellare, Francine Chen, Gregory Druck, Benson Limketkai, David Mimno, Ray Mooney, Prakash Nadkarni, Oscar Täckström, Wenhao Wang, Vittorio Ferrari, and three anonymous reviewers for useful comments on earlier versions of this tutorial. A previous version of this survey has appeared in Sutton and McCallum [136], and as part of Charles Sutton's doctoral dissertation [132].

# References


[1] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[2] Y. Altun, I. Tsochantaridis, and T. Hofmann, "Hidden Markov support vector machines," in *International Conference on Machine Learning (ICML)*, 2003.

[3] G. Andrew and J. Gao, "Scalable training of $l_1$-regularized log-linear models," in *International Conference on Machine Learning (ICML)*, 2007.

[4] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "Maximum mutual information estimation of hidden Markov model parameters for speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 11, pp. 49–52, 1986.

[5] G. H. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan, eds., *Predicting Structured Data*. MIT Press, 2007.

[6] T. Berg-Kirkpatrick, A. Bouchard-Côté, J. DeNero, and D. Klein, "Painless unsupervised learning with features," in *Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, pp. 582–590.

[7] A. Bernal, K. Crammer, A. Hatzigeorgiou, and F. Pereira, "Global discriminative learning for higher-accuracy computational gene prediction," *PLoS Computational Biology*, vol. 3, no. 3, 2007.

[8] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2nd ed., 1999.

[9] J. Besag, "Statistical analysis of non-lattice data," *The Statistician*, vol. 24, no. 3, pp. 179–195, 1975.

[10] A. Blake, P. Kohli, and C. Rother, eds., *Markov Random Fields for Vision and Image Processing*. MIT Press, 2011.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, p. 993, 2003.
[12] P. Blunsom and T. Cohn, "Discriminative word alignment with conditional random fields," in *International Conference on Computational Linguistics and Annual Meeting of the Association for Computational Linguistics (COLING-ACL)*, pp. 65–72, 2006.
[13] L. Bottou, "Stochastic gradient descent examples on toy problems," 2010.
[14] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in nd images," in *International Conference on Computer Vision (ICCV)*, vol. 1, pp. 105–112, 2001.
[15] J. K. Bradley and C. Guestrin, "Learning tree conditional random fields," in *International Conference on Machine Learning (ICML)*, 2010.
[16] R. Bunescu and R. J. Mooney, "Collective information extraction with relational Markov networks," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.
[17] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 2, pp. 129–156, 1994.
[18] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.
[19] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms using different performance metrics," Technical Report TR2005-1973, Cornell University, 2005.
[20] H. L. Chieu and H. T. Ng, "Named entity recognition with a maximum entropy approach," in *Conference on Natural Language Learning (CoNLL)*, pp. 160–163, 2003.
[21] Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan, "Identifying sources of opinions with conditional random fields and extraction patterns," in *Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.
[22] C.-T. Chu, S. K. Kim, Y.-A. Lin, 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*, pp. 281–288, MIT Press, 2007.
[23] S. Clark and J. R. Curran, "Parsing the WSJ using CCG and log-linear models," in *Proceedings of the Meeting of the Association for Computational Linguistics (ACL)*, pp. 103–110, 2004.
[24] T. Cohn, "Efficient inference in large conditional random fields," in *European Conference on Machine Learning (ECML)*, pp. 606–613, Berlin, Germany, September 2006.
[25] M. 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.
[26] P. J. Cowans and M. Szummer, "A graphical model for simultaneous partitioning and labeling," in *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.
[27] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, 2006.

[28] K. Crammer and Y. Singer, "Ultraconservative online algorithms for multiclass problems," *Journal of Machine Learning Research*, vol. 3, pp. 951–991, January 2003.

[29] A. Culotta, R. Bekkerman, and A. 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.

[30] A. Culotta and A. McCallum, "Confidence estimation for information extraction," in *Human Language Technology Conference (HLT)*, 2004.

[31] H. Daumé III, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine Learning Journal*, 2009.

[32] H. Daumé III and D. Marcu, "Learning as search optimization: Approximate large margin methods for structured prediction," in *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.

[33] T. Deselaers, B. Alexe, and V. Ferrari, "Localizing objects while learning their appearance," in *European Conference on Computer Vision (ECCV)*, 2010.

[34] J. V. Dillon and G. Lebanon, "Stochastic composite likelihood," *Journal of Machine Learning Research*, vol. 11, pp. 2597–2633, October 2010.

[35] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006.

[36] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[37] J. Finkel, T. Grenager, and C. D. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[38] J. R. Finkel, A. Kleeman, and C. D. Manning, "Efficient, feature-based, conditional random field parsing," in *Annual Meeting of the Association for Computational Linguistics (ACL/HLT)*, pp. 959–967, 2008.

[39] K. Ganchev, J. Graca, J. Gillenwater, and B. Taskar, "Posterior regularization for structured latent variable models," Technical Report MS-CIS-09-16, University of Pennsylvania Department of Computer and Information Science, 2009.

[40] A. E. Gelfand and A. F. M. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American Statistical Association*, vol. 85, pp. 398–409, 1990.

[41] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.

[42] N. Ghamrawi and A. McCallum, "Collective multi-label classification," in *Conference on Information and Knowledge Management (CIKM)*, 2005.

[43] A. Globerson, T. Koo, X. Carreras, and M. Collins, "Exponentiated gradient algorithms for log-linear structured prediction," in *International Conference on Machine Learning (ICML)*, 2007.

[44] J. Goodman, "Exponential priors for maximum entropy models," in *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2004.

[45] J. Graca, K. Ganchev, B. Taskar, and F. Pereira, "Posterior vs parameter sparsity in latent variable models," in *Advances in Neural Information Processing Systems 22*, (Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 664–672, 2009.

[46] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[47] M. L. Gregory and Y. Altun, "Using conditional random fields to predict pitch accents in conversational speech," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 677–683, 2004.

[48] A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt, "Hidden conditional random fields for phone classification," in *International Conference on Speech Communication and Technology*, 2005.

[49] X. He, R. S. Zemel, and M. A. Carreira-Perpiñián, "Multiscale conditional random fields for image labelling," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[50] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, 2002.

[51] L. Hirschman, A. Yeh, C. Blaschke, and A. Valencia, "Overview of BioCreAtIvE: critical assessment of information extraction for biology," *BMC Bioinformatics*, vol. 6, no. Suppl 1, no. Suppl 1, 2005.

[52] F. Jiao, S. Wang, C.-H. 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.

[53] S. S. Keerthi and S. Sundararajan, "CRF versus SVM-struct for sequence labeling," Technical report, Yahoo! Research, 2007.

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

[55] J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier, "Introduction to the bio-entity recognition task at JNLPBA," in *International joint workshop on natural language processing in biomedicine and its applications*, pp. 70–75, Association for Computational Linguistics, 2004.

[56] P. Kohli, L. Ladickỳ, and P. H. S. Torr, "Robust higher order potentials for enforcing label consistency," *International Journal of Computer Vision*, vol. 82, no. 3, no. 3, pp. 302–324, 2009.

[57] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[58] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, no. 2, pp. 498–519, 2001.

[59] T. Kudo, K. Yamamoto, and Y. Matsumoto, "Applying conditional random fields to Japanese morphological analysis," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[60] A. Kulesza and F. Pereira, "Structured learning with approximate inference," in *Advances in Neural Information Processing Systems*, 2008.

[61] S. Kumar and M. Hebert, "Discriminative fields for modeling spatial dependencies in natural images," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[62] S. Kumar and M. Hebert, "Discriminative random fields," *International Journal of Computer Vision*, vol. 68, no. 2, no. 2, pp. 179–201, 2006.

[63] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *International Conference on Machine Learning (ICML)*, 2001.

[64] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," in *Advances in Neural Information Processing Systems 22*, (Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 2331–2339, 2009.

[65] T. Lavergne, O. Cappé, and F. Yvon, "Practical very large scale CRFs," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 504–513, 2010.

[66] Y. Le Cun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524, Springer Verlag, 1998.

[67] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[68] Y. LeCun, S. Chopra, R. Hadsell, R. Marc'Aurelio, and F.-J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, (G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, eds.), MIT Press, 2007.

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

[70] W. Li and A. McCallum, "A note on semi-supervised learning using Markov random fields," 2004.

[71] P. Liang, H. Daumé III, and D. Klein, "Structure compilation: Trading structure for features," in *International Conference on Machine Learning (ICML)*, pp. 592–599, 2008.

[72] P. Liang and M. I. Jordan, "An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators," in *International Conference on Machine Learning (ICML)*, pp. 584–591, 2008.

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

[74] C.-J. Lin, R. C.-H. Weng, and S. Keerthi, "Trust region newton methods for large-scale logistic regression," in *Interational Conference on Machine Learning (ICML)*, 2007.

[75] B. G. Lindsay, "Composite likelihood methods," *Contemporary Mathematics*, pp. 221–239, 1988.

[76] Y. Liu, J. Carbonell, P. Weigele, and V. Gopalakrishnan, "Protein fold recognition using segmentation conditional random fields (SCRFs)," *Journal of Computational Biology*, vol. 13, no. 2, no. 2, pp. 394–406, 2006.

[77] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1150–1157, 1999.

[78] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter, "WinBUGS — a Bayesian modelling framework: Concepts, structure, and extensibility," *Statistics and Computing*, vol. 10, no. 4, no. 4, pp. 325–337, 2000.

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

[80] R. Malouf, "A comparison of algorithms for maximum entropy parameter estimation," in *Conference on Natural Language Learning (CoNLL)*, (D. Roth and A. van den Bosch, eds.), pp. 49–55, 2002.

[81] G. Mann and A. McCallum, "Generalized expectation criteria for semi-supervised learning of conditional random fields," in *Proceedings of Association of Computational Linguistics*, 2008.

[82] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, no. 2, pp. 313–330, 1993.

[83] A. McCallum, "Efficiently inducing features of conditional random fields," in *Conference on Uncertainty in AI (UAI)*, 2003.

[84] A. McCallum, K. Bellare, and F. Pereira, "A conditional random field for discriminatively-trained finite-state string edit distance," in *Conference on Uncertainty in AI (UAI)*, 2005.

[85] A. McCallum, D. Freitag, and F. Pereira, "Maximum entropy Markov models for information extraction and segmentation," in *International Conference on Machine Learning (ICML)*, pp. 591–598, San Francisco, CA, 2000.

[86] A. McCallum and W. 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.

[87] A. McCallum, K. Schultz, and S. Singh, "FACTORIE: Probabilistic programming via imperatively defined factor graphs," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[88] A. McCallum and B. Wellner, "Conditional models of identity uncertainty with application to noun coreference," in *Advances in Neural Information Processing Systems 17*, (L. K. Saul, Y. Weiss, and L. Bottou, eds.), pp. 905–912, Cambridge, MA: MIT Press, 2005.

[89] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, no. 2, pp. 140–152, 1998.

[90] S. Miller, J. Guinness, and A. Zamanian, "Name tagging with word clusters and discriminative training," in *HLT-NAACL 2004: Main Proceedings*, (D. Marcu, S. Dumais, and S. Roukos, eds.), pp. 337–342, Boston, Massachusetts, USA: Association for Computational Linguistics, May 2–May 7 2004.

[91] T. P. Minka, "The EP energy function and minimization schemes," Technical report, 2001.

[92] T. P. Minka, "A comparsion of numerical optimizers for logistic regression," Technical report, 2003.

[93] T. P. Minka, "Discriminative models, not discriminative training," Technical Report MSR-TR-2005-144, Microsoft Research, October 2005.

[94] T. P. Minka, "Divergence measures and message passing," Technical Report MSR-TR-2005-173, Microsoft Research, 2005.

[95] I. Murray, "Advances in Markov chain Monte Carlo methods," PhD thesis, Gatsby computational neuroscience unit, University College London, 2007.

[96] I. Murray, Z. Ghahramani, and D. J. C. MacKay, "MCMC for doubly-intractable distributions," in *Uncertainty in Artificial Intelligence (UAI)*, pp. 359–366, AUAI Press, 2006.

[97] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in *International Conference on Machine Learning (ICML)*, 2004.

[98] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Information Processing Systems 14*, (T. G. Dietterich, S. Becker, and Z. Ghahramani, eds.), pp. 841–848, Cambridge, MA: MIT Press, 2002.

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

[100] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.

[101] S. Nowozin and C. H. Lampert, "Structured prediction and learning in computer vision," *Foundations and Trends in Computer Graphics and Vision*, vol. 6, no. 3-4, no. 3-4, 2011.

[102] C. Pal, C. Sutton, and A. McCallum, "Sparse forward-backward using minimum divergence beams for fast training of conditional random fields," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

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

[104] F. Peng, F. Feng, and A. McCallum, "Chinese segmentation and new word detection using conditional random fields," in *International Conference on Computational Linguistics (COLING)*, pp. 562–568, 2004.

[105] F. Peng and A. McCallum, "Accurate information extraction from research papers using conditional random fields," in *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2004.

[106] D. Pinto, A. McCallum, X. Wei, and W. B. Croft, "Table extraction using conditional random fields," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[107] Y. Qi, M. Szummer, and T. P. Minka, "Bayesian conditional random fields," in *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[108] Y. Qi, M. Szummer, and T. P. Minka, "Diagram structure recognition by Bayesian conditional random fields," in *International Conference on Computer Vision and Pattern Recognition*, 2005.

[109] A. Quattoni, M. Collins, and T. Darrell, "Conditional random fields for object recognition," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1104, 2005.

[110] A. Quattoni, S. Wang, L.-P. Morency, M. Collins, and T. Darrell, "Hidden-state conditional random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[111] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, no. 2, pp. 257–286, 1989.

[112] N. Ratliff, J. A. Bagnell, and M. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning*, July 2006.

[113] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1–2, no. 1–2, pp. 107–136, 2006.

[114] S. Riezler, T. King, R. Kaplan, R. Crouch, J. T. Maxwell III, and M. Johnson, "Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2002.

[115] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[116] C. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer, 2004.

[117] D. Rosenberg, D. Klein, and B. Taskar, "Mixture-of-parents maximum entropy Markov models," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[118] D. Roth and W. Yih, "Integer linear programming inference for conditional random fields," in *International Conference on Machine Learning (ICML)*, pp. 737–744, 2005.

[119] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 23, no. 3, no. 3, pp. 309–314, 2004.

[120] E. F. T. K. Sang and S. 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.

[121] E. F. T. K. Sang and F. D. Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," in *Proceedings of CoNLL-2003*, (W. Daelemans and M. Osborne, eds.), pp. 142–147, Edmonton, Canada, 2003.

[122] S. Sarawagi and W. W. Cohen, "Semi-Markov conditional random fields for information extraction," in *Advances in Neural Information Processing Systems 17*, (L. K. Saul, Y. Weiss, and L. Bottou, eds.), pp. 1185–1192, Cambridge, MA: MIT Press, 2005.

[123] K. Sato and Y. Sakakibara, "RNA secondary structural alignment with conditional random fields," *Bioinformatics*, vol. 21, pp. ii237–242, 2005.

[124] B. Settles, "Abner: An open source tool for automatically tagging genes, proteins, and other entity names in text," *Bioinformatics*, vol. 21, no. 14, no. 14, pp. 3191–3192, 2005.

[125] F. Sha and F. Pereira, "Shallow parsing with conditional random fields," in *Conference on Human Language Technology and North American Association for Computational Linguistics (HLT-NAACL)*, pp. 213–220, 2003.

[126] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated subgradient solver for SVM," in *International Conference on Machine Learning (ICML)*, 2007.

[127] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost: Joint appearance, shape and context modeling for mulit-class object recognition and segmentation," in *European Conference on Computer Vision (ECCV)*, 2006.

[128] P. Singla and P. Domingos, "Discriminative training of Markov logic networks," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 868–873, Pittsburgh, PA, 2005.

[129] F. K. Soong and E.-F. Huang, "A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1991.

[130] D. H. Stern, T. Graepel, and D. J. C. MacKay, "Modelling uncertainty in the game of go," in *Advances in Neural Information Processing Systems 17*, (L. K. Saul, Y. Weiss, and L. Bottou, eds.), pp. 1353–1360, Cambridge, MA: MIT Press, 2005.

[131] I. Sutskever and T. Tieleman, "On the convergence properties of contrastive divergence," in *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[132] C. Sutton, "Efficient Training Methods for Conditional Random Fields," PhD thesis, University of Massachusetts, 2008.

[133] C. Sutton and A. 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.

[134] C. Sutton and A. McCallum, "Piecewise training of undirected models," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[135] C. Sutton and A. McCallum, "Improved dynamic schedules for belief propagation," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[136] C. Sutton and A. McCallum, "An introduction to conditional random fields for relational learning," in *Introduction to Statistical Relational Learning*, (L. Getoor and B. Taskar, eds.), MIT Press, 2007.

[137] C. Sutton and A. McCallum, "Piecewise training for structured prediction," *Machine Learning*, vol. 77, no. 2–3, no. 2–3, pp. 165–194, 2009.

[138] C. Sutton, A. McCallum, and K. Rohanimanesh, "Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data," *Journal of Machine Learning Research*, vol. 8, pp. 693–723, March 2007.

[139] C. Sutton and T. Minka, "Local training and belief propagation," Technical Report TR-2006-121, Microsoft Research, 2006.

[140] C. Sutton, K. Rohanimanesh, and A. McCallum, "Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data," in *International Conference on Machine Learning (ICML)*, 2004.

[141] C. Sutton, M. Sindelar, and A. McCallum, "Reducing weight undertraining in structured discriminative learning," in *Conference on Human Language Technology and North American Association for Computational Linguistics (HLT-NAACL)*, 2006.

[142] B. Taskar, P. Abbeel, and D. Koller, "Discriminative probabilistic models for relational data," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2002.

[143] B. Taskar, C. Guestrin, and D. Koller, "Max-margin Markov networks," in *Advances in Neural Information Processing Systems 16*, (S. Thrun, L. Saul, and B. Schölkopf, eds.), Cambridge, MA: MIT Press, 2004.

[144] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning, "Max-margin parsing," in *Empirical Methods in Natural Language Processing (EMNLP04)*, 2004.

[145] B. Taskar, S. Lacoste-Julien, and D. Klein, "A discriminative matching approach to word alignment," in *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pp. 73–80, 2005.

[146] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *HLT-NAACL*, 2003.

[147] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *International Conference on Machine Learning (ICML)*, ICML '04, 2004.

[148] P. Viola and M. Narasimhan, "Learning to extract information from semi-structured text using a discriminative context free grammar," in *Proceedings of the ACM SIGIR*, 2005.

[149] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. Murphy, "Accelerated training of conditional random fields with stochastic meta-descent," in *International Conference on Machine Learning (ICML)*, pp. 969–976, 2006.

[150] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1-2, no. 1-2, pp. 1–305, 2008.

[151] M. J. Wainwright, "Estimating the wrong Markov random field: Benefits in the computation-limited setting," in *Advances in Neural Information Processing Systems 18*, (Y. Weiss, B. Schölkopf, and J. Platt, eds.), Cambridge, MA: MIT Press, 2006.

[152] M. J. Wainwright, T. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Transactions on Information Theory*, vol. 45, no. 9, no. 9, pp. 1120–1146, 2003.

[153] H. Wallach, "Efficient training of conditional random fields," M.Sc. thesis, University of Edinburgh, 2002.

[154] M. Welling and S. Parise, "Bayesian random fields: The Bethe-Laplace approximation," in *Uncertainty in Artificial Intelligence (UAI)*, 2006.

[155] M. Wick, K. Rohanimanesh, A. Culotta, and A. McCallum, "SampleRank: Learning preferences from atomic gradients," in *Neural Information Processing Systems (NIPS) Workshop on Advances in Ranking*, 2009.

[156] M. Wick, K. Rohanimanesh, A. McCallum, and A. Doan, "A discriminative approach to ontology alignment," in *International Workshop on New Trends in Information Integration (NTII)*, 2008.

[157] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free energy approximations and generalized belief propagation algorithms," Technical Report TR2004-040, Mitsubishi Electric Research Laboratories, 2004.

[158] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282–2312, July 2005.

[159] C.-N. Yu and T. Joachims, "Learning structural svms with latent variables," in *International Conference on Machine Learning (ICML)*, 2009.

[160] J. Yu, S. V. N. Vishwanathan, S. Güunter, and N. N. Schraudolph, "A quasi-Newton approach to nonsmooth convex optimization problems in machine learning," *Journal of Machine Learning Research*, vol. 11, pp. 1145–1200, March 2010.

[161] Y. Zhang and C. Sutton, "Quasi-Newton Markov chain Monte Carlo," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.