EFFICIENT TRAINING METHODS FOR CONDITIONAL RANDOM FIELDS

A Dissertation Presented

by

CHARLES A. SUTTON

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2008

Computer Science

EFFICIENT TRAINING METHODS FOR CONDITIONAL RANDOM FIELDS

A Dissertation Presented

by

CHARLES A. SUTTON

In memoriam, Marian Janice Sutton (1948 – 2002)
From Alice, Beanna, and Chip

ACKNOWLEDGMENTS

It is customary for doctoral theses to begin with embarrassingly fulsome thanks to one's advisor, professors, family, and friends. This thesis will be no exception. I am therefore fortunate that I *have* had a wonderful network of mentors, family, and friends, so that I can express my thanks without risk of overstatement.

I could not have completed this thesis without the help of my advisor Andrew McCallum. Andrew is a dynamo of enthusiasm, encouragement, and interesting suggestions—as anyone who has had him drop by their cubicle can attest. I have been especially influenced by his keen sense of how to find research areas that are theoretically interesting, practically important, and ripe for the attack. I am also grateful to have benefited from the feedback of the other members of my dissertation committee: Sridhar Mahadevan, Erik Learned-Miller, Jonathan Machta, and Tommi Jaakkola.

I spent a delightful summer doing an internship at Microsoft Research Cambridge. I thank Tom Minka and Martin Szummer for hosting me, and for our many discussions that helped me to better understand approximate inference and training, and that helped many of the explanations in this thesis to become crisper and more insightful.

At UMass, I have also learned much from discussions with Wei Li, Aron Culotta, Greg Druck, Gideon Mann, and Jerod Weinman. I have also had the opportunity to collaborate on several projects outside of my thesis, and for this I thank my collaborators, which have included Rob Hall, Gideon Mann, Chris Pal, Khashayar Rohanimanesh, Hanna Wallach, and Max Welling.

Many friends outside of work have helped me maintain my emotional well-being. They are too many to list here—or at least I would like to think so—but I should at least offer thanks to Brent Heeringa for playing matchmaker; to Micah Feldman and Matt Noonan for helping me to improve at Go (although I wish they could have done just a little bit better . . .); and Jodi Schneider, who deserves an acknowledgments section of her own.

I have enjoyed the staggering privilege of twenty-three years of public education. The seven of those that I spent in graduate school would not have been possible without the support of many funding agencies. I am therefore grateful to acknowledge that this work was supported in part by the Center for Intelligent Information Retrieval; in part by the Defense Advanced Research Projects Agency (DARPA) under contract number HR0011-06-C-0023; in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grants #IIS-0326249 and #IIS-0427594; and in part by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under contract number NBCHD030010.

Finally, I thank my brother Robert for his support; my stepmother Mary for taking care of Dad; and my parents, Charles and Marian, for everything.

ABSTRACT

EFFICIENT TRAINING METHODS FOR CONDITIONAL RANDOM FIELDS

FEBRUARY 2008

CHARLES A. SUTTON

B.A., ST. MARY'S COLLEGE OF MARYLAND

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Many applications require predicting not a just a single variable, but multiple variables that depend on each other. Recent attention has therefore focused on *structured prediction* methods, which combine the modeling flexibility of graphical models with the ability to employ complex, dependent features typical of traditional classification methods. Especially popular have been conditional random fields (CRFs), which are graphical models of the conditional distribution over outputs given a set of observed features. Unfortunately, parameter estimation in CRFs requires repeated inference, which can be computationally expensive. Complex graphical structures are increasingly desired in practical applications, but then training time often becomes prohibitive.

In this thesis, I investigate efficient training methods for conditional random fields with complex graphical structure, focusing on *local methods* which avoid propagating information globally along the graph. First, I investigate *piecewise training*, which trains each of a model's factors separately. I present three views of piecewise training: as maximizing the likelihood in a so-called "node-split graph", as maximizing the Bethe likelihood with uniform messages, and as generalizing the pseudo-moment matching estimator of Wainwright, Jaakkola, and Willsky. Second, I propose *piecewise pseudolikelihood*, a hybrid procedure which "pseudolikelihood-izes" the piecewise likelihood, and is therefore more efficient if the variables have large cardinality. Piecewise pseudolikelihood performs well even on applications in which standard pseudolikelihood performs poorly. Finally, motivated by the connection between piecewise training and BP, I explore training methods using beliefs arising from stopping BP before convergence. I propose a new schedule for message propagation that improves upon the dynamic schedule proposed recently by Elidan, McGraw, and Koller, and present suggestive results applying dynamic schedules to the system of equations that combine inference and learning.

I also present two novel families of loopy CRFs, which appear as test cases throughout. First is the dynamic CRF, which combines the factorized state representation of dynamic Bayesian networks with the modeling flexibility of conditional models. The second of these is the skip-chain CRF, which models the fact that identical words are likely to have the same label, even if they occur far apart.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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 [61], estimating the score in a game of Go [116], segmenting genes in a strand of DNA [5], and extracting syntax from natural-language text [130]. In such applications, we wish to predict a vector $\mathbf{y} = \{y_0, y_1, \ldots, y_T\}$ of random variables given an observed feature vector $\mathbf{x}$. A relatively simple example from natural-language processing is part-of-speech tagging, in which each variable y_s is the part-of-speech tag of the word at position s, and the input $\mathbf{x}$ is divided into feature vectors $\{\mathbf{x}_0, \mathbf{x}_1 \ldots \mathbf{x}_T\}$. Each $\mathbf{x}_s$ contains various information about the word at position s, such as its identity, orthographic features such as prefixes and suffixes, membership in domain-specific lexicons, and information in semantic databases such as WordNet.

One approach to this multivariate prediction problem, especially if our goal is to maximize the number of y_s that are correctly classified, is to learn a per-position classifier that maps $\mathbf{x} \mapsto y_s$ for each s. There are two difficulties with this method, however. The first is that we are not always interested in maximizing the number of correct predictions. Sometimes the objective function over predictions may be to maximize the probability that the entire sequence is correct, or to maximize a more complicated function like BLEU or F_1. The second difficulty is that the size of both the input and output vectors can be extremely large. For example, in part-of-speech tagging, each vector $\mathbf{x}_s$ may have tens of thousands of components, so a classifier based on all of $\mathbf{x}$ would have many parameters. But using only $\mathbf{x}_s$ to predict y_s is also bad, because information from neighboring feature vectors is also useful in making predictions. Both of these difficulties can be addressed by explicitly modeling the dependence between outputs, so that a confident prediction about one variable needs to be able to influence nearby, possibly less confident predictions.

A natural way to represent the manner in which 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 of the early work in learning with graphical models, especially in statistical natural-language processing, focused on *generative* models that explicitly attempted to model a joint probability distribution over inputs and outputs. Although there are advantages to this approach, it also has important limitations. Not only can the dimensionality of $\mathbf{x}$ become very large, but the features have complex dependencies, so constructing a probability distribution over them is difficult. In practice, one must either employ a complex model of the features, which is intractable, or make strong independence assumptions among the features, which can lead to reduced performance.

An alternative approach is to predict $\mathbf{y}$ directly, without modeling $\mathbf{x}$. This is the idea behind *structured prediction*. Structured prediction methods are essentially a combination of classification and graphical modeling, combining the ability to compactly model multivariate data with the ability to perform prediction using large sets of input features. The idea is, for an input $\mathbf{x}$, to define a discriminant function $F_{\mathbf{x}}(\mathbf{y})$, and predict $\mathbf{y}^* = \arg\max_{\mathbf{y}} F_{\mathbf{x}}(\mathbf{y})$. This function factorizes according to a set of local factors, just as in graphical models. But as in classification, each local factor is modeled a linear function of $\mathbf{x}$, although perhaps in some induced high-dimensional space. To understand the benefits of this approach, consider a hidden Markov model (formally introduced in Section 2.2.2) and a set of per-position classifiers, both with fixed parameters. In principle, the per-position classifiers predict an output y_s given all of $\mathbf{x}_0 \dots \mathbf{x}_T$.[1] In the HMM, on the other hand, to predict y_s it is statistically sufficient to know only the local input $\mathbf{x}_s$, the previous forward message $p(y_{s-1}, \mathbf{x}_0 \dots \mathbf{x}_{s-1})$, and the backward message $p(\mathbf{x}_{s+1} \dots \mathbf{x}_T | y_s)$. So the forward and backward messages serve as a summary of the rest of the input, a summary that is generally non-linear in the observed features.

1.1 Approaches to Structured Prediction

Several types of structured prediction algorithms have been studied. All such algorithms assume that the discriminant function $F_{\mathbf{x}}(\mathbf{y})$ over labels can be written as a sum of local functions $F_{\mathbf{x}}(\mathbf{y}) = \sum_a f_a(\mathbf{y}_a, \mathbf{x}, \theta)$. The task is to estimate the real-valued parameter vector θ given a training set $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$. The methods differ in how the parameters are selected.

In this thesis, I focus on *conditional random fields (CRFs)* [58], in which the score $F_{\mathbf{x}}(\mathbf{y})$ is viewed as defining a conditional probability distribution $p(\mathbf{y}|\mathbf{x}) \propto \exp\{F_{\mathbf{x}}(\mathbf{y})\}$. As we see in detail in Chapter 2, training requires computing marginal distributions, which are intractable in general. The task of dealing with this intractable will be the main focus of this thesis. Perhaps the main advantage of prob-

[1]To be fair, in practice the classifier for y_s would probably depend only on a sliding window around $\mathbf{x}_s$, rather than all of $\mathbf{x}$. But still the structured approach has the advantage that the forward and backward messages serve as a flexible, nonlinear summary of the surrounding input. This has the effect of choosing an effective window size from the data.

abilistic methods is that they can incorporate latent variables in a natural way, by marginalization. A particularly powerful example of this is provided by Bayesian methods, in which the model parameters themselves are integrated out.

Alternative structured prediction methods are based on maximizing over assignments rather than marginalizing. Perhaps the most popular of these methods has been *maximum-margin* methods that are so successful for univariate classification. Recently max-margin methods have been generalized to the structured case [3, 129]. Both batch and online algorithms are available to maximize this objective function. The perceptron update can also be generalized to structured models [22]. 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 [25], which may perform better than the perceptron update.

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

A general advantage of all of these maximization-based methods is that they do not require summation over all configurations for the partition function or for marginal distributions. There are certain combinatorial problems, such as matching and network flow problems, in which finding an optimal configuration is tractable, but summing over configurations is not (for an example of applying max-margin methods in such situations, see Taskar et al. [131]). For more complex problems, neither summation nor maximization is tractable, so this advantage is perhaps not as significant. Another advantage of these methods is that kernels can be naturally incorporated.

Finally, LeCun et al. [60] generalizes many prediction methods, including the ones listed above, under the rubric of *energy-based* methods, and presents interesting historical information about their use. They advocate changing the loss function to avoid probabilities altogether, and so their work may serve as an interesting complement to my work in this thesis.

An older method for predicting multiple outputs simultaneously is neural networks. The main difference between the more recent structured prediction work and neural networks 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. Therefore, the main insight of structured models can be expressed in the language of neural networks as: If you add connections among the nodes in the output layer, then in some problems you do not need a hidden layer to get good performance. Omitting the hidden layer has great computational advantages, because the objective functions used for training become convex, and we do not need to worry about the problems of local minima that arise when training neural networks. For harder problems, however, one might expect that even after modeling output structure, incorporating hidden state will still yield additional benefit. Once hidden state is introduced into the model, whether it be a neural network or a structured model, the loss of convexity is inevitable. There are currently few examples of structured models with latent variables (for exceptions, see Quattoni et al. [97] and McCallum et al. [73]), but it is likely that such models will become more important in the future.

The differences between the various structured prediction methods are not well understood. For example, I am not aware of any empirical study that compares these algorithms on a broad range of data sets. Indeed, my presentation in this section is motivated by the view that the similarities between various structured prediction methods are more important than the differences. For this reason, my choice in this thesis to focus on conditional random fields is difficult to justify, but perhaps of secondary importance. As I explain next, my main area of interest is to incorporate approximate inference algorithms into training of models with intractable structure, and most structured prediction methods do require performing inference during training.

1.2 Efficient Training of CRFs

Early work on CRFs focused on the case in which the variables $\mathbf{y}$ are arranged in a linear chain, because this choice allows the marginals of the distribution to be computed exactly using the forward-backward algorithm, and because this choice is very natural for certain tasks such as information extraction and shallow parsing. Recently, however, research in NLP has begun to explore global models, which exploit long-distance dependencies between words to improve performance [14, 32, 118]. With such rich models, however, a major difficulty with CRFs becomes the amount of computation required for training. This is because the likelihood gradient requires matching the marginal distributions of the model to those of the training data, and computing the model marginals is intractable for general graphs. In addition, the amount of training data can be somewhat large, with the largest data sets reaching one million words of labeled training data. The standard approach for dealing with this is to approximate the model marginals, most commonly using variational methods such as belief propagation, although many other methods are also possible.

Because inference in graphical models is intractable, there is a vast literature on approximate inference. Here I mention a few recent studies that are relevant to efficient training methods in undirected models and in conditional random fields.

For generative models, Abbeel et al. [1] present parameter estimation and structure learning algorithms that, unlike maximum likelihood, require polynomial time and have polynomial sample complexity. These algorithms are not yet practical, however, because they make poor use of the training data.

Remarkably, the pseudo-moment matching estimator of Wainwright et al. [142] computes a parameter setting that maximizes the BP approximation to the likelihood without ever requiring any of the message updates to be computed. This estimator appears to have very limited practical applicability, however, because it applies only to generative random fields with a special parameterization that does not allow tied parameters. In this thesis I present a generalization that handles these issues (Section 4.1.3).

Although the original work on CRFs used iterative scaling, it was later found that second-order gradient-based methods converged much faster [112]. Recently, however, online methods have been shown to converge much faster than second-order methods for CRFs [136]. Globerson et al. [42] present a particularly interesting method using the dual of the likelihood, provably finding the minimum if the step size is small enough, but with an online-like update. The difference between these methods and my work is that they focus on the case where there are many iid training instances, not when there is intractable structure. In cases where there are both many training examples and intractable structure, then one of these online-style updates could be incorporated orthogonally into the methods presented in this thesis.

1.3 Main Contributions

The main contributions of this thesis are:

- **Modeling** (Chapter 3). I present two novel classes of loopy conditional models: dynamic conditional random fields (DCRFs) and skip-chain CRFs. A common theme of these models is to show that adding long-distance dependencies between words can improve performance in sequence labeling tasks. These models also motivate the need for more efficient training methods for CRFs.

- **Piecewise Training** (Chapter 4). I introduce piecewise training, a method for training regions of a factor graph separately and combining them at test time. This method has appeared as a heuristic in scattered places in the literature, but has never been studied systematically. On several benchmark NLP tasks, I show that the factor-as-piece approximation performs surprisingly well, always exceeding pseudolikelihood and sometimes rivaling exact maximum likelihood. In addition, piecewise training can be understood as a generalization of the pseudo-moment matching estimator of Wainwright et al. [142] that allows for conditional models with arbitrary parameterization. This provides a satisfying theoretical explanation of the positive experimental results of the factor-as-piece approximation.

- **Piecewise Pseudolikelihood** (Chapter 5). Piecewise training scales poorly in computation time to variables or factors that have large cardinality. Pseudolikelihood scales much better in computation time, but has poor accuracy in the NLP data that I consider. Therefore I propose a hybrid method called *piecewise pseudolikelihood (PWPL)*, which "pseudolikelihoodizes" (in a sense that can be stated formally) the individual terms in the piecewise likelihood. I show that under certain conditions, PWPL converges to the piecewise solution in the limit of infinite data. In terms of accuracy, PWPL performs more like piecewise training than like pseudolikelihood, making it a practical alternative to pseudolikelihood for the problems considered here.

- **Improved Dynamic Schedules for Belief Propagation** (Section 6.1). In Chapter 4, I show how piecewise training can be viewed as a type of early stopping of belief propagation. But the running time of BP and even its convergence depend greatly on the schedule used to send the messages. Dynamic update schedules have recently been shown to converge much faster on hard networks than static schedules by Elidan et al. [30], who propose a simple and effective schedule which they call residual BP. But that RBP algorithm wastes message updates: many messages are computed solely to determine their priority, and are never actually performed. I show that estimating the residual, rather than calculating it directly, leads to significant decreases in the number of messages required for convergence, and in the total running time. The residual is estimated using an upper bound based on recent work on message errors in BP. On both synthetic and real-world networks, this dramatically decreases the running time of BP, in some cases by a factor of five, without affecting the quality of the solution.

- **Integrating Inference and Learning** (Section 6.2). Piecewise training can be seen as a sort of early stopping of BP, one in which BP is stopped before sending any messages. Therefore it is natural to ask whether one can do better by using some less drastic form of early stopping. I propose a view of such methods as attempting to find fixed points of a single system of equations, which includes both gradient updates on the model parameters and BP message updates. Update schedules for solving this system can be seen as integrating inference and learning, because they have the freedom to dynamically choose when to make parameter updates and when to send messages.

1.4 Declaration of Previous Work

Most the work of this thesis has been previously published. The rest has been collected into several technical reports. These are:

- Much of the tutorial information in Chapter 2 has been published in Sutton and McCallum [121].

- The work on DCRFs in Section 3.1 has appeared as Sutton, McCallum, and Rohanimanesh [126], which was based on two earlier conference papers [120, 125].
- The work on skip-chain CRFs in Section 3.2 appeared as Sutton and McCallum [118].
- The initial work on piecewise training was Sutton and McCallum [119].
- The work on shared-unary piecewise and one-step cutout (Section 4.4 and 4.5) was done at Microsoft Research in collaboration with Tom Minka, and appears in an MSR tech report [124].
- Chapter 5 on piecewise pseudolikelihood was originally published as Sutton and McCallum [122].
- The work on zero-lookahead RBP0 in Section 6.1 has been published as Sutton and McCallum [123].

CHAPTER 2
BACKGROUND

This chapter provides the statistical and algorithmic background necessary to understand the current work. I review necessary concepts in graphical models and inference (Section 2.1), and then explain conditional random fields, starting with the linear-chain case (Section 2.3) and then describing CRFs in general (Section 2.4).

2.1 Graphical Models

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 [76], computer vision [38], knowledge representation [91], Bayesian statistics [37], and natural-language processing [11, 58].

Distributions over many variables can be very 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 even when a distribution is defined over a large set of variables, it can often be represented as a product of *local functions* that 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, I introduce graphical models from both the factorization and conditional independence viewpoints, focusing on those models which are based on undirected graphs, because such models are the principal topic of this thesis. I also discuss a few approximate inference algorithms that are useful in the present work.

2.1.1 Undirected Graphical Models

We consider probability distributions over sets of random variables $V = X \cup Y$, where X is a set of *input variables* that we assume are observed, and Y is a set of *output variables* that we wish to predict. Every variable $s \in V$ takes outcomes from a set $\mathcal{V}$, which can be either continuous or discrete, although I consider only the discrete case in this thesis. An arbitrary assignment to X is denoted by a vector $\mathbf{x}$. Given a variable $s \in X$, the notation x_s denotes the value assigned to s by $\mathbf{x}$, and similarly for an assignment to a subset $a \subset X$ by $\mathbf{x}_a$. The notation $\mathbf{1}_{\{x=x'\}}$ denotes an indicator function of x which takes the value 1 when $x = x'$ and 0 otherwise. We also require a special notation for marginalization. For a fixed variable assignment y_s, we use the summation $\sum_{\mathbf{y}\backslash y_s}$ to indicate a summation over all possible assignments $\mathbf{y}$ whose value for variable s is equal to y_s.

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

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

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

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

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

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

The quantity Z, considered as a function of the set F of factors, is called the *partition function* in the statistical physics and graphical models communities. Computing Z is intractable in general, but much work exists on how to approximate it.

We will generally assume further that each local function has the form

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

for some real-valued parameter vector θ_a, and for some set of *feature functions* or *sufficient statistics* $\{f_{ak}\}$. If $\mathbf{x}$ and $\mathbf{y}$ are discrete, then this is no restriction, because we can have features have indicator functions for every possible value, that is, if we include one feature function $f_{ak}(\mathbf{x}_a, \mathbf{y}_a) = \mathbf{1}_{\{\mathbf{x}_a = \mathbf{x}_a^*\}} \mathbf{1}_{\{\mathbf{y}_a = \mathbf{y}_a^*\}}$ for every possible value $\mathbf{x}_a^*$ and $\mathbf{y}_a^*$.

Also, a consequence of this parameterization is that the family of distributions over V parameterized by θ is an exponential family. Indeed, much of the discussion of maximum-likelihood parameter estimation in this chapter applies to exponential families in general.

want

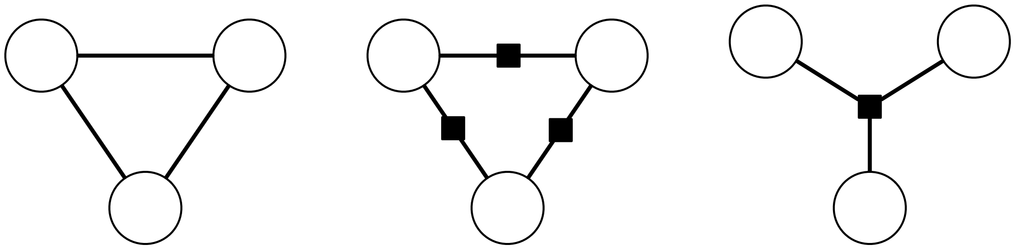


Figure 2.1. A Markov network with an ambiguous factorization. Both of the factor graphs at right factorize according to the Markov network at left.

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

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

In other words, if p factorizes according to G, then p is Markov with respect to G. The converse direction also holds, as long as p is strictly positive. This is stated in the following classical result [7, 45]:

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

A Markov network has an undesirable ambiguity from the factorization perspective, however. Consider the three-node Markov network in Figure 2.1 (left). Any distribution that factorizes as $p(x_1, x_2, x_3) \propto a(x_1, x_2, x_3)$ for some positive function a is Markov with respect to this graph. However, we may wish to use a more restricted parameterization, where $p(x_1, x_2, x_3) \propto a(x_1, x_2)b(x_2, x_3)c(x_1, x_3)$. This second parameterization denotes a smaller set of models, which therefore may be more amenable to parameter estimation. But the Markov network formalism cannot distinguish between these two parameterizations. In order to state models more precisely, the factorization (2.1) can be represented directly by means of a *factor graph* [54]. A factor graph is a bipartite graph $G = (V, F, E)$ in which a variable node $v_s \in V$ is connected to a factor node $\Psi_a \in F$ if v_s is an argument to Ψ_a. An example of a factor graph is shown graphically in Figure 2.2 (right). In that figure, the circles are variable nodes, and the shaded boxes are factor nodes.

2.1.2 Directed Graphical Models

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

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

where $\pi(v)$ are the parents of v in G. An example of a directed model is shown in Figure 2.2 (left). It can be shown by structural induction on G that p is properly normalized. Directed models are often used as generative models, as we explain in Section 2.2.3.

2.1.3 Inference

The fundamental problem in graphical models is *inference*, that is, given a specification of the joint distribution $p(\mathbf{y})$ over all the variables, to compute the resulting *marginal distribution* over subsets of Y. (In order to simplify notation, we have omitted the variables X in this section.) There are two inference problems of interest in this thesis: first, computing the marginal distributions of single variables

$$p(y_s) = \sum_{\mathbf{y} \backslash y_s} p(\mathbf{y}), \tag{2.5}$$

and second, that of computing *max-marginals*

$$\delta(y_s) = \max_{\mathbf{y} \backslash \mathbf{y}_s} p(\mathbf{y}). \tag{2.6}$$

We will also be interested in the problem of computing marginals over factors, that is, $p(y_a) = \sum_{\mathbf{y} \backslash \mathbf{y}_a} p(\mathbf{y})$ where the set $a \subset V$ is the scope of some factor.

These two inference problems can be seen as fundamentally the same operation on two different semirings [2], that is, to change the marginal problem to the max-marginal problem, we simply substitute max for plus. Indeed, many algorithms for computing marginals have analogous procedures for computing max-marginals. Although for discrete variables the marginals can be computed by brute-force summation, the time required to do this is exponential in the size of Y. Indeed, both inference problems are intractable for general graphs, because any propositional satisfiability problem can be easily represented as a factor graph.

Exact inference algorithms are known for general graphs. Although these algorithms require exponential time in the worst case, they can still be efficient for many graphs that occur in practice. The most popular such algorithm, the junction tree algorithm, successively clusters variables until the graph becomes a tree. Once such an equivalent tree has been constructed, its marginals can be computed using exact inference algorithms that are specific to trees, one of which is described in the next section. For certain complex graphs, the junction tree algorithm is forced to make clusters which are very large, so that on such graphs the procedure requires exponential time. For more details on exact inference, see Cowell et al. [24].

For this reason, an enormous amount of effort has been devoted to approximate inference algorithms. Two classes of approximate inference algorithms have received the most attention: *Monte Carlo* algorithms, that attempt to sample from the distribution of interest; and *variational* algorithms, that convert the inference problem into an optimization problem, which is then relaxed or approximated until it becomes tractable. Generally, Monte Carlo algorithms 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 faster, but they tend to be biased, by which I 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, I focus on variational algorithms in this thesis, for two reasons. First, parameter estimation requires performing inference many times, and so a fast inference procedure is vital to efficient training. Second, a natural connection exists between variational inference algorithms and performing parameter estimation on subgraphs, which is one of the central ideas in this thesis (Chapter 4).

2.1.4 Belief Propagation

An important variational inference algorithm for is *belief propagation (BP)*, which I explain in this section. I choose to explain BP in detail for two reasons: First, it is a direct generalization of the exact inference algorithms for linear-chain CRFs. Second, and more important for the purposes of this thesis, it has a close connection to the piecewise methods that I introduce in Chapter 4.

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

$$p(y_s) \propto \sum_{\mathbf{y} \backslash y_s} \prod_a \Psi_a(\mathbf{y}_a) \tag{2.7}$$

$$= \prod_{a \in N(s)} \sum_{\mathbf{y}_{V_a}} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b) \tag{2.8}$$

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

$$m_{as}(x_s) = \sum_{\mathbf{y}_{V_a}} \prod_{\Psi_b \in F_a} \Psi_b(\mathbf{y}_b), \tag{2.9}$$

can be thought of as a *message* from the factor a to the variable s that summarizes the impact of the network upstream of a on the belief in s. In a similar fashion, we can define messages from variables to factors as

$$m_{sA}(x_s) = \sum_{\mathbf{y}_{V_s}} \prod_{\Psi_b \in F_s} \Psi_b(\mathbf{y}_b). \tag{2.10}$$

Then, from (2.8), we have that the marginal $p(y_s)$ is proportional to the product of all the incoming messages to variable s. Similarly, factor marginals can be computed as

$$p(\mathbf{y}_a) \propto \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(\mathbf{y}_a). \tag{2.11}$$

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

Naively computing the messages according to (2.9) is impractical, because the messages as we have defined them require summation over possible many variables in the graph. Fortunately, the messages can also be written using a recursion that requires only local summation. The recursion is

$$\begin{aligned} m_{as}(x_s) &= \sum_{\mathbf{y}_a \backslash y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in a \backslash s} m_{ta}(x_t) \\ m_{sa}(x_s) &= \prod_{b \in N(s) \backslash a} m_{bs}(x_s) \end{aligned} \tag{2.12}$$

That this recursion matches the explicit definition of m can be seen by repeated substitution, and proven by induction. In a tree, it is possible to schedule these recursions such that the antecedent messages are always sent before their dependents, by first sending messages from the root, and so on. This is the algorithm known as *belief propagation* [91].

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

$$p(\mathbf{y}_a) = \kappa \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(y_s), \tag{2.13}$$

where κ is a proportionality constant, computed to make the distribution sum to 1. In fact, a similar idea works for any connected set of variables—not just a set that happens to be the domain of some factor—although if the set is too large, then computing κ is impractical.

This comment motivates the second problem, computing joint probabilities $p(\mathbf{y})$. Perhaps the most convenient way to do this is to use the fact that in a tree-structured distribution

$$p(\mathbf{y}) = \prod_{s \in Y} p(y_s) \prod_a \frac{p_a(\mathbf{y}_a)}{\prod_{t \in a} p(y_t)} \tag{2.14}$$

This is because any tree can be represented as a junction tree with one cluster for each factor. Using this identity, we can compute $p(\mathbf{y})$ (or $\log Z$) from the per-variable and per-factor marginals.

The preceding discussion assumes that the graph G is a tree. If G is not a tree, the message updates (2.12) 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, I refer to the approximate marginals that result from loopy BP as *beliefs* rather than as marginals, and denote them by $q(y_s)$.

Surprisingly, loopy BP can be seen as a variational method as follows. The general variational idea is to:

1. Define a family of tractable distributions $\mathcal{Q}$ and an objective function $\mathcal{O}(q)$. The function $\mathcal{O}$ should be designed to measure how well a tractable distribution $q \in \mathcal{Q}$ approximates the distribution p of interest.

2. Find the "closest" tractable distribution $q^* = \min_{q \in \mathcal{Q}} \mathcal{O}(q)$.

3. Use the marginals of q^* to approximate those of p.

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

$$\mathcal{O}(q) = \mathrm{KL}(q\|p) - \log Z \tag{2.15}$$

$$= -H(q) - \sum_a q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a). \tag{2.16}$$

Then the solution to this variational problem is $q^* = p$ with optimal value $\mathcal{O}(q^*) = \log Z$. Solving this particular variational formulation is thus equivalent to performing exact inference. Approximate inference techniques can be devised by changing the set $\mathcal{Q}$—for example, by requiring q to be fully factorized—or by using a different objective $\mathcal{O}$.

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

$$H_{\text{BETHE}}(q) = \sum_a q(\mathbf{y}_a) \log q(\mathbf{y}_a) + \sum_i (1 - d_i) q(y_i) \log q(y_i). \tag{2.17}$$

This follows from substituting the junction-tree formulation (2.14) of the joint into the definition of entropy. If q is not a tree, then we can still take H_{BETHE} as an approximation to H to compute the exact variational objective $\mathcal{O}$. This yields the *Bethe free energy*:

$$\mathcal{O}_{\text{BETHE}}(q) = H_{\text{BETHE}}(q) - \sum_a q(\mathbf{y}_a) \log \Psi_a(\mathbf{y}_a) \tag{2.18}$$

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* [138]. However, for intractable models, the marginal polytope can have extremely complex structure.

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

$$\sum_{\mathbf{y}_a \setminus y_i} q_a(\mathbf{y}_a) = q_i(y_i) \qquad \forall a, i \in a \tag{2.19}$$

Under these constraints, Yedidia et al. [149] show that constrained stationary points of $\mathcal{O}_{\text{BETHE}}$ fixed points of loopy BP. So we can view the Bethe energy $\mathcal{O}_{\text{BETHE}}$ as an objective function that the loopy BP fixed-point operations attempt to optimize.

A second way of formulating BP as a variational algorithm yields a dual form of the Bethe energy that will prove particularly useful [81]. This dual energy arises from the expectation propagation view of BP [77]. Suppose we have a set of BP messages $\{m_{ai}\}$, which have not necessarily converged. Then we view the outgoing messages from each factor as approximating it, that is,

$$\Psi_a(\mathbf{y}_a) \approx \tilde{t}_a(\mathbf{y}_a) = \prod_{i \in a} m_{ai}(y_i). \tag{2.20}$$

This yields an approximation to the distribution p, namely, $p(\mathbf{y}) \approx q(\mathbf{y}) = \prod_a \tilde{t}_a(\mathbf{y}_a)$. Observe that q is possibly unnormalized. As Minka [77] observes, each message update of loopy BP can be viewed as refining one of the terms $\tilde{t}_a$ so that q is closer, in terms of KL divergence.

Since q was therefore chosen to approximate p, it makes sense to use the mass of q to approximate the mass of p. More precisely, let p' be the unnormalized version of p, that is, $p'(\mathbf{y}) = \prod_a \Psi_a(\mathbf{y}_a)$. Then define rescaled versions of $\tilde{t}_a$ and q as

$$\bar{t}_a(\mathbf{y}_a) = s_a \tilde{t}_a(\mathbf{y}_a) \tag{2.21}$$

$$\bar{q}(\mathbf{y}) = \prod_a \bar{t}_a(\mathbf{y}_a) \tag{2.22}$$

Then the idea is to scale each of the $\bar{t}_a$ so that the resulting $\sum_{\mathbf{y}} \bar{q}(\mathbf{y})$ matches as closely as possible the partition function $\sum_{\mathbf{y}} p'(\mathbf{y})$. This can be done by optimizing local divergences in an analogous manner to EP. Define $\bar{q}^{\backslash a}$ as the approximating $\bar{q}$ without the factor $\tilde{t}_a$, that is, each s_a is separately chosen to optimize

$$\min_{s_a} \mathrm{KL}(\Psi_a(\mathbf{y}_a)\bar{q}^{\backslash a}(\mathbf{y}_a) \| s_a \tilde{t}_a(\mathbf{y}_a)\bar{q}^{\backslash a}(\mathbf{y}_a)). \tag{2.23}$$

Observe that because $\bar{q}^{\backslash a}$ depends on all of the scale factors s_b for all factors b, the local objective function depends on all of the other scale factors as well. The optimal s_a is given by

$$s_a = \frac{\sum_{\mathbf{y}} \frac{\Psi_a(\mathbf{y}_a)}{\tilde{t}(\mathbf{y}_a)} q(\mathbf{y})}{\sum_{\mathbf{y}} q(\mathbf{y})}. \tag{2.24}$$

Thus the optimal s_a actually does not depend on the other scale values. Now taking the integral $\sum_{\mathbf{y}} \bar{q}(\mathbf{y})$ yields the following approximation to the partition function

$$Z_{\textsc{BetheDual}} = \prod_i \left(\sum_{y_i} q_i(y_i) \right)^{1-d_i} \prod_a \left(\sum_{\mathbf{y}_a} \frac{\Psi_a(\mathbf{y}_a)}{\tilde{t}(\mathbf{y}_a)} q(\mathbf{y}_a) \right). \tag{2.25}$$

It can be shown [78] that this is also a free energy for BP, that is, that fixed points of BP are stationary points of this objective.

Another view of BP is the reparameterization viewpoint [141]. In this view, the BP updates are expressed solely in terms of the beliefs b^n at each iteration n of the algorithm. The beliefs are initialized as $b_a^0 \propto \Psi_a$ for all factors, and $b_s^0 \propto 1$ for all variables. The updates at iteration n are

$$\begin{aligned} b_s^n(y_s) &= \sum_{\mathbf{y}_{N(s)}} B_s^{n-1}(y_s, \mathbf{y}_{N(s)}) \\ b_a^n(\mathbf{y}_a) &= \sum_{\mathbf{y}_{N(a)}} B_a^{n-1}(\mathbf{y}_a, \mathbf{y}_{N(a)}) \end{aligned} \tag{2.26}$$

where the distributions B_a^{n-1} and B_s^{n-1} are defined as

$$\begin{aligned} B_s^{n-1}(y_s, \mathbf{y}_{N(s)}) &\propto b^{n-1}(y_s) \prod_{a \ni s} \frac{b_a^{n-1}(\mathbf{y}_a)}{b_s^{n-1}(y_s)} \\ B_a^{n-1}(\mathbf{y}_a, \mathbf{y}_{N(a)}) &\propto b_a^{n-1}(\mathbf{y}_a) \prod_{s \in a} \prod_{c \ni s \setminus a} \frac{b_c(\mathbf{y}_c)}{b_s(y_s)} \end{aligned} \tag{2.27}$$

(This notation is adapted from Rosen-Zvi et al. [103].)

To see how this corresponds to the message-based recursions (2.12), consider the following message passing schedule. At each iteration n, first compute all of the to-variable messages simultaneously as

$$m_{as}^n(y_s) = \sum_{\mathbf{y}_a \setminus y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in a \setminus s} m_{ta}^{n-1}(y_t) \tag{2.28}$$

then compute all of the to-factor messages as

$$m_{sa}^n(y_s) = \prod_{c \ni s \setminus a} m_{cs}^n(y_s). \tag{2.29}$$

Then define the beliefs as usual: $q_s^n \propto \prod_{a \ni s} m_{as}^n$ for all variables s, and $q_a^n \propto \Psi_a \prod_{s \in a} \prod_{c \ni s \setminus a} m_{cs}^n$ for all factors a. In order to obtain a clean correspondence between these reparameterization updates, we also need the assumption that the graphs defined by B_a^{n-1} and B_s^{n-1} are trees, which always happens if all factors are pairwise.

Now we can state the correspondence between the message-based and reparameterization viewpoints: The beliefs from the message-based recursions are equal to those from the reparameterization-based recursions. That is, for all iterations m, $q_s^n = b_s^n$ and $q_a^n = b_a^n$. This can be seen by induction, substituting the messages corresponding to q^{n-1} into the update equations (2.26) for b^n.

The reason for the term "reparameterization" is that at each iteration n, we can construct a distribution $T^n(\mathbf{y})$ over the full space with factors

$$T_s^n = b_s^n, \qquad T_a^n = \frac{b_a^n}{\prod_{s \in a} b_s^n}. \tag{2.30}$$

This distribution is invariant under the message update, that is, $T^n = T^{n-1} = \cdots = T^0 = p$. So each T^n can be viewed as a reparameterization of the original distribution. This view of BP will prove especially useful in Section 4.1.3.

2.2 Applications of Graphical Models

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

2.2.1 Classification

First we discuss the problem of *classification*, that is, predicting a single 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.

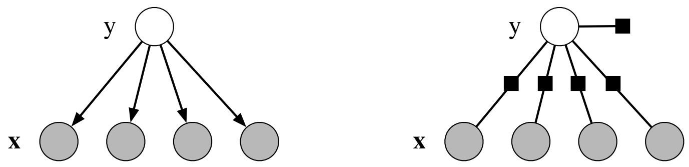


Figure 2.2. The naive Bayes classifier, as a directed model (left), and as a factor graph (right).

The resulting classifier is called the *naive Bayes classifier.* It is based on a joint probability model of the form:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^{K} p(x_k|y). \tag{2.31}$$

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

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

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{ \lambda_y + \sum_{j=1}^{K} \lambda_{y,j} x_j \right\}, \tag{2.32}$$

where $Z(\mathbf{x}) = \sum_y \exp\{\lambda_y + \sum_{j=1}^{K} \lambda_{y,j} x_j\}$ is a normalizing constant, and λ_y is a bias weight that acts like $\log p(y)$ in naive Bayes. Rather than using one weight vector per class, as in (2.32), we can use a different notation in which a single set of weights is shared across all the classes. The trick is to define a set of *feature functions* that are nonzero only for a single class. To do this, the feature functions can be defined as $f_{y',j}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}} x_j$ for the feature weights and $f_{y'}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}}$ for the bias weights. Now we can use f_k to index each feature function $f_{y',j}$, and λ_k to index its corresponding weight $\lambda_{y',j}$. Using this notational trick, the logistic regression model becomes:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{ \sum_{k=1}^{K} \lambda_k f_k(y, \mathbf{x}) \right\}. \tag{2.33}$$

We introduce this notation because it mirrors the usual notation for conditional random fields.

2.2.2 Sequence Models

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

One approach to NER is to classify each word independently as one of either PERSON, LOCATION, ORGANIZATION, or OTHER (meaning not an entity). The problem with this approach is that it assumes that given the input, all of the named-entity labels are independent. In fact, the named-entity labels of neighboring words are dependent; for example, while *New York* is a location, *New York Times* is an organization. This independence assumption can be relaxed by arranging the output variables in a linear chain. This is the approach taken by the hidden Markov model (HMM) [98]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^{T}$ by assuming that there is an underlying sequence of *states* $Y = \{y_t\}_{t=1}^{T}$ drawn from a finite state set S. In the named-entity example, each observation x_t is the identity of the word at position t, and each state y_t is the named-entity label, that is, one of the entity types Person, Location, Organization, and Other.

To model the joint distribution $p(\mathbf{y}, \mathbf{x})$ tractably, an HMM makes two independence assumptions. First, it assumes that each state depends only on its immediate predecessor, that is, each state y_t is independent of all its ancestors $y_1, y_2, \ldots, y_{t-2}$ given the preceding state y_{t-1}. Second, an HMM 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

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1}) p(x_t|y_t), \tag{2.34}$$

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

2.2.3 Discriminative and Generative Models

An important difference between naive Bayes and logistic regression is that naive Bayes is *generative*, meaning that it is based on a model of the joint distribution $p(y, \mathbf{x})$, while logistic regression is *discriminative*, meaning that it is based on a model of the conditional distribution $p(y|\mathbf{x})$. In this section, we discuss the differences between generative and discriminative modeling, and the potential advantages of discriminative modeling. For concreteness, we focus on the examples of naive Bayes and logistic regression, but the discussion in this section actually applies in general to the differences between generative models and conditional random fields.

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

The principal advantage of discriminative modeling is that it is better suited to including rich, overlapping features. To understand this, consider the family of naive Bayes distributions (2.31). This is a family of joint distributions whose conditionals all take the "logistic regression form" (2.33). But there are many other joint models, some with complex dependencies among $\mathbf{x}$, whose conditional distributions also have the form (2.33). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(\mathbf{x})$. This may explain why it has been observed that conditional random fields tend to be more robust than generative models to violations of their independence assumptions [58]. Simply put, CRFs make independence assumptions among $\mathbf{y}$, but not among $\mathbf{x}$.

To include interdependent features in a generative model, we have two choices: enhance the model to represent dependencies among the inputs, or make simplifying independence assumptions, such as the naive Bayes assumption. The first approach, enhancing the model, is often difficult to do while retaining tractability. For example, it is hard to imagine how to model the dependence between the capitalization of a word and its suffixes, nor do we particularly wish to do so, since we always observe the test sentences anyway. The second approach—to include a large number of dependent features in a generative model, but to include independence assumptions among them—is possible, and in some domains can work well. But it can also be problematic because the independence assumptions can hurt performance. For example, although the naive Bayes classifier performs well in document classification, it performs worse on average across a range of applications than logistic regression [17].

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

Actually, 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 (2.31) defines the same family of distributions as the logistic regression model (2.33), if we interpret it generatively as

$$p(y, \mathbf{x}) = \frac{\exp\left\{\sum_k \lambda_k f_k(y, \mathbf{x})\right\}}{\sum_{\tilde{y}, \tilde{\mathbf{x}}} \exp\left\{\sum_k \lambda_k f_k(\tilde{y}, \tilde{\mathbf{x}})\right\}}. \tag{2.35}$$

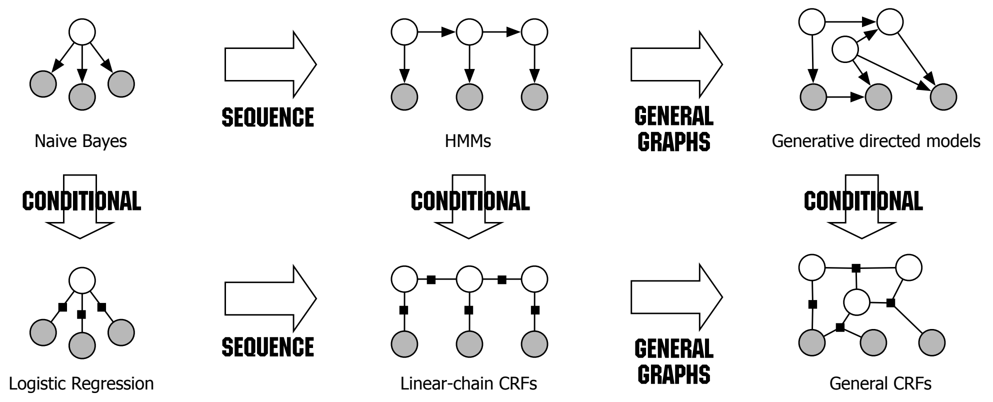


Figure 2.3. Diagram of the relationship between naive Bayes, logistic regression, HMMs, linear-chain CRFs, generative models, and general CRFs.

This means that if the naive Bayes model (2.31) 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 (2.35), 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 [88], naive Bayes and logistic regression form a *generative-discriminative pair*.

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

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta) p_g(\mathbf{x}|\mathbf{y}; \theta). \tag{2.36}$$

But we could also rewrite p_g using Bayes rule as

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta) p_g(\mathbf{y}|\mathbf{x}; \theta), \tag{2.37}$$

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

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

$$p_c(\mathbf{y}, \mathbf{x}) = p_c(\mathbf{x}; \theta')p_c(\mathbf{y}|\mathbf{x}; \theta). \tag{2.38}$$

By comparing (2.37) with (2.38), it can be seen that the conditional approach has more freedom to fit the data, because it does not require that $\theta = \theta'$. Intuitively, because the parameters θ in (2.37) are used in both the input distribution and the conditional, a good set of parameters must represent both well, potentially at the cost of trading off accuracy on $p(\mathbf{y}|\mathbf{x})$, the distribution we care about, for accuracy on $p(\mathbf{x})$, which we care less about. On the other hand, this added freedom brings about an increased risk of overfitting the training data, and generalizing worse on unseen data.

So far I have tried provide intuition on why discriminative models can have better accuracy than generative models. To be fair, however, generative models have several advantages of their own. First, generative models tend to be able to incorporate partially-labeled or semi-supervised data more naturally, although there has been work on incorporating both in discriminative models. In the most extreme case, when the data is entirely unlabeled, generative models can be applied in an unsupervised fashion, which can sometimes yield insight into the data, whereas a discriminative model is not useful in this case. Second, on some data a generative model can perform better than a discriminative model, intuitively because the input model $p(\mathbf{x})$ may have a smoothing effect on the conditional. Ng and Jordan [88] 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 future inputs and outputs, making a generative model preferable.

It is often natural to represent generative models by a directed graphs in which in outputs $\mathbf{y}$ topologically precede the inputs. Thus, the generative model describes how the outputs probabilistically "generate" the inputs. An example of this is the hidden Markov model, discussed in Section 2.2.2. Similarly, as we will see, discriminative models are often naturally described by undirected graphs. This correspondence need not always hold, however. The key distinction is whether the parameters that control $p(\mathbf{x})$ and $p(\mathbf{y}|\mathbf{x})$ are forced to be identical, as in a directed model, or are modeled as independent, as in a discriminative model. Indeed, hybrids of these two regimes are also possible [59].

In this section, we have discussed the relationship between naive Bayes and logistic regression in detail because it 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 analog to hidden Markov models, and this analog is a particular type of conditional random field, as we explain next. The analogy between naive Bayes, logistic regression, generative models, and conditional random fields is depicted in Figure 2.3.

2.3 Linear-Chain Conditional Random Fields

In the previous section, we have seen advantages both to discriminative modeling and to sequence modeling. So it makes sense to combine the two. This yields a linear-chain CRF, which we describe in this section. First, in Section 2.3.1, we define linear-chain CRFs, motivating them from HMMs. Then, we discuss parameter estimation (Section 2.3.2) and inference (Section 2.3.3) in linear-chain CRFs.

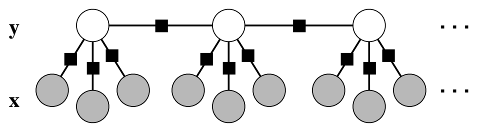


Figure 2.4. Graphical model of an HMM-like linear-chain CRF.

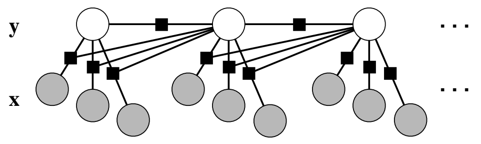


Figure 2.5. Graphical model of a linear-chain CRF in which the transition score depends on the current observation.

2.3.1 From HMMs to CRFs

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

First, we rewrite the HMM joint (2.34) in a form that is more amenable to generalization. This is

$$p(\mathbf{y},\mathbf{x}) = \frac{1}{Z}\exp\left\{\sum_t \sum_{i,j\in S} \lambda_{ij}\mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{y_{t-1}=j\}} + \sum_t \sum_{i\in S}\sum_{o\in O} \mu_{oi}\mathbf{1}_{\{y_t=i\}}\mathbf{1}_{\{x_t=o\}}\right\}, \tag{2.39}$$

where $\theta = \{\lambda_{ij}, \mu_{oi}\}$ are the parameters of the distribution, and can be any real numbers. Every HMM can be written in this form, as can be seen simply by setting $\lambda_{ij} = \log p(y' = i|y = j)$ and so on. Because we do not require the parameters to be log probabilities, we are no longer guaranteed that the distribution sums to 1, unless we explicitly enforce this by using a normalization constant Z. Despite this added flexibility, it can be shown that (2.39) describes exactly the class of HMMs in

(2.34); we have added flexibility to the parameterization, but we have not added any distributions to the family.

We can write (2.39) more compactly by introducing the concept of *feature functions*, just as we did for logistic regression in (2.33). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (2.39), there needs to be one feature $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$ for each transition (i, j) and one feature $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{x=o\}}$ for each state-observation pair (i, o). Then we can write an HMM as:

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}. \tag{2.40}$$

Again, equation (2.40) defines exactly the same family of distributions as (2.39), and therefore as the original HMM equation (2.34).

The last step is to write the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that results from the HMM (2.40). This is

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}'} p(\mathbf{y}', \mathbf{x})} = \frac{\exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{\mathbf{y}'} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_{t-1}, x_t)\right\}}. \tag{2.41}$$

This conditional distribution (2.41) is a linear-chain CRF, in particular 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 $f_k(y_t, y_{t-1}, \mathbf{x}_t)$ to be more general than indicator functions. This leads to the general definition of linear-chain CRFs, which we present now.

Definition 2.2. *Let* Y, X *be random vectors,* $\Lambda = \{\lambda_k\} \in \mathbb{R}^K$ *be a parameter vector, and* $\{f_k(y, y', \mathbf{x}_t)\}_{k=1}^K$ *be a set of real-valued feature functions. Then a* linear-chain conditional random field *is a distribution* $p(\mathbf{y}|\mathbf{x})$ *that takes the form*

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}, \tag{2.42}$$

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

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp \left\{ \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \tag{2.43}$$

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

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

Finally, note that the normalization constant $Z(\mathbf{x})$ sums over all possible state sequences, an exponentially large number of terms. Nevertheless, it can be computed efficiently by forward-backward, as we explain in Section 2.3.3.

2.3.2 Parameter Estimation

In this section we discuss how to estimate the parameters $\theta = \{\lambda_k\}$ of a linear-chain CRF. We are given iid training data $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$, where each $\mathbf{x}^{(i)} =$

$\{\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \dots \mathbf{x}_T^{(i)}\}$ is a sequence of inputs, and each $\mathbf{y}^{(i)} = \{y_1^{(i)}, y_2^{(i)}, \dots y_T^{(i)}\}$ is a sequence of the desired predictions. Thus, we have relaxed the iid assumption within each sequence, but we still assume that distinct sequences are independent. (In Section 2.4, we will see how to relax this assumption as well.)

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

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}). \tag{2.44}$$

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

$$\log p(\mathbf{y}, \mathbf{x}) = \log p(\mathbf{y}|\mathbf{x};\theta) + \log p(\mathbf{x};\theta'), \tag{2.45}$$

the two terms on the right-hand side are decoupled, that is, the value of θ' does not affect the optimization over θ. If we do not need to estimate $p(\mathbf{x})$, then we can simply drop the second term, which leaves (2.44).

After substituting in the CRF model (2.42) into the likelihood (2.44), we get the following expression:

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}), \tag{2.46}$$

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

$$\ell(\theta) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}. \tag{2.47}$$

The parameter σ^2 is a free parameter which determines how much to penalize large weights. The notation for the regularizer is intended to suggest that regularization can also be viewed as performing maximum a posteriori estimation of θ, if θ is assigned a Gaussian prior with mean 0 and covariance $\sigma^2 I$. Determining the best regularization parameter can require a computationally-intensive parameter sweep. Fortunately, often the accuracy of the final model is not sensitive to changes in σ^2, even when σ^2 is varied up to a factor of 10. An alternative choice of regularization is to use the L_1 norm instead of the Euclidean norm, which corresponds to an exponential prior on parameters [43]. This regularizer tends to encourage sparsity in the learned parameters. Many other choices of regularization are possible as well.

In general, the function $\ell(\theta)$ cannot be maximized in closed form, so numerical optimization is used. The partial derivatives of (2.47) are

$$\frac{\partial \ell}{\partial \lambda_k} = \sum_{i=1}^{N}\sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y'|\mathbf{x}^{(i)}) - \frac{\lambda_k}{\sigma^2}. \tag{2.48}$$

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

$$\tilde{p}(\mathbf{y}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}} \mathbf{1}_{\{\mathbf{x}=\mathbf{x}^{(i)}\}}. \tag{2.49}$$

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.

Now we discuss how to optimize $\ell(\theta)$. The function $\ell(\theta)$ is concave, which follows from the convexity of functions of the form $g(\mathbf{x}) = \log \sum_i \exp x_i$. Convexity is extremely helpful for parameter estimation, because it means that every local optimum is also a global optimum. Adding regularization ensures that ℓ is strictly concave, which implies that it has exactly one global optimum.

Perhaps the simplest approach to optimize ℓ is steepest ascent along the gradient (2.48), 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, even storing the full Hessian is not practical.

Instead, current techniques for optimizing (2.47) make approximate use of second-order information. Particularly successful have been quasi-Newton methods such as BFGS [6], which compute an approximation to the Hessian from only the first derivative of the objective function. A full $K \times K$ approximation to the Hessian still requires quadratic size, however, so a limited-memory version of BFGS is used, due to Byrd et al. [15]. As an alternative to limited-memory BFGS, conjugate gradient is another optimization technique that also makes approximate use of second-order information and has been used successfully with CRFs. 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. [58], as shown experimentally by several authors [65, 79, 112, 143]. Finally, trust region methods have recently been shown to perform well on multinomial logistic regression [62], and may work well for more general CRFs as well.

Recently, stochastic gradient methods, which make updates based on subsets of the training instances, have been shown to be highly effective [136], and may be an attractive alternative to second-order methods, which tend to evaluate the gradient over all the training instances before making an update. A promising new alternative to stochastic gradient methods is presented by Globerson et al. [42]. They make online-style updates in a dual of the original likelihood rather than in the primal representation, which both provides stronger convergence guarantees and added stability in practice.

Finally, it is important to remark on the computational cost of training. Both the partition function $Z(\mathbf{x})$ in the likelihood and the marginal distributions $p(y_t, y_{t-1}|\mathbf{x})$ in the gradient can be computed by forward-backward, which uses computational complexity $O(TM^2)$. However, each training instance will have a different partition function and marginals, so we need to run forward-backward for each training instance for each gradient computation, for a total training cost of $O(TM^2NG)$, where N is the number of training examples, and G the number of gradient computations required by the optimization procedure. (Unfortunately, the number of iterations G depends on the data set, and is difficult to predict in advance. For batch L-BFGS on linear-chain CRFs, it is usually but not always under 100.) For many data sets, this cost is reasonable, but if the number of states is large, or the number of training sequences is very large, then this can become expensive. For example, on a standard named-entity data set, with 11 labels and 200,000 words of training data, CRF training finishes in under two hours on current hardware. However, on a part-of-speech tagging data set, with 45 labels and one million words of training data, CRF training requires over a week.

2.3.3 Inference

There are two common inference problems for CRFs. First, during training, computing the gradient requires marginal distributions for each edge $p(y_t, y_{t-1}|\mathbf{x})$, and computing the likelihood requires $Z(\mathbf{x})$. Second, to label an unseen instance, we compute the most likely labeling $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. In linear-chain CRFs, both inference tasks can be performed efficiently and exactly by variants of the standard dynamic-programming algorithms for HMMs: forward-backward for computing marginal distributions and Viterbi algorithm for computing max-marginals. In this section, we briefly review the HMM algorithms, and extend them to linear-chain CRFs. These standard inference algorithms are described in more detail by Rabiner [98]. Both of these algorithms are special cases of the belief propagation algorithm described in Section 2.1.4, but I discuss this special case in detail both because it may help to make the earlier discussion more concrete, and because it is very 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:

$$\Psi_t(j, i, x) \overset{\text{def}}{=} p(y_t = j|y_{t-1} = i)p(x_t = x|y_t = j). \tag{2.50}$$

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:

$$p(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t) \tag{2.51}$$

$$= \sum_{y_{\mathrm{T}}} \sum_{y_{\mathrm{T}-1}} \Psi_{\mathrm{T}}(y_{\mathrm{T}}, y_{\mathrm{T}-1}, x_{\mathrm{T}}) \sum_{y_{\mathrm{T}-2}} \Psi_{\mathrm{T}-1}(y_{\mathrm{T}-1}, y_{\mathrm{T}-2}, x_{\mathrm{T}-1}) \sum_{y_{\mathrm{T}-3}} \cdots \tag{2.52}$$

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

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

$$\alpha_t(j) \overset{\text{def}}{=} p(\mathbf{x}_{\langle 1 \ldots t \rangle}, y_t = j) \tag{2.53}$$

$$= \sum_{\mathbf{y}_{\langle 1 \ldots t-1 \rangle}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{2.54}$$

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

$$\alpha_t(j) = \sum_{i \in S} \Psi_t(j, i, x_t) \alpha_{t-1}(i), \tag{2.55}$$

with initialization $\alpha_1(j) = \Psi_1(j, y_0, x_1)$. (Recall that y_0 is the fixed initial state of the HMM.) It is easy to see that $p(\mathbf{x}) = \sum_{y_{\mathrm{T}}} \alpha_{\mathrm{T}}(y_{\mathrm{T}})$ by repeatedly substituting the recursion (2.55) to obtain (2.52). A formal proof would use induction.

The backward recursion is exactly the same, except that in (2.52), we push in the summations in reverse order. This results in the definition

$$\beta_t(i) \overset{\text{def}}{=} p(\mathbf{x}_{\langle t+1 \ldots \mathrm{T} \rangle} | y_t = i) \tag{2.56}$$

$$= \sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{\mathrm{T}} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{2.57}$$

and the recursion

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}(j, i, x_{t+1}) \beta_{t+1}(j), \tag{2.58}$$

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

By combining results from the forward and backward recursions, we can compute the marginal distributions $p(y_{t-1}, y_t|\mathbf{x})$ needed for the gradient (2.48). This can be seen from either the probabilistic or the factorization perspectives. First, taking a probabilistic viewpoint we can write

$$\begin{aligned} p(y_{t-1}, y_t|\mathbf{x}) &= \frac{p(\mathbf{x}|y_{t-1}, y_t)p(y_t, y_{t-1})}{p(\mathbf{x})} & (2.59) \\ &= \frac{p(\mathbf{x}_{\langle 1 \ldots t-1 \rangle}, y_{t-1})p(y_t|y_{t-1})p(x_t|y_t)p(\mathbf{x}_{\langle t+1 \ldots \mathrm{T} \rangle}|y_t)}{p(\mathbf{x})} & (2.60) \\ &\propto \alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x_t)\beta_t(y_t), & (2.61) \end{aligned}$$

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. Second, from the factorization perspective, we can apply the distributive law to obtain we see that

$$p(y_{t-1}, y_t, \mathbf{x}) = \Psi_t(y_t, y_{t-1}, x_t) \left(\sum_{\mathbf{y}_{\langle 1 \ldots t-2 \rangle}} \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right) \left(\sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right), \tag{2.62}$$

which can be computed from the forward and backward recursions as

$$p(y_{t-1}, y_t, \mathbf{x}) = \alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x_t)\beta_t(y_t). \tag{2.63}$$

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

Finally, to compute the globally most probable assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, we observe that the trick in (2.52) still works if all the summations are replaced by maximization. This yields the Viterbi recursion:

$$\delta_t(j) = \max_{i \in S} \Psi_t(j, i, x_t)\delta_{t-1}(i) \tag{2.64}$$

Now that we have described the forward-backward and Viterbi algorithms for HMMs, the generalization to linear-chain CRFs is fairly straightforward. The forward-backward algorithm for linear-chain CRFs is identical to the HMM version, except that the transition weights $\Psi_t(j, i, x_t)$ are defined differently. We observe that the CRF model (2.42) can be rewritten as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}_t), \tag{2.65}$$

where we define

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp\left\{\sum_k \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}. \tag{2.66}$$

With that definition, the forward recursion (2.55), the backward recursion (2.58), and the Viterbi recursion (2.64) can be used unchanged for linear-chain CRFs. Instead of computing $p(\mathbf{x})$ as in an HMM, in a CRF the forward and backward recursions compute $Z(\mathbf{x})$.

A final inference task that is useful in some applications is to compute a marginal probability $p(y_t, y_{t+1}, \ldots y_{t+k}|\mathbf{x})$ over a possibly non-contiguous range of nodes. For example, this is useful for measuring the model's confidence in its predicted labeling over a segment of input. This marginal probability can be computed efficiently using constrained forward-backward, as described by [26].

2.4 CRFs in General

In this section, we define CRFs with general graphical structure, as they were introduced originally [58]. Although initial applications of CRFs used linear chains, there have been many later applications of CRFs with more general graphical structures. Such structures are especially useful for relational learning, because they allow relaxing the iid assumption among entities, or for more complicated entities, such as grids and trees. Also, although CRFs have typically been used for across-network classification, in which the training and testing data are assumed to be independent, we will see that CRFs can be used for within-network classification as well, in which we model probabilistic dependencies between the training and testing data.

The generalization from linear-chain CRFs to general CRFs is fairly straightforward. We simply move from using a linear-chain factor graph to a more general factor graph, and from forward-backward to more general (perhaps approximate) inference algorithms.

2.4.1 Model

First we present the general definition of a conditional random field.

Definition 2.3. *Let G be a factor graph over Y. Then $p(\mathbf{y}|\mathbf{x})$ is a conditional random field if for any fixed $\mathbf{x}$, the distribution $p(\mathbf{y}|\mathbf{x})$ factorizes according to G.*

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

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\Psi_A \in G} \exp\left\{ \sum_{k=1}^{K(A)} \lambda_{ak} f_{ak}(\mathbf{y}_a, \mathbf{x}_a) \right\}. \tag{2.67}$$

In addition, practical models rely extensively on parameter tying. For example, in the linear-chain case, often the same weights are used for the factors $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ at each time step. To denote this, we partition the factors of G into $\mathcal{C} = \{C_1, C_2, \dots C_P\}$, where each C_p is a *clique template* whose parameters are tied. This notion of clique template generalizes that in Taskar et al. [128], Sutton et al. [125], and Richardson and Domingos [101]. Each clique template C_p is a set of factors which has a corresponding set of sufficient statistics $\{f_{pk}(\mathbf{x}_p, \mathbf{y}_p)\}$ and parameters $\theta_p \in \mathbb{R}^{K(p)}$. Then the CRF can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p), \tag{2.68}$$

where each factor is parameterized as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = \exp \left\{ \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \right\}, \tag{2.69}$$

and the normalization function is

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p). \tag{2.70}$$

For example, in a linear-chain conditional random field, typically one clique template $C = \{\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)\}_{t=1}^{\mathrm{T}}$ is used for the entire network.

Several special cases of conditional random fields are of particular interest. First, *dynamic conditional random fields* [125] are sequence models which allow multiple labels at each time step, rather than single labels as in linear-chain CRFs. Second, *relational Markov networks* [128] are a type of general CRF in which the graphical structure and parameter tying are determined by an SQL-like syntax. Finally, *Markov logic networks* [101, 113] are a type of probabilistic logic in which there are parameters for each first-order rule in a knowledge base.

2.4.2 Applications of CRFs

CRFs have been applied to a variety of domains, including text processing, computer vision, and bioinformatics. In this section, we discuss several applications, highlighting the different graphical structures that occur in the literature.

One of the first large-scale applications of CRFs was by Sha and Pereira [112], 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 [70], feature induction for NER [68], identifying protein names in biology abstracts [111], segmenting addresses in Web pages [27], finding semantic roles in text [106], identifying the sources of opinions [19], Chinese word segmentation [92], Japanese morphological analysis [56], and many others.

In bioinformatics, CRFs have been applied to RNA structural alignment [109] and protein structure prediction [64]. Semi-Markov CRFs [108] add somewhat more flexibility in choosing features, which may be useful for certain tasks in information extraction and especially bioinformatics.

General CRFs have also been applied to several tasks in NLP. One promising application is to performing multiple labeling tasks simultaneously. For example, [125] 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 [40] present a CRF that learns dependencies between the categories, resulting in improved classification performance. Finally, the skip-chain CRF, which we present in Chapter 3, is a general CRF that represents long-distance dependencies in information extraction.

An interesting graphical CRF structure has been applied to the problem of proper-noun coreference, that is, of determining which mentions in a document, such as *Mr. President* and *he*, refer to the same underlying entity. McCallum and Wellner [71] learn a distance metric between mentions using a fully-connected conditional random field in which inference corresponds to graph partitioning. A similar model has been used to segment handwritten characters and diagrams [23, 96].

In some applications of CRFs, efficient dynamic programs exist even though the graphical model is difficult to specify. For example, [73] 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 [21, 102, 117, 135]. A potentially useful unifying framework for this type of model is provided by case-factor diagrams [67].

In computer vision, several authors have used grid-shaped CRFs [46, 57] for labeling and segmenting images. Also, for recognizing objects, Quattoni et al. [97] use a tree-shaped CRF in which latent variables are designed to recognize characteristic parts of an object.

2.4.3 Parameter Estimation

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 is given by

$$\ell(\theta) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x}). \tag{2.71}$$

It is worth noting that 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 λ_{pk} associated with a clique template C_p is

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) p(\mathbf{y}'_c | \mathbf{x}). \tag{2.72}$$

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.

Now, we discuss the latent-variable case, in which the model contains variables that are observed at neither training nor test time. It is more difficult to train CRFs with latent variables, because the latent variables need to be marginalized out to compute the likelihood. Because of this difficultly, the original work on CRFs focused on fully-observed training data, but recently there has been increasing interest in training latent-variable CRFs [73, 97].

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

$$p(\mathbf{y}, \mathbf{w} | \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{2.73}$$

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

$$\ell(\theta) = \log p(\mathbf{y}|\mathbf{x}) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x}). \tag{2.74}$$

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

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}) = \frac{1}{Z(\mathbf{y}, \mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p), \tag{2.75}$$

where the normalization factor is

$$Z(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{2.76}$$

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

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p) = \frac{Z(\mathbf{y}, \mathbf{x})}{Z(\mathbf{x})}. \tag{2.77}$$

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

We discuss two different ways to maximize ℓ: directly using the gradient, as in Quattoni et al. [97]; and using EM, as in McCallum et al. [73]. To maximize ℓ directly, we need to calculate its gradient. The simplest way to do this is to use the following fact. For any function $f(\lambda)$, we have

$$\frac{df}{d\lambda} = f(\lambda)\frac{d \log f}{d\lambda}, \tag{2.78}$$

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

$$\begin{aligned} \frac{\partial \ell}{\partial \lambda_{pk}} &= \frac{1}{\sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})} \sum_{\mathbf{w}} \frac{\partial}{\partial \lambda_{pk}} \big[p(\mathbf{y}, \mathbf{w}|\mathbf{x})\big] & (2.79) \\ &= \sum_{\mathbf{w}} p(\mathbf{w}|\mathbf{y}, \mathbf{x}) \frac{\partial}{\partial \lambda_{pk}} \big[\log p(\mathbf{y}, \mathbf{w}|\mathbf{x})\big]. & (2.80) \end{aligned}$$

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

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c} p(\mathbf{w}'_c|\mathbf{y}, \mathbf{x}) f_k(\mathbf{y}_c, \mathbf{x}_c, \mathbf{w}'_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c, \mathbf{y}'_c} p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c) f_k(\mathbf{y}'_c, \mathbf{x}_c, \mathbf{w}'_c). \tag{2.81}$$

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 (2.75). The second term contains a different marginal $p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c)$, which is the same marginal probability required in a fully-observed CRF. Once we have computed the gradient, ℓ can be maximized by standard techniques such as conjugate gradient. In our experience, conjugate gradient tolerates violations of convexity better than limited-memory BFGS, so it may be a better choice for latent-variable CRFs.

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

$$\theta^{(j+1)} = \arg\max_{\theta'} \sum_{\mathbf{w}'} q(\mathbf{w}') \log p(\mathbf{y}, \mathbf{w}'|\mathbf{x}; \theta'). \tag{2.82}$$

The direct maximization algorithm and the EM algorithm are strikingly similar. This can be seen by substituting the definition of q into (2.82) and taking derivatives. The gradient is almost identical to the direct gradient (2.81). 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.

2.4.4 Inference

In general CRFs, just as in the linear-chain case, gradient-based training requires computing marginal distributions $p(\mathbf{y}_c|\mathbf{x})$, and testing requires computing the most likely assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. This can be accomplished using any inference algorithm for graphical models. If the graph has small treewidth, then the junction tree algorithm can be used to exactly compute the marginals, but because both inference problems are NP-hard for general graphs, this is not always possible. In such cases, approximate inference must be used to compute the gradient.

When choosing an inference algorithm to use within CRF training, the important thing to understand is that it will be invoked repeatedly, once for each time that the gradient is computed. This can cause difficultly with sampling-based approaches, such as Markov chain Monte Carlo, which may take many iterations to converge for each parameter setting. However, contrastive divergence [47], a more computationally efficient method in which an MCMC sampler is run for only a few samples, has been successfully applied to CRFs in vision [46]. Because of their computational efficiency, variational approaches can be well-suited for CRF training. Several authors [125, 128] have used loopy belief propagation, described in Section 2.1.4.

2.4.5 Discussion

This section contains miscellaneous remarks about CRFs. First, it is easily seen that logistic regression model (2.33) is a conditional random field with a single output variable. Thus, CRFs can be viewed as an extension of logistic regression to arbitrary graphical structures.

Linear-chain CRFs were originally introduced as an improvement to the *maximum-entropy Markov model* (MEMM) [72], which is essentially a Markov model in which the transition distributions are given by a logistic regression model. MEMMs can exhibit the problems of label bias [58] and observation bias [53]. Both of these problems can be readily understood graphically: the directed model of an MEMM implies that for all time steps t, the observation x_t is marginally independent of the labels y_{t-1}, y_{t-2}, and so on—an independence assumption which is usually strongly violated in sequence modeling. Sometimes this assumption can be effectively avoided by including information from previous time steps as features, and this explains why MEMMs have had success in some NLP applications.

Although we have emphasized the view of a CRF as a model of the conditional distribution, one could view it as an objective function for parameter estimation of joint distributions. As such, it is one objective among many, including generative likelihood, pseudolikelihood [9], and the maximum-margin objective [3, 129]. Another related discriminative technique for structured models is the averaged perceptron, which has been especially popular in the natural language community [22], in large part because of its ease of implementation. To date, there has been little careful comparison of these, especially CRFs and max-margin approaches, across different structures and domains.

Given this view, it is natural to imagine training directed models by conditional likelihood, and in fact this is commonly done in the speech community, where it is called maximum mutual information training. However, it is no easier to maximize the conditional likelihood in a directed model than an undirected model, because in a directed model the conditional likelihood requires computing $\log p(\mathbf{x})$, which plays the same role as $Z(\mathbf{x})$ in the CRF likelihood. In fact, training is more complex in a directed model, because the model parameters are constrained to be probabilities—constraints which can make the optimization problem more difficult. This is in stark contrast to the joint likelihood, which is much easier to compute for directed models than undirected models (although recently several computationally efficient parameter estimation techniques have been proposed for undirected factor graphs, such as Abbeel et al. [1] and Wainwright et al. [142]).

2.4.6 Implementation Concerns

There are a few implementation techniques that can help both training time and accuracy of CRFs, but are not always fully discussed in the literature. Although these apply especially to language applications, they are also useful more generally.

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

$$f_{pk}(\mathbf{y}_c, \mathbf{x}_c) = \mathbf{1}_{\{\mathbf{y}_c = \tilde{\mathbf{y}}_c\}} q_{pk}(\mathbf{x}_c). \tag{2.83}$$

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

This representation can lead to a large number of features, which can have significant memory and time requirements. For example, to match state-of-the-art results on a standard natural language task, Sha and Pereira [112] use 3.8 million features. Not all of these features are ever nonzero in the training data. In particular, some observation functions q_{pk} are nonzero only for certain output configurations. This point can be confusing: One might think that such features can have no effect on the likelihood, but actually they do affect $Z(\mathbf{x})$, so putting a negative weight on them can improve the likelihood by making wrong answers less likely. In order to save memory, however, sometimes these *unsupported features*, that is, those which never occur in the training data, are removed from the model. In practice, however, including unsupported features typically results in better accuracy.

In order to get the benefits of unsupported features with less memory, we have had success with an ad hoc technique for selecting a small set of unsupported features. The idea is to add unsupported features only for likely paths, as follows: first train a CRF without any unsupported features, stopping after a few iterations; then add unsupported features $f_{pk}(\mathbf{y}_c, \mathbf{x}_c)$ for cases where $\mathbf{x}_c$ occurs in the training data for some instance $\mathbf{x}^{(i)}$, and $p(\mathbf{y}_c|\mathbf{x}^{(i)}) > \epsilon$. McCallum [68] presents a more principled method of feature selection for CRFs.

Second, if the observations are categorical rather than ordinal, that is, if they are discrete but have no intrinsic order, it is important to convert them to binary features. For example, it makes sense to learn a linear weight on $f_k(y, x_t)$ when f_k is 1 if x_t is the word *dog* and 0 otherwise, but not when f_k is the integer index of word x_t in the text's vocabulary. Thus, in text applications, CRF features are typically binary; in other application areas, such as vision and speech, they are more commonly real-valued.

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

Fourth, sometimes it is preferable to use L_1 regularization instead of L_2, particularly if it is desired that the trained weights be sparse; it also has certain theoretical advantages [87]. The L_1 regularizer is not differentiable at 0, which complicates numerical parameter estimation somewhat [4, 43].

Finally, often the probabilities involved in forward-backward and belief propagation become too small to be represented within numerical precision. There are two standard approaches to this common problem. One approach is to normalize each of the vectors α_t and β_t to sum to 1, thereby magnifying small values. This scaling does not affect our ability to compute $Z(\mathbf{x})$. The details of how to do this are given by Rabiner [98].

A second approach is to perform computations in the logarithmic domain, e.g., the forward recursion becomes

$$\log \alpha_t(j) = \bigoplus_{i \in S} \big(\log \Psi_t(j, i, x_t) + \log \alpha_{t-1}(i) \big), \tag{2.84}$$

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

$$a \oplus b = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b}), \tag{2.85}$$

which can be much more numerically stable, particularly if we pick the version of the identity with the smaller exponent. CRF implementations often use the log-space approach because it makes computing $Z(\mathbf{x})$ more convenient, but in some applications, the computational expense of taking logarithms is an issue, making normalization preferable.

Notes on Terminology

Different parts of the theory of graphical models have been developed independently in many different areas, so many of the concepts in this chapter have different names in different areas. For example, undirected models are commonly also referred to *Markov random fields*, *Markov networks*, and *Gibbs distributions*. As mentioned, I 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 I 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.

Acknowledgments

I thank Francine Chen and Benson Limketkai for useful comments on an earlier version of this chapter.

CHAPTER 3
MODELS

Although many sequence tagging and information extraction tasks have proven amenable to linear-chain CRFs, higher-order dependencies do exist in these problems. In this chapter, I introduce two families of loopy conditional random fields that model limited forms of long-distance structure, and achieve better accuracy as a result. Not only are these models of practical interest, but also they will prove useful for testing the approximate training algorithms that I present in the remainder of the thesis.

Both of the model families that I discuss are constructed as augmentations of linear-chain CRFs. First, I introduce the *dynamic CRF* (Section 3.1), which augments a linear-chain CRF with factorized state. That is, the state at each sequence position is represented as a vector rather than as a single variable, which allows the transition distribution to be represented more efficiently. Second, I introduce the *skip-chain CRF* (Section 3.2), which captures the idea that similar tokens should receive similar labels, even if they are far apart in the sequence. This smoothing effect can be achieved by adding long-distance factors that depend on the similar token pairs. For both DCRFs and skip-chain CRFs, I explore the use of approximate inference algorithms, particularly loopy belief propagation, during training. Approximate inference can greatly improve training speed while maintaining accuracy; however, if inference is too inaccurate, then the quality of the solution can degrade markedly.

3.1 Dynamic CRFs

Many sequence labeling problems have a natural notion of factorized state. In a factorized state representation, rather than representing the state as a single random variable y_t for each sequence position t, the state is represented as a vector $\mathbf{y}_t = \{y_{t1}, y_{t2}, \ldots, y_{tm}\}$, allowing the model to be more compact because it can represent the graphical structure among components of $\mathbf{y}$. In generative sequence models, this factorization is typically represented by a dynamic Bayesian network (DBN) [29, 84], which is a directed graphical model whose structure is repeated across a sequence. DBNs have been used for applications as diverse as robot navigation [133], audio-visual speech recognition [86], activity recognition [13], information extraction [93, 114], and automatic speech recognition [10]. DBNs are typically trained to maximize the joint probability distribution $p(\mathbf{y}, \mathbf{x})$ of a set of observation sequences $\mathbf{x}$ and labels $\mathbf{y}$. As discussed in Section 2.2.3, however, when the task does not require the ability to generate $\mathbf{x}$, such as in segmentation and labeling, modeling the joint distribution is a waste of modeling effort.

A solution to this problem is to model instead the conditional probability distribution $p(\mathbf{y}|\mathbf{x})$, as in a conditional random field. For this reason, we introduce *dynamic CRFs (DCRFs)*, which are a generalization of linear-chain CRFs that repeat structure and parameters over a sequence of state vectors. This allows us to both represent distributed hidden state and complex interaction among labels, as in DBNs, and to use rich, overlapping feature sets, as in conditional models. For example, the factorial structure in Figure 3.1(b) includes links between cotemporal labels, explicitly modeling limited probabilistic dependencies between two different label sequences. Other types of DCRFs can model higher-order Markov dependence between labels (Figure 3.2), or incorporate a fixed-size memory. For example, a DCRF for part-of-speech tagging could include for each word a hidden state that is true if any previous word has been tagged as a verb.

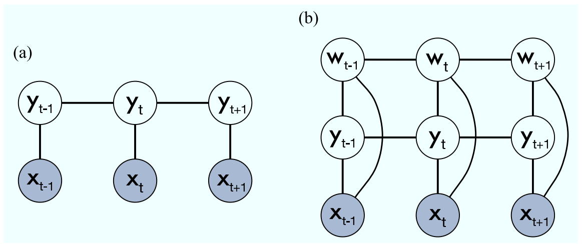


Figure 3.1. Graphical representation of (a) linear-chain CRF, and (b) factorial CRF. Although the hidden nodes can depend on observations at any time step, for clarity we have shown links only to observations at the same time step.

Any DCRF with multiple state variables can be collapsed into a linear-chain CRF whose state space is the cross-product of the outcomes of the original state variables. However, such a linear-chain CRF needs exponentially many parameters in the number of variables. Like DBNs, DCRFs represent the joint distribution with fewer parameters by exploiting conditional independence relations.

In natural-language processing, DCRFs are especially attractive because they are a probabilistic generalization of cascaded, weighted finite-state transducers [82]. In general, many sequence-processing problems are traditionally solved by chaining errorful subtasks, such as chains of finite state transducers. In such an approach, however, errors early in processing nearly always cascade through the chain, causing errors in the final output. This problem can be solved by jointly representing the subtasks in a single graphical model, both explicitly representing their dependence, and preserving uncertainty between them. DCRFs can represent dependence between subtasks solved using finite-state transducers, such as phonological and morphological analysis, POS tagging, shallow parsing, and information extraction.

More specifically, in information and data mining, McCallum and Jensen [69] argue that the same kind of probabilistic unification can potentially be useful, because in many cases, we wish to mine a database that has been extracted from raw text. A unified probabilistic model for extraction and mining can allow data mining to take

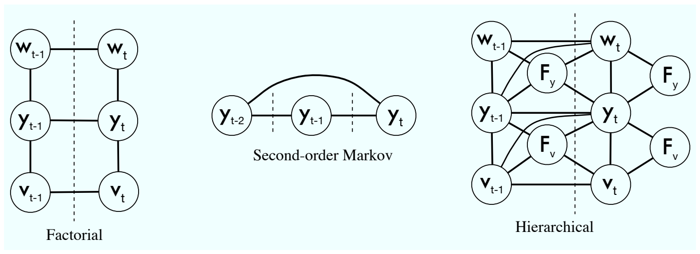


Figure 3.2. Examples of DCRFs. The dashed lines indicate the boundary between time steps. The input variables $\mathbf{x}$ are not shown.

into account the uncertainty in the extraction, and allow extraction to benefit from emerging pattern produced by data mining. The applications here, in which DCRFs are used to jointly perform multiple sequence labeling tasks, can be viewed as an initial step toward that goal.

In this chapter, we evaluate DCRFs on several natural-language processing tasks. First, a factorial CRF that learns to jointly predict parts of speech and segment noun phrases performs better than cascaded models that perform the two tasks in sequence. Also, we compare several schedules for belief propagation, showing that although exact inference is feasible, on this task approximate inference has lower total training time with no loss in testing accuracy.

In addition to conditional maximum likelihood training, we present an alternative training method for DCRFs, *cascaded training.* Cascaded training is intended for situations in which a single fully-labeled data set is not available, and instead the outputs are partitioned into sets $(\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_\ell)$, and we have one data set D_0 labeled for $\mathbf{y}_0$, another data set D_1 labeled for $\mathbf{y}_1$, and so on. For example, this can be the case in *transfer learning*, in which we wish to use previous learning problems (that is, $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{\ell-1}$) to improve performance on a new task $\mathbf{y}_\ell$. To handle the fact that a single fully-labeled training set is unavailable, our procedure works in a cascaded fashion, in which first we train a CRF p_0 to predict $\mathbf{y}_0$ on D_0, then we annotate D_1 with the most likely prediction from p_0, then we train a CRF p_1 on

$p(\mathbf{y}_1|\mathbf{y}_0, \mathbf{x})$, and so on. Compared to other work in transfer learning, an interesting aspect of this approach is that the model includes no shared latent structure between subtasks; rather, the probabilistic dependence between tasks is modeled directly. On a benchmark information extraction task, we show that a DCRF trained in a cascaded fashion performs better than a linear-chain CRF on the original task.

In the rest of this section, we first define DCRFs (Section 3.1.1), explaining methods for approximate inference and parameter estimation, including parameter estimation using BP (Section 3.1.4) and cascaded parameter estimation (Section 3.1.5). In Section 3.1.6, we present the experimental results, including evaluation of FCRFs on noun-phrase chunking (Section 3.1.6.1), comparison of BP schedules in FCRFs (Section 3.1.6.2), and cascaded training of DCRFs for transfer learning (Section 3.1.6.3).

3.1.1 Model Representation

A dynamic CRF (DCRF) is a conditional distribution that factorizes according to an undirected graphical model whose structure and parameters are repeated over a sequence. As with a DBN, a DCRF can be specified by a template that gives the graphical structure, features, and weights for two time slices, which can then be unrolled given an input $\mathbf{x}$. The same set of features and weights is used at each sequence position, so that the parameters are tied across the network. Several example templates are given in Figure 3.2.

Now we give a formal description of the unrolling process. Let $\mathbf{y} = \{\mathbf{y}_1 \dots \mathbf{y}_T\}$ be a sequence of random vectors $\mathbf{y}_i = (y_{i1} \dots y_{im})$, where $\mathbf{y}_i$ is the state vector at time i, and y_{ij} is the value of variable j at time i. To give the likelihood equation for arbitrary DCRFs, we require a way to describe a clique in the unrolled graph independent of its position in the sequence. For this purpose we introduce the concept of a *clique index*. Given a time t, we can denote any variable $\mathbf{y}_{ij}$ in $\mathbf{y}$ by two integers: its index j in the state vector $\mathbf{y}_i$, and its time offset $\Delta t = i - t$. We will call a set

$c = \{(\Delta t, j)\}$ of such pairs a clique index, which denotes a set of variables $\mathbf{y}_{t,c}$ by $\mathbf{y}_{t,c} \equiv \{y_{t+\Delta t,j} \,|\, (\Delta t, j) \in c\}$. That is, $\mathbf{y}_{t,c}$ is the set of variables in the unrolled version of clique index c at time t.

Now we can formally define DCRFs:

Definition 3.1. *Let C be a set of clique indices, $F = \{f_k(\mathbf{y}_{t,c}, \mathbf{x}, t)\}$ be a set of feature functions and $\Lambda = \{\lambda_k\}$ be a set of real-valued weights. Then the distribution p is a* dynamic conditional random field *if and only if*

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_t \prod_{c \in C} \exp\left(\sum_k \lambda_k f_k(\mathbf{y}_{t,c}, \mathbf{x}, t) \right) \tag{3.1}$$

where $Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_t \prod_{c \in C} \exp\left(\sum_k \lambda_k f_k(\mathbf{y}_{t,c}, \mathbf{x}, t)\right)$ is the partition function.

Although we define a DCRF has having the same set of features for all the cliques, in practice we choose feature functions f_k so that they are non-zero except on cliques with some index c_k. Thus, we will sometimes think of each clique index has having its own set of features and weights, and speak of f_k and λ_k as having an associated clique index c_k.

DCRFs generalize not only linear-chain CRFs, but more complicated structures as well. For example, in this chapter, we use a *factorial CRF (FCRF)*, which has linear chains of labels, with connections between cotemporal labels. We name these after factorial HMMs [39]. Figure 3.1(b) shows an unrolled factorial CRF. Consider an FCRF with L chains, where $Y_{\ell,t}$ is the variable in chain ℓ at time t. The clique indices for this DCRF are of the form $\{(0, \ell), (1, \ell)\}$ for each of the within-chain edges and $\{(0, \ell), (0, \ell + 1)\}$ for each of the between-chain edges. The FCRF G defines a distribution over hidden states as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \left(\prod_{t=1}^{T-1} \prod_{\ell=1}^{L} \Phi_\ell(y_{\ell,t}, y_{\ell,t+1}, \mathbf{x}, t) \right) \left(\prod_{t=1}^{T} \prod_{\ell=1}^{L-1} \Psi_\ell(y_{\ell,t}, y_{\ell+1,t}, \mathbf{x}, t) \right), \tag{3.2}$$

where $\{\Phi_\ell\}$ are the potentials over the within-chain edges, $\{\Psi_\ell\}$ are the potentials over the between-chain edges, and $Z(\mathbf{x})$ is the partition function. The potentials factorize according to the features $\{f_k\}$ and weights $\{\lambda_k\}$ of G as:

$$\Phi_\ell(y_{\ell,t}, y_{\ell,t+1}, \mathbf{x}, t) = \exp\left\{\sum_k \lambda_k f_k(y_{\ell,t}, y_{\ell,t+1}, \mathbf{x}, t)\right\}$$
$$\Psi_\ell(y_{\ell,t}, y_{\ell+1,t}, \mathbf{x}, t) = \exp\left\{\sum_k \lambda_k f_k(y_{\ell,t}, y_{\ell+1,t}, \mathbf{x}, t)\right\}$$

More complicated structures are also possible, such as second-order CRFs, and hierarchical CRFs, which are moralized versions of the hierarchical HMMs of Fine et al. [31].[1] As in DBNs, this factorized structure can use many fewer parameters than the cross-product state space: even the two-level FCRF we discuss below uses less than an eighth of the parameters of the corresponding cross-product CRF.

3.1.2 Inference in DCRFs

Inference in a DCRF can be done using any inference algorithm for undirected models. For an unlabeled sequence $\mathbf{x}$, we typically wish to solve two inference problems: (a) computing the marginals $p(\mathbf{y}_{t,c}|\mathbf{x})$ over all cliques $\mathbf{y}_{t,c}$, and (b) computing the Viterbi decoding $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. The Viterbi decoding can be used to label a new sequence, and marginal computation is used for parameter estimation.

If the number of states is not large, the simplest approach is to form a linear chain whose output space is the cross-product of the original DCRF outputs, and then perform forward-backward. In other words, a DCRF can always be viewed as a linear-chain CRF whose feature functions take a special form, analogous to the relationship between generative DBNs and HMMs. The cross-product space is often very large, however, in which case this approach is infeasible. Alternatively, one

[1]Hierarchical HMMs were shown to be DBNs by Murphy and Paskin [83].

can perform exact inference by applying the junction tree algorithm to the unrolled DCRF, or by using the special-purpose inference algorithms that have been designed for DBNs [84], which can avoid storing the full unrolled graph.

In complex DCRFs, though, exact inference can still be expensive, so that approximate methods are necessary. Furthermore, because marginal computation is needed during training, inference must be efficient so that we can use large training sets even if there are many labels. The largest experiment reported here required computing pairwise marginals in 866,792 different graphical models: one for each training example in each iteration of a convex optimization algorithm.

We focus on approximate inference using loopy belief propagation, which was described in Section 2.1.4. In the experiments here, we pay special attention to the order in which messages are propagated. At each iteration of belief propagation, messages can be sent in any order, and choosing a good schedule can affect how quickly the algorithm converges. We describe two schedules for belief propagation: tree-based and random. The tree-based schedule, also known as tree reparameterization (TRP) [137, 139], propagates messages along a set of cross-cutting spanning trees of the original graph. At each iteration of TRP, a spanning tree $\mathcal{T}^{(i)} \in \Upsilon$ is selected, and messages are sent in both directions along every edge in $\mathcal{T}^{(i)}$, which amounts to exact inference on $\mathcal{T}^{(i)}$. Many possible sets of spanning trees can be imagined, but here we select trees randomly, except that edges that have never been used in any previous iteration are selected first.

The random schedule simply sends messages across all edges in random order. To improve convergence, we arbitrarily order each edge $e_i = (s_i, t_i)$ and send all messages $m_{s_i}(t_i)$ before any messages $m_{t_i}(s_i)$. Note that for a graph with V nodes and E edges, TRP sends $O(V)$ messages per BP iteration, while the random schedule sends $O(E)$ messages.

An alternative schedule is a synchronous schedule, in which conceptually all messages are sent at the same time. In the tree-based and random schedules, once a message is updated, its new values are immediately available for other messages. In the synchronous schedule, on the other hand, when computing a message $m_u^{(j)}(x_v)$ at iteration j of BP, the previous message values $m_t^{(j-1)}(x_u)$ are always used, even if an updated value $m_t^{(j)}(x_u)$ has been computed. We do not report results from the synchronous schedule because preliminary experiments indicated that it requires many more iterations to converge than the other schedules.

3.1.3 Parameter Estimation in DCRFs

Parameter estimation of DCRFs by conditional maximum likelihood follows the general method explained in Section 2.4.3. Written in the notation of this chapter, the likelihood is

$$\mathcal{L}(\Lambda) = \sum_i \log p_\Lambda(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}). \tag{3.3}$$

The derivative of this with respect to a parameter λ_k associated with clique index c is

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \lambda_k} &= \sum_i \sum_t f_k(\mathbf{y}_{t,c}^{(i)}, \mathbf{x}^{(i)}, t) \\ &\quad - \sum_i \sum_t \sum_{\mathbf{y}_{t,c}} p_\Lambda(\mathbf{y}_{t,c} \mid \mathbf{x}^{(i)}) f_k(\mathbf{y}_{t,c}, \mathbf{x}^{(i)}, t).\end{aligned} \tag{3.4}$$

where $\mathbf{y}_{t,c}^{(i)}$ is the assignment to $\mathbf{y}_{t,c}$ in $\mathbf{y}^{(i)}$, and $\mathbf{y}_{t,c}$ ranges over assignments to the clique c. Observe that it is the factor $p_\Lambda(\mathbf{y}_{t,c} \mid \mathbf{x}^{(i)})$ that requires us to compute marginal probabilities in the unrolled DCRF. As before, to reduce overfitting, we define a spherical Gaussian prior $p(\Lambda)$ over parameters, mean $\mu = 0$ and covariance matrix $\Sigma = \sigma^2 I$, so that the gradient becomes

$$\frac{\partial p(\Lambda|\mathcal{D})}{\partial \lambda_k} = \frac{\partial \mathcal{L}}{\partial \lambda_k} - \frac{\lambda_k}{\sigma^2}.$$

In the experiments here, we optimize the gradient using batch limited-memory BFGS.

3.1.4 Approximate Parameter Estimation Using BP

Several additional issues arise when loopy BP is used during training. First, to simplify notation in this section, we will write a DCRF as $p(\mathbf{y}|\mathbf{x}) = Z(\mathbf{x})^{-1} \prod_t \prod_c \psi_{t,c}(\mathbf{y}_{t,c})$, where each factor in the unrolled DCRF is defined as

$$\psi_{t,c}(\mathbf{y}_{t,c}) = \exp\{\lambda_k f_k(\mathbf{y}_{t,c}, \mathbf{x}, t)\}. \tag{3.5}$$

The basic procedure is to optimize the likelihood (3.3) as described in the last section, but instead of running an exact inference algorithm on each training example to obtain marginal distributions $p_\Lambda(\mathbf{y}_{t,c} \mid \mathbf{x}^{(i)})$, we run BP on each training instance to obtain approximate beliefs $b_{t,c}(\mathbf{y}_{t,c})$ for each clique $\mathbf{y}_{t,c}$ and approximate node belief $b_s(y_s)$ for each output variable s.

Now, although BP provides approximate marginal distributions that allow calculating the gradient, there is still the issue of how to calculate an approximate likelihood. In particular, we need an approximate objective function whose gradient is equal to the approximate gradient we have just described. We use the approximate likelihood

$$\hat{\ell}(\Lambda; \{b\}) = \sum_i \log \left[\frac{\prod_t \prod_c b_{t,c}(\mathbf{y}^{(i)}_{t,c})}{\prod_s b_s(y^{(i)}_s)^{d_s - 1}} \right], \tag{3.6}$$

where s ranges over output variables (that is, components of $\mathbf{y}$), and d_s is the degree of s (that is, the number of factors $\psi_{c,t}$ that depend on the variable s). In other words, we approximate the joint likelihood by the product over each clique's approximate belief, dividing by the node beliefs to avoid overcounting. In the remainder of this section, we justify this choice.

BP can be viewed as attempting to solve an optimization problem over possible choices of marginal distributions, for a particular cost function called the *Bethe free energy*. More technically, it has been shown that fixed points of BP are stationary points of the Bethe free energy [for more details, see 150], when minimized over locally-consistent marginal distributions. The Bethe energy is an approximation to another cost function, which Yedidia et al. call the Helmholtz free energy. Since the minimum Helmholtz energy is is exactly $-\log Z(\mathbf{x})$, we approximate $-\log Z(\mathbf{x})$ by the minimizing value of the Bethe energy, that is:

$$\ell_{\text{BETHE}}(\Lambda) = \sum_i \sum_t \sum_c \log \psi_{t,c}(\mathbf{y}_{t,c}) + \sum_i \min_{\{b\}} \mathcal{F}_{\text{BETHE}}(b), \tag{3.7}$$

where $\mathcal{F}_{\text{BETHE}}$ is the Bethe free energy, which is defined as

$$\mathcal{F}_{\text{BETHE}}(b) = \sum_t \sum_c \sum_{\mathbf{y}_{t,c}} b_{t,c}(\mathbf{y}_{t,c}) \log \frac{b_{t,c}(\mathbf{y}_{t,c})}{\psi_{t,c}(\mathbf{y}_{t,c})} - \sum_s (d_s - 1) \sum_{x_s} b_s(y_s) \log b_s(y_s). \tag{3.8}$$

So approximate training with BP can be viewed as solving a saddle point problem of maximizing ℓ_{BETHE} with respect to the model parameters and minimizing with respect to the beliefs $b_{t,c}(\mathbf{x}_{t,c})$. Approximate training using BP is just coordinate ascent: BP optimizes ℓ_{BETHE} with respect to b for fixed Λ; and a step along the gradient (3.4) optimizes ℓ_{BETHE} with respect to Λ for fixed b. Taking the partial derivative of (3.7) with respect to a weight λ_k, we obtain the gradient (3.4) with marginal distributions replaced by beliefs, as desired.

To justify the approximate likelihood (3.6), we note that the Bethe free energy can be written a dual form, in which the variables are interpreted as log messages rather than beliefs. Details of this are presented by Minka [78]. Substituting the Bethe dual problem into (3.7) and simplifying yields (3.6).

3.1.5 Cascaded Parameter Estimation

Joint maximum likelihood training assumes that we have access to data in which we have observed all of the variables. Sometimes this is not the case. One example is *transfer learning*, which is the general problem of using previous learning problems that a system has seen to aid its learning of new, related tasks. Usually in transfer learning, we have one data set labeled with the old task variables and one with the new task variables, but no data that is jointly labeled. In this section, we describe a cascaded parameter estimation procedure that can be applied to situations without fully-labeled data.

For a factorial CRF with N levels, the basic idea is to train each level separately as if it were a linear-chain CRF, using the single-best prediction of the previous level as a feature. At the end, each set of individually-trained weights define a pair of factors, which are simply multiplied together to form the full FCRF. The cascaded procedure for an N-level FCRF is described formally in Algorithm 3.1. In this description, the within-level clique template for level ℓ has features $f^{\text{W}}_{\ell,k}(y^\ell_t, y^\ell_{t+1}, \mathbf{x}, t)$ and weights Λ^{W}_ℓ; and the between-level clique template has features $f^{\text{P}}_{\ell,k}(y^\ell_t, y^{\ell-1}_t, \mathbf{x}, t)$ and weights Λ^{P}_ℓ.

Algorithm 3.1 Cascaded training for Factorial CRFs

```
1: Train a linear-chain CRF on $\log p(\mathbf{y}_0|\mathbf{x})$, yielding weights $\Lambda^{\text{W}}_0$.
2: for all levels $\ell$ do
3:     Compute Viterbi labeling $\mathbf{y}^*_{\ell-1} = \arg\max_{\mathbf{y}_{\ell-1}} p(\mathbf{y}_{\ell-1}|\mathbf{y}^*_{\ell-2}, \mathbf{x})$ for each training
       instance $i$.
4:     Train a linear-chain CRF to maximize $\log p(\mathbf{y}_\ell|\mathbf{y}^*_{\ell-1}, \mathbf{x})$, yielding weights $\Lambda^{\text{W}}_\ell$
       and $\Lambda^{\text{P}}_\ell$.
5: end for
6: return factorial CRF defined as
```

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_{\ell=0}^{N} \prod_{t=1}^{T} \Psi^{\text{W}}(y^\ell_t, y^\ell_{t+1}, \mathbf{x}, t) \Psi^{\text{P}}(y^\ell_t, y^{\ell-1}_t, \mathbf{x}, t) \tag{3.9}$$

where

$$\Psi^{\text{W}}(y^\ell_t, y^\ell_{t+1}, \mathbf{x}, t) = \exp\{\sum_k \lambda^{\text{W}}_{k,\ell} f^{\text{W}}_{\ell,k}(y^\ell_t, y^\ell_{t+1}, \mathbf{x}, t)\} \tag{3.10}$$

$$\Psi^{\text{P}}(y^\ell_t, y^{\ell-1}_t, \mathbf{x}, t) = \exp\{\sum_k \lambda^{\text{P}}_{k,\ell} f^{\text{P}}_{\ell,k}(y^\ell_t, y^{\ell-1}_t, \mathbf{x}, t)\} \tag{3.11}$$

For simplicity, we have presented cascaded training for factorial CRFs, but it can be generalized to other DCRF structures, as long as the DCRF templates can be

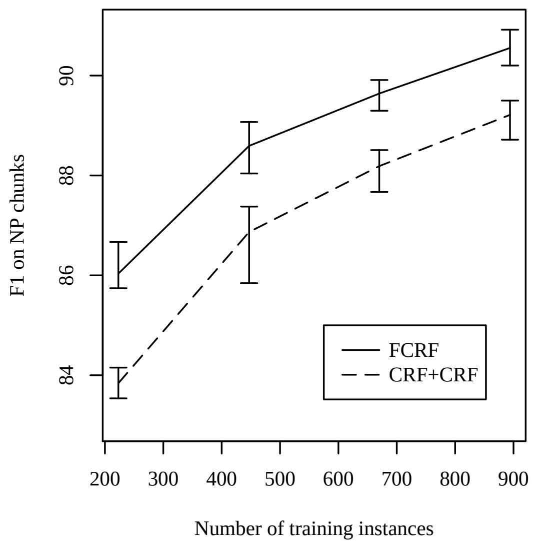


Figure 3.3. Performance of FCRFs and cascaded approaches on noun-phrase chunking, averaged over five repetitions. The error bars on FCRF and CRF+CRF indicate the range of the repetitions.

partitioned in a way that respects the available labels. In Section 3.1.6.3, we evaluate cascaded training on a transfer learning problem.

3.1.6 Experiments

We present experiments comparing factorial CRFs to other approaches on noun-phrase chunking [107]. Also, we compare different schedules of loopy belief propagation in factorial CRFs.

3.1.6.1 FCRFs for Noun-Phrase Chunking

Automatically finding the base noun phrases in a sentence can be viewed as a sequence labeling task by labeling each word as either Begin-Phrase, Inside-Phrase, or Other [99]. The task is typically performed by an initial pass of part-of-speech tagging, but then it can be difficult to recover from errors by the tagger. In this section, we address this problem by performing part-of-speech tagging and noun-phrase segmentation jointly in a single factorial CRF.

	Size	CRF+CRF	Brill+CRF	FCRF
POS accuracy	223	86.23	N/A	**93.12**
	447	90.44		**95.43**
	670	92.33		**96.34**
	894	93.56		**96.85**
	2234	96.18		**97.87**
	8936	98.28		**98.92**
Joint accuracy	223	81.92	N/A	**89.19**
	447	86.58		**91.85**
	670	88.68		**92.86**
	894	90.06		**93.60**
	2234	93.00		**94.90**
	8936	95.56		**96.48**
NP F1	223	83.84	86.02	**86.03**
	447	86.87	88.56	**88.59**
	670	88.19	**89.65**	89.64
	894	89.21	90.31	**90.55**
	2234	91.07	91.90	**92.02**
	8936	93.10	93.33	**93.87**

Table 3.1. Performance comparison of cascaded models and FCRFs on simultaneous noun-phrase chunking and POS tagging. The column SIZE lists the number of sentences used in training. The row CRF+CRF lists results from cascaded CRFs, and Brill+CRF lists results from a linear-chain CRF given POS tags from the Brill tagger. The FCRF always outperforms CRF+CRF, and given sufficient training data outperforms Brill+CRF. With small amounts of training data, Brill+CRF and the FCRF perform comparably, but the Brill tagger was trained on over 40,000 sentences, including some in the CoNLL 2000 test set.

Our data comes from the CoNLL 2000 shared task [107], and consists of sentences from the Wall Street Journal annotated by the Penn Treebank project [66]. We consider each sentence to be a training instance, with single words as tokens. The data are divided into a standard training set of 8936 sentences and a test set of 2012 sentences. There are 45 different POS labels, and the three NP labels.

We compare a factorial CRF to two cascaded approaches, which we call *CRF+CRF* and *Brill+CRF*. CRF+CRF uses one linear-chain CRF to predict POS labels, and another linear-chain CRF to predict NP labels, using as a feature the Viterbi POS labeling from the first CRF. Brill+CRF predicts NP labels using the POS labels pro-

$w_{t-\delta} = w$
w_t matches `[A-Z][a-z]+`
w_t matches `[A-Z]`
w_t matches `[A-Z]+`
w_t matches `[A-Z]+[a-z]+[A-Z]+[a-z]`
w_t matches `.*[0-9].*`
w_t appears in list of first names, last names, company names, days, months, or geographic entities
w_t is contained in a lexicon of words with POS T (from Brill tagger)
$T_t = T$
$q_k(\mathbf{x}, t + \delta)$ for all k and $\delta \in [-3, 3]$

Table 3.2. Input features $q_k(\mathbf{x}, t)$ for the CoNLL data. In the above w_t is the word at position t, T_t is the POS tag at position t, w ranges over all words in the training data, and T ranges over all part-of-speech tags.

vided from the Brill tagger, which we expect to be more accurate than those from our CRF, because the Brill tagger was trained on over four times more data, including sentences from the CoNLL 2000 test set.

The factorial CRF uses the graph structure in Figure 3.1(b), with one chain modeling the part-of-speech process and the other modeling the noun-phrase process. We use L-BFGS to optimize the posterior $p(\Lambda|\mathcal{D})$, and TRP to compute the marginal probabilities required by $\partial\mathcal{L}/\partial\lambda_k$. Based on past experience with linear-chain CRFs, we use the prior variance $\sigma^2 = 10$ for all models.

We factorize our features as $f_k(y_{t,c}, x, t) = p_k(y_{t,c})q_k(\mathbf{x}, t)$ where $p_k(y_{t,c})$ is a binary function on the assignment, and $q_k(\mathbf{x}, t)$ is a function solely of the input string. Table 3.2 shows the features we use. All three approaches use the same features, with the obvious exception that the FCRF and the first stage of CRF+CRF do not use the POS features $T_t = T$.

Performance on noun-phrase chunking is summarized in Table 3.1. As usual, we measure performance on chunking by *precision*, the percentage of returned phrases that are correct; *recall*, the percentage of correct phrases that were returned; and their harmonic mean F_1. In addition, we also report accuracy on POS labels,[2] and joint accuracy on (POS, NP) pairs. Joint accuracy is simply the number of sequence positions for which all labels were correct.

Each row in Table 3.1 is the average of five different random subsets of the training data, except for row 8936, which is run on the single official CoNLL training set. All conditions used the same 2012 sentences in the official test set.

On the full training set, FCRFs perform better on NP chunking than either of the cascaded approaches, including Brill+POS. The Brill tagger [12] is an established part-of-speech tagger whose training set is not only over four times bigger than the CoNLL 2000 data set, but also includes the WSJ corpus from which the CoNLL 2000 test set was derived. The Brill tagger is 97% accurate on the CoNLL data. Also, note that the FCRF—which predicts both noun-phrase boundaries and POS—is more accurate than a linear-chain CRF which predicts only part-of-speech. We conjecture that the NP chain captures long-run dependencies between the POS labels.

On smaller training subsets, the FCRF outperforms CRF+CRF and performs comparably to Brill+CRF. For all the training subset sizes, the difference between CRF+CRF and the FCRF is statistically significant by a two-sample t-test ($p < 0.002$). In fact, there was no subset of the data on which CRF+CRF performed better than the FCRF. The variation over the randomly selected training subsets is small—the standard deviation over the five repetitions has mean 0.39—indicating that the observed improvement is not due to chance. Performance and variance on noun-phrase chunking is shown in Figure 3.3.

[2]To simulate the effects of a cascaded architecture, the POS labels in the CoNLL-2000 training and test sets were automatically generated by the Brill tagger. Thus, POS accuracy measures agreement with the Brill tagger, not agreement with human judgments.

Method	Time (hr)		NP F1		LBFGS iter
	μ	s	μ	s	μ
Random (3)	15.67	2.90	88.57	0.54	63.6
Tree (3)	13.85	11.6	88.02	0.55	32.6
Tree (∞)	13.57	3.03	88.67	0.57	65.8
Random (∞)	13.25	1.51	88.60	0.53	76.0
Exact	20.49	1.97	88.63	0.53	73.6

Table 3.3. Comparison of F1 performance on the chunking task by inference algorithm. The columns labeled μ give the mean over five repetitions, and s the sample standard deviation. Approximate inference methods have labeling accuracy very similar to exact inference with lower total training time. The differences in training time between Tree (∞) and Exact and between Random (∞) and Exact are statistically significant by a paired t-test ($df = 4; p < 0.005$).

On this data set, several systems are statistically tied for best performance. Kudo and Matsumoto [55] report an F1 of 94.39 using a combination of voting support vector machines. Sha and Pereira [112] give a linear-chain CRF that achieves an F1 of 94.38, using a second-order Markov assumption, and including bigram and trigram POS tags as features. An FCRF imposes a first-order Markov assumption over labels, and represents dependencies only between cotemporal POS and NP label, not POS bigrams or trigrams. Thus, Sha and Pereira's results suggest that more richly-structured DCRFs could achieve better performance than an FCRF.

Other DCRF structures can be applied to many different language tasks, including information extraction. Peshkin and Pfeffer [93] apply a generative DBN to extraction from seminar announcements, attaining improved results, especially in extracting locations and speakers, by adding a factor to remember the identity of the last non-background label.

3.1.6.2 Comparison of Inference Algorithms

Because DCRFs can have rich graphical structure, and require many marginal computations during training, inference is critical to efficient training with many labels and large data sets. In this section, we compare different inference methods both on training time and labeling accuracy of the final model.

Because exact inference is feasible for a two-chain FCRF, this provides a good case to test whether the final classification accuracy suffers when approximate methods are used to calculate the gradient. Also, we can compare different methods for approximate inference with respect to speed and accuracy.

We train factorial CRFs on the noun-phrase chunking task described in the last section. We compute the gradient using exact inference and approximate belief propagation using both random and tree-based schedules, as described in Section 3.1.2. Algorithms are considered to have converged when no message changes by more than 10^{-3}. In these experiments, the approximate BP algorithms always converged, although this is not guaranteed in general. We trained on five random subsets of 5% of the training data, and the same five subsets were used in each condition. All experiments were performed on a 2.8 GHz Intel Xeon with 4 GB of memory.

For each message-passing schedule, we compare two termination conditions: terminating on convergence (Random(∞) and Tree(∞) in Table 3.3) and terminating after three iterations (Random (3) and Tree (3)). Although the early-terminating BP runs are less accurate, they are faster, which we hypothesized could result in lower overall training time. If the gradient is too inaccurate, however, then the optimization will require many more iterations, resulting in greater training time overall, even though the time per gradient computation is lower. Another hazard is that no maximizing step may be possible along the approximate gradient, even if one is possible along the true gradient. In this case, the gradient descent algorithm terminates prematurely, leading to decreased performance.

Table 3.3 shows the average F1 score and total training times of DCRFs trained by the different inference methods. Unexpectedly, letting the belief propagation algorithms run to convergence led to lower training time than the early cutoff. For example, even though Random(3) averaged 427 sec per gradient computation compared to 571 sec for Random(∞), Random(∞) took less total time to train, because Random(∞) needed an average of 83.6 gradient computations per training run, compared to 133.2 for Random(3).

As for final classification performance, the various approximate methods and exact inference perform similarly, except that Tree(3) has lower final performance because maximization ended prematurely, averaging only 32.6 maximizer iterations. The variance in F1 over the subsets, although not large, is much larger than the F1 difference between the inference algorithms.

In all cases, the messages were initialized to uniform messages. One might think to take advantage of the fact that BP is embedded in a gradient-based optimizer, by initializing the BP iterations at the final messages from the previous gradient step. In preliminary experiments, this did not appreciably help early stopping.

Previous work [137] has shown that TRP converges faster than *synchronous* belief propagation, that is, with Jacobi updates. Both the schedules discussed in section 3.1.2 use asynchronous Gauss-Seidel updates. We emphasize that the graphical models in these experiments are always pairs of coupled chains. On more complicated models, or with a different choice of spanning trees, tree-based updates could outperform random asynchronous updates. Also, in complex models, the difference in classification accuracy between exact and approximate inference could be larger, although in such cases exact inference is likely to be intractable.

In summary, we draw three conclusions about belief propagation on this particular model. First, using approximate inference instead of exact inference leads to lower overall training time with no loss in accuracy. Indeed, the two-level FCRFs that we consider here appear to have been particularly easy cases for BP, because we observed little difficulty with convergence. Second, there is little difference between a random tree schedule and a completely random schedule for belief propagation.

$w_t = w$
w_t matches `[A-Z][a-z]+`
w_t matches `[A-Z][A-Z]+`
w_t matches `[A-Z]`
w_t matches `[A-Z]+`
w_t matches `[A-Z]+[a-z]+[A-Z]+[a-z]`
w_t appears in list of first names, last names, honorifics, etc.
w_t appears to be part of a time followed by a dash
w_t appears to be part of a time preceded by a dash
w_t appears to be part of a date
$q_k(\mathbf{x}, t + \delta)$ for all k and $\delta \in [-4, 4]$

Table 3.4. Input features $q_k(\mathbf{x}, t)$ for the seminars data. In the above w_t is the word at position t, T_t is the POS tag at position t, w ranges over all words in the training data, and T ranges over all Penn Treebank part-of-speech tags. The "appears to be" features are based on hand-designed regular expressions that can span several tokens.

Third, running belief propagation to convergence leads both to increased classification accuracy and lower overall training time than an early cutoff.

3.1.6.3 Cascaded Training for Transfer Learning

In this section, we consider an application of DCRFs to transfer learning, both as an additional application of DCRFs, and as an evaluation of the cascaded training procedure described in Section 3.1.5. The task is to extract the details of an academic seminar—including its starting time, ending time, location, and speaker—from an email announcement. The data is a collection of 485 e-mail messages announcing seminars at Carnegie Mellon University, gathered by Freitag [35], and has been the subject of much previous work using a wide variety of learning methods. Despite all this work, however, the best reported systems have precision and recall on speaker names and locations of only about 75%—too low to use in a practical system. This task is so challenging because the messages are written by many different people, who each have different ways of presenting the announcement information.

System		stime	etime	location	speaker	overall
WHISK	Soderland [115]	92.6	86.1	66.6	18.3	65.9
SRV	Freitag [35]	98.5	77.9	72.7	56.3	76.4
HMM	Frietag and McCallum [36]	98.5	62.1	78.6	76.6	78.9
RAPIER	Califf and Mooney [16]	95.9	94.6	73.4	53.1	79.3
SNOW-IE	Roth and Wen-tau Yih [105]	**99.6**	96.3	75.2	73.8	86.2
$(LP)^2$	Ciravegna [20]	99.0	95.5	75.0	**77.6**	86.8
CRF (no transfer)	This chapter	99.1	**97.3**	81.0	73.7	87.8
FCRF (cascaded)	This chapter	99.2	96.0	84.3	74.2	88.4
FCRF (joint)	This chapter	99.1	96.0	**85.3**	76.3	**89.2**

Table 3.5. Comparison of F_1 performance on the seminars data. Joint decoding performs significantly better than cascaded decoding. The overall column is the mean of the other four. (This table was adapted from Peshkin and Pfeffer [93].)

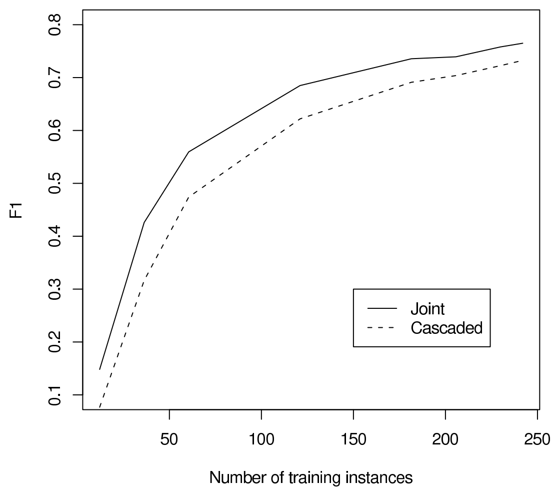


Figure 3.4. Learning curves for the seminars data set on the speaker field, averaged over 10-fold cross validation. Joint training performs equivalently to cascaded decoding with 25% more data.

Because the task includes finding locations and person names, the output of a named-entity tagger is a useful feature. It is not a perfectly indicative feature, however, because many other kinds of person names appear in seminar announcements—for example, names of faculty hosts, departmental secretaries, and sponsors of lecture series. For example, the token *Host:* indicates strongly both that what follows is a person name, but that person is not the seminars' speaker.

Even so, named-entity predictions do improve performance on this task. Therefore, we wish to transfer learning from the named-entity task to the seminar announcement task. To do this, we define an FCRF that predicts both named-entity labels and seminar labels, training it using cascaded training (Section 3.1.5). Although on the noun-phrase chunking data, cascaded training performs worse than cascaded training and decoding (Section 3.1.6.1), here we do not have a single data set that is labeled for both tasks. Performing joint inference over both chains is therefore impossible during training; at test time, however, we can still perform joint inference over both chains. We call this procedure *joint decoding*, as opposed to the cascaded procedure of using the single-best named-entity label in the seminar predictor. Joint decoding might be expected to perform better because of helpful feedback between the tasks: Information from the seminar-field predictions can improve named-entity predictions, which in turn improve the seminar-field predictions. Therefore, we present two comparisons: (a) between the FCRF trained to incorporate transfer and a comparable linear-chain CRF, and (b) at test time, between cascaded decoding or joint decoding.

We use the predictions from a CRF named-entity tagger that we train on the standard CoNLL 2003 English data set. The CoNLL 2003 data set consists of newswire articles from Reuters labeled as either people, locations, organizations, or miscellaneous entities. It is much larger than the seminar announcements data set. While the named-entity data contains 203,621 tokens for training, the seminar announcements data set contains only slightly over 60,000 training tokens.

Previous work on the seminars data has used a one-field-per-document evaluation. That is, for each field, the CRF selects a single field value from its Viterbi path, and this extraction is counted as correct if it exactly matches any of the true field mentions in the document. We compute precision and recall following this convention, and report their harmonic mean F_1. As in the previous work, we use 10-fold cross validation with a 50/50 training/test split. We use a spherical Gaussian prior on parameters with variance $\sigma^2 = 0.5$.

We evaluate whether joint decoding with cascaded training performs better than cascaded training and decoding. Table 3.5 compares cascaded and joint decoding for CRFs with other previous results from the literature.[3] The features we use are listed in Table 3.4. Although previous work has used very different feature sets, we include a no-transfer CRF baseline to assess the impact of transfer from the CoNLL data set. All the CRF runs used exactly the same features.

On the most challenging fields, location and speaker, cascaded transfer is more accurate than no transfer at all, and joint decoding is more accurate than cascaded decoding. In particular, for speaker, we see an error reduction of 8% by using joint decoding over cascaded. The difference in F1 between cascaded and joint decoding is statistically significant for speaker (paired t-test; $p = 0.017$) but only marginally significant for location ($p = 0.067$). Our results are competitive with previous work; for example, on location, the CRF is more accurate than any of the existing systems, and the CRF has the highest overall performance, that is, averaged over all fields, than the previously reported systems.

Figure 3.4 shows the difference in performance between joint and cascaded decoding as a function of training set size. Cascaded decoding with the full training set of

[3]We omit one relevant paper [93] because its evaluation method differs from all the other previous work.

242 emails performs equivalently to joint decoding on only 181 training instances, a 25% reduction in the training set.

Examining the trained models, we can observe errors made by the general-purpose named entity tagger, and how they can be corrected by considering the seminars labels. In newswire text, long runs of capitalized words are rare, often indicating the name of an entity. In email announcements, runs of capitalized words are common in formatted text blocks like:

```
Location:  Baker Hall
    Host:  Michael Erdmann
```

In this type of situation, the general named entity tagger often mistakes *Host:* for the name of an entity, especially because the word preceding *Host* is also capitalized. On one of the cross-validated testing sets, of 80 occurrences of the word *Host:*, the named-entity tagger labels 52 as some kind of entity. When joint decoding is used, however, only 20 occurrences are labeled as entities. Recall that in both of these settings, training is performed in exactly the same way; the only difference is that joint decoding takes into account information about the seminar labels when choosing named-entity labels. This is an example of how domain-specific information from the main task can improve performance on a more standard, general-purpose subtask.

3.1.7 Related Work

Since the original work on conditional random fields [58], there has been much interest in training discriminative models with more general graphical structures. One of the first such applications was relational Markov networks [128], which were first applied to collective classification of Web pages. There has also been interest in grid-structured loopy CRFs for computer vision [46, 57], in which jointly-trained Markov random fields are a classical technique. Another type of structured problem which has seen some attention in the literature is discriminative learning of distributions over context-free parse trees, in which training has done done using max-margin methods [74, 130] and perceptron-like methods [135].

Currently, the most popular alternative approaches to training structured discriminative models are maximum-margin training [3, 129], and perceptron training [22], which has been especially popular in NLP because of its ease of implementation.

The factorial CRF that we present here should not be confused with the factorial Markov random fields that have been proposed in the computer vision community [52]. In that model, each of the factors is a grid, rather than a chain, and they interact through a directed model, as in a factorial HMM.

Finally, some results presented here have appeared in earlier conference versions, in particular the results on noun-phrase chunking [125] and transfer learning [120].

3.2 Skip-chain CRFs

Another type of long-range dependence that arises in information extraction occurs occurs on repeated mentions of the same field. When the same entity is mentioned more than once in a document, such as *Robert Booth*, in many cases all mentions have the same label, such as SEMINAR-SPEAKER. We can take advantage of this fact by favoring labelings that treat repeated words identically, and by combining features from all occurrences so that the extraction decision can be made based on global information. Furthermore, identifying all mentions of an entity can be useful in itself, because each mention might contain different useful information. However, most extraction systems, whether probabilistic or not, do not take advantage of this dependency, instead treating the separate mentions independently.

To perform collective labeling, we need to represent dependencies between distant terms in the input. But this reveals a general limitation of sequence models, whether generatively or discriminatively trained. Sequence models make a Markov assumption among labels, that is, that any label y_t is independent of all previous labels given

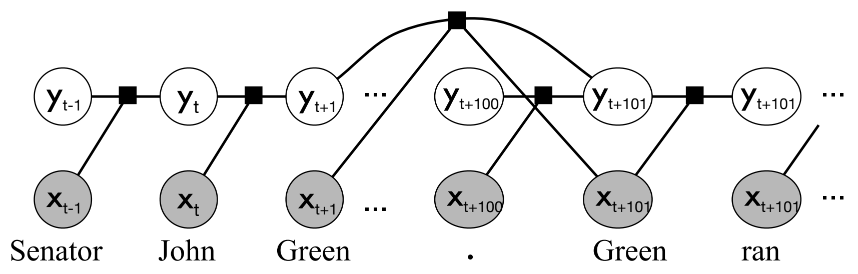


Figure 3.5. Graphical representation of a skip-chain CRF. Identical words are connected because they are likely to have the same label.

its immediate predecessors $y_{t-k} \dots y_{t-1}$. This represents dependence only between nearby nodes—for example, between bigrams and trigrams—and cannot represent the higher-order dependencies that arise when identical words occur throughout a document.

To relax this assumption, we introduce the *skip-chain CRF*, a conditional model that collectively segments a document into mentions and classifies the mentions by entity type, while taking into account probabilistic dependencies between distant mentions. These dependencies are represented in a skip-chain model by augmenting a linear-chain CRF with factors that depend on the labels of distant but similar words. This is shown graphically in Figure 3.5.

Even though the limitations of n-gram models have been widely recognized within the natural language processing community, long-distance dependencies are difficult to represent in generative models, because full n-gram models have too many parameters if n is large. We avoid this problem by selecting which skip edges to include based on the input string. This kind of input-specific dependence is difficult to represent in a generative model, because it makes generating the input more complicated. In other words, conditional models have been popular because of their flexibility in allowing overlapping features; skip-chain CRFs take advantage of their flexibility in allowing input-specific model structure.

3.2.1 Model

The skip-chain CRF is essentially a linear-chain CRF with additional long-distance edges between similar words. We call these additional edges *skip edges.* The features on skip edges can incorporate information from the context of both endpoints, so that strong evidence at one endpoint can influence the label at the other endpoint.

When applying the skip-chain model, we must choose which skip edges to include. The simplest choice is to connect all pairs of identical words, but more generally we can connect any pair of words that we believe to be similar, for example, pairs of words that belong to the same stem class, or have small edit distance. In addition, we must be careful not to include too many skip edges, because this could result in a graph that makes approximate inference difficult. So we need to use similarity metrics that result in a sufficiently sparse graph. In the experiments below, we focus on named-entity recognition, so we connect pairs of identical capitalized words.

Formally, the skip-chain CRF is defined as a general CRF with two clique templates: one for the linear-chain portion, and one for the skip edges. For an sentence $\mathbf{x}$, let $\mathcal{I} = \{(u, v)\}$ be the set of all pairs of sequence positions for which there are skip edges. For example, in the experiments reported here, $\mathcal{I}$ is the set of indices of all pairs of identical capitalized words. Then the probability of a label sequence $\mathbf{y}$ given an input $\mathbf{x}$ is modeled as

$$p_\theta(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}) \prod_{(u,v)\in\mathcal{I}} \Psi_{uv}(y_u, y_v, \mathbf{x}), \tag{3.12}$$

where Ψ_t are the factors for linear-chain edges, and Ψ_{uv} are the factors over skip edges. These factors are defined as

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\left\{\sum_k \lambda_{1k} f_{1k}(y_t, y_{t-1}, \mathbf{x}, t)\right\} \tag{3.13}$$

$$\Psi_{uv}(y_u, y_v, \mathbf{x}) = \exp\left\{\sum_k \lambda_{2k} f_{2k}(y_u, y_v, \mathbf{x}, u, v)\right\}, \tag{3.14}$$

where $\theta_1 = \{\lambda_{1k}\}_{k=1}^{K_1}$ are the parameters of the linear-chain template, and $\theta_2 = \{\lambda_{2k}\}_{k=1}^{K_2}$ are the parameters of the skip template. The full set of model parameters are $\theta = \{\theta_1, \theta_2\}$.

As described in Section 2.4.6, both the linear-chain features and skip-chain features are factorized into indicator functions of the outputs and observation functions, as in (2.83). In general the observation functions $q_k(\mathbf{x}, t)$ can depend on arbitrary positions of the input string. For example, a useful feature for NER is $q_k(\mathbf{x}, t) = 1$ if and only if x_{t+1} is a capitalized word.

The observation functions for the skip edges are chosen to combine the observations from each endpoint. Formally, we define the feature functions for the skip edges to factorize as:

$$f'_k(y_u, y_v, \mathbf{x}, u, v) = \mathbf{1}_{\{y_u = \tilde{y}_u\}} \mathbf{1}_{\{y_v = \tilde{y}_v\}} q'_k(\mathbf{x}, u, v) \tag{3.15}$$

This choice allows the observation functions $q'_k(\mathbf{x}, u, v)$ to combine information from the neighborhood of y_u and y_v. For example, one useful feature is $q'_k(\mathbf{x}, u, v) = 1$ if and only if $x_u = x_v =$ "Booth" and $x_{v-1} =$ "Speaker:". This can be a useful feature if the context around x_u, such as "Robert Booth is manager of control engineering. . . ," may not make clear whether or not Robert Booth is presenting a talk, but the context around x_v is clear, such as "Speaker: Robert Booth." [4]

[4]This example is taken from an actual error made by a linear-chain CRF on the seminars data set. We present results from this data set in Section 3.2.3.

System	stime	etime	location	speaker	overall
BIEN [93]	96.0	**98.8**	87.1	76.9	89.7
Linear-chain CRF	**97.5**	97.5	**88.3**	77.3	90.2
Skip-chain CRF	96.7	97.2	88.1	**80.4**	**90.6**

Table 3.6. Comparison of F_1 performance on the seminars data. The top line gives a dynamic Bayes net that has been previously used on this data set. The skip-chain CRF beats the previous systems in overall F1 and on the speaker field, which has proved to be the hardest field of the four. Overall F1 is the average of the F1 scores for the four fields.

Field	Linear-chain	Skip-chain
stime	12.6	17
etime	3.2	5.2
location	6.4	0.6
speaker	30.2	4.8

Table 3.7. Number of inconsistently mislabeled tokens, that is, tokens that are mislabeled even though the same token is labeled correctly elsewhere in the document. Learning long-distance dependencies reduces this kind of error in the speaker and location fields. Numbers are averaged over 5 folds.

3.2.2 Parameter Estimation

Because the loops in a skip-chain CRF can be long and overlapping, exact inference, and hence maximum likelihood, is intractable for the models considered here. The running time required by exact inference is exponential in the size of the largest clique in the graph's junction tree. In junction trees created from the seminars data, 29 of the 485 instances have a maximum clique size of 10 or greater, and 11 have a maximum clique size of 14 or greater. (The worst instance has a clique with 61 nodes.) These cliques are far too large to perform inference exactly. For reference, representing a single factor that depends on 14 variables requires more memory than can be addressed in a 32-bit architecture.

Instead, we perform approximate inference using loopy belief propagation, which was described in Section 2.4.4. As with the FCRFs in Section 3.1, inference uses the

TRP schedule with random spanning trees. Parameters are selected to maximize the Bethe likelihood (3.6) using limited-memory BFGS. As discussed in Section 3.1.4, the gradient of this objective is identical to that of the true likelihood, except that where the true gradient uses the true marginal distributions of the model, the gradient of the Bethe likelihood uses the beliefs resulting from BP.

It is important to carefully choose the initial parameter setting for the optimization procedure. At first, this may seem surprising, because the likelihood is a concave function of the parameters, and so gradient-based optimization should find the global optimum from any starting point. In fact, initialization does matter, for two reasons. First, even though the true likelihood is convex, the BP approximation to the likelihood is not, so it is possible to find a local maximum that is not globally optimal. If this happens, it means that the final parameter setting has a zero BP gradient with respect to one fixed point, but not with respect to other fixed points.

A second explanation, and perhaps more relevant, is that many parameter settings have likelihoods that are numerically very close to optimal, but perform differently on unseen data. In the models considered in this thesis, the features are often rich enough that it is possible to fit the training data arbitrarily closely. Furthermore, because the features are linearly dependent, there are many different lines in parameter space that approach the empirical expectations from different directions. The numerical optimization algorithm terminates when the difference in value or gradient becomes too small, so approaching the maximum from different directions can still lead to different parameter setting. As a concrete example, consider the two-level DCRF of the previous section. Using only the within-chain features, it is possible to find solutions with likelihood only slightly worse than the full model, which uses both within-chain and between-chain factors. But we saw in the experimental results that using the between-chain factors improves accuracy on unseen data. So this is case where a poor initialization of the optimization procedure, namely, one that encouraged the within-chain factors to be given too much weight, can significantly degrade accuracy.

In the experiments reported in the next section, the skip-chain CRF is initialized from a linear-chain CRF. That is, the linear-chain factors of the model are initialized from the weights of a fully-trained linear-chain CRF, and the long-distance factors are initialized to uniform. If we instead start at the uniform distribution—that is, by initializing all parameters to 0—not only does loopy BP training take much longer, but testing performance is much worse, because the convex optimization procedure has difficulty with noisier gradients. With uniform initialization, loopy BP does not converge for all training instances, especially at early iterations of training. That is, carefully initializing the parameters avoids regions of parameter space in which BP performs poorly.

3.2.3 Results

We evaluate skip-chain CRFs on the seminar announcements data set discussed in Section 3.1.6.3. The messages are annotated with the seminar's starting time, ending time, location, and speaker. Often the fields are listed multiple times in the message. For example, the speaker name might be included both near the beginning and later on, in a sentence like "If you would like to meet with Professor Smith. . . " As mentioned earlier, it can be useful to find both such mentions, because different information can occur in the surrounding context of each mention: for example, the first mention might be near an institutional affiliation, while the second mentions that Smith is a professor.

We evaluate a skip-chain CRF with skip edges between identical capitalized words. The motivation for this is that the hardest aspect of this data set is identifying speakers and locations, and capitalized words that occur multiple times in a seminar announcement are likely to be either speakers or locations.

Table 3.4 shows the list of input features we used. For a skip edge (u, v), the input features we used were the disjunction of the input features at u and v, that is,

$$q'_k(\mathbf{x}, u, v) = q_k(\mathbf{x}, u) \oplus q_k(\mathbf{x}, v) \tag{3.16}$$

where $\oplus$ is binary or. All of our results are averaged over 5-fold cross-validation with an 80/20 split of the data. We report results from both a linear-chain CRF and a skip-chain CRF with the same set of input features.

We calculate precision and recall as[5]

$$P = \frac{\text{\# tokens extracted correctly}}{\text{\# tokens extracted}}$$
$$R = \frac{\text{\# tokens extracted correctly}}{\text{\# true tokens of field}}$$

As usual, we report $F_1 = (2PR)/(P + R)$.

Table 3.6 compares a skip-chain CRF to a linear-chain CRF and to a dynamic Bayes net used in previous work [93]. The skip-chain CRF performs much better than all the other systems on the Speaker field, which is the field for which the skip edges would be expected to make the most difference. On the other fields, however, the skip-chain CRF does slightly worse (less than 1% absolute F1).

We expected that the skip-chain CRF would do especially well on the speaker field, because speaker names tend to appear multiple times in a document, and a skip-chain CRF can learn to label the multiple occurrences consistently. To test this hypothesis, we measure the number of *inconsistently mislabeled* tokens, that is, tokens

[5]Previous work on this data set has traditionally reported precision and recall only at the document level, that is, from each document the system extracts only one field of each type. Because the goal of the skip-chain CRF is to extract all mentions in a document, these metrics are inappropriate, so we cannot compare with this previous work. Peshkin and Pfeffer [93] do use the per-token metric (personal communication), so our comparison is fair in that respect.

that are mislabeled even though the same token is classified correctly elsewhere in the document. Table 3.7 compares the number of inconsistently mislabeled tokens in the test set between linear-chain and skip-chain CRFs. For the linear-chain CRF, on average 30.2 true speaker tokens are inconsistently mislabeled. Because the linear-chain CRF mislabels 121.6 true speaker tokens, this situation includes 24.7% of the missed speaker tokens.

The skip-chain CRF shows a dramatic decrease in inconsistently mislabeled tokens on the speaker field, from 30.2 tokens to 4.8. Consequently, the skip-chain CRF also has much better recall on speaker tokens than the linear-chain CRF (70.0 R linear chain, 76.8 R skip chain). This explains the increase in F1 from linear-chain to skip-chain CRFs, because the two have similar precision (86.5 P linear chain, 85.1 skip chain). These results support the original hypothesis that treating repeated tokens consistently especially benefits recall on the Speaker field.

On the Location field, on the other hand, where we might also expect skip-chain CRFs to perform better, there is no benefit. We explain this by observing in Table 3.7 that inconsistent misclassification occurs much less frequently in this field.

3.2.4 Related Work

Bunescu and Mooney [14] have used a relational Markov network to collectively classify the mentions in a document, achieving increased accuracy by learning dependencies between similar mentions. In their work, candidate phrases are extracted heuristically, which can introduce errors if a true entity is not selected as a candidate phrase. Our model performs collective segmentation and labeling simultaneously, so that the system can take into account dependencies between the two tasks.

After we first presented the skip-chain CRF [118], several other authors have introduced interesting extensions. As one extension, Finkel et al. [32] augment the skip-chain model with richer kinds of long-distance factors than just over pairs of words. These factors are useful for modeling exceptions to the assumption that similar words tend to have similar labels. For example, in named-entity recognition, the word *China* is as a place name when it appears alone, but when it occurs within the phrase *The China Daily*, it should be labeled as a organization. Because this model is more complex than the original skip-chain model, Finkel et al. estimate its parameters in two stages, first training the linear-chain component as a separate CRF, and then heuristically selecting parameters for the long-distance factors. Finkel et al. report improved results both on the seminars data set that we consider in this chapter, and on several other standard information extraction data sets.

An alternative to the skip-chain CRF, Rosenberg et al. [104] propose an MEMM with long-distance edges. This results in some nodes having many parents, so in order to reduce the number of parameters, every conditional probability table that has multiple parents is assumed to be a mixture of CPTs involving single parents. This mixture-of-parents assumption is similar to the restriction to pairwise factors in the skip-chain CRF. The potential advantage of such a model is that training is much simpler and more computationally efficient than the skip-chain CRF. The potential disadvantage is label bias, that is, that observations late in the sequence have no effect on earlier labels.

3.3 Summary

In this chapter, I have presented two simple loopy extensions to linear-chain CRFs: dynamic CRFs and skip-chain CRFs. Dynamic CRFs are conditionally-trained undirected sequence models with repeated graphical structure and tied parameters. They combine the best of both conditional random fields and the widely successful dynamic Bayesian networks (DBNs). DCRFs address difficulties of DBNs, by easily incorporating arbitrary overlapping input features, and of previous conditional models, by allowing more complex dependence between labels. Inference in DCRFs can be done using approximate methods, and training can be done by maximum a posteriori estimation.

Empirically, we have shown that factorial CRFs can be used to jointly perform several labeling tasks at once, sharing information between them. Such a joint model performs better than a model that does the individual labeling tasks sequentially, and has potentially many practical implications, because cascaded models are ubiquitous in NLP.

The skip-chain CRF segments a sequence while modeling long-distance dependencies between similar tokens. The skip-chain CRF can also be viewed as performing extraction while taking into account a simple form of coreference information, since the reason that identical words are likely to have similar tags is that they are likely to be coreferent. Thus, the skip-chain CRF, like the FCRF, can be viewed as a step toward joint probabilistic models for extraction and data mining as advocated by McCallum and Jensen [69].

CHAPTER 4
PIECEWISE TRAINING

This chapter begins our investigation of approximate methods for training CRFs. One attractive family of approximate training methods is *local training* methods, by which I mean methods that depend on sums of local functions of only a few factors, such as the conditional probability of a node given its Markov blanket, rather than on global functions of the entire graph, such the likelihood. The best-known example of a local training method is Besag's pseudolikelihood [8], which is a product of per-node conditional probabilities.

In this chapter, I present a novel local training method called *piecewise training*, in which the model's factors are divided into possibly overlapping sets of *pieces*, which are each trained separately. At test time, the resulting weights are used just as if they had been trained using maximum likelihood, that is, on the unseen data they are used to predict the labels using a standard approximate inference algorithm, such as max-product BP. When using piecewise training, the modeler must decide how to split the model into pieces before training. In this thesis most of the experiments use the factor-as-piece approximation, in which each factor of the model is placed in a separate piece.

This training procedure can be viewed in two ways. First, separate training of each piece can be accomplished by numerically maximizing an approximation to the likelihood. This approximate likelihood can be seen as the true likelihood on a transformation of the original graph, which I call the node-split graph, in which each of the pieces is an isolated component. The second view is based on belief propagation;

namely, the objective function of piecewise training is the same as the BP approximate likelihood (3.6) with uniform messages, as if BP has been stopped after zero iterations. I call this the *pseudomarginal* view of piecewise training, for reasons explained in Section 4.4. These two viewpoints will prove useful both for understanding these algorithms, and for designing the extensions in this chapter and the next.

In this chapter, I define piecewise training (Section 4.1), explaining it from both the local graph and the pseudomarginal perspectives. I apply the factor-as-piece approximation to several natural-language data sets. The model resulting from the piecewise approximation has better accuracy than pseudolikelihood and is sometimes comparable to exact maximum likelihood (Section 4.3). Then I consider several extensions to the basic method. First, I consider an extension called *reweighted piecewise training*, based on a connection between piecewise training and the upper bounds of Wainwright et al. [140], but unfortunately the results here are negative: reweighted piecewise training has worse accuracy than standard piecewise on our data. A more interesting family of extensions is based on the connection to standard belief propagation. To develop these, I introduce three views of approximate training algorithms, which I call the neighborhood graph view, the pseudomarginal view, and the belief view (Section 4.4). This development motivates two different extensions of piecewise training, namely *shared-unary piecewise* (Section 4.5.1) and *one-step cutout* (Section 4.5.3). Just as standard piecewise corresponds to zero iterations of BP, shared-unary corresponds to 1 iteration and one-step cutout to 2 iterations. Simulated data provides illuminating insight into when shared-unary piecewise and one-step cutout may be more appropriate than standard piecewise (Section 4.5.5).

4.1 Definition

In this section, I present piecewise training, explaining how it maximizes a loose lower bound on the likelihood. The motivation is that in some applications, the local information in each factor alone, without performing inference, is enough to do fairly well at predicting the outputs, but some amount of global information can help. Therefore, to reduce training time, it makes sense to perform less inference at training time than at test time, because at training time we loop through the examples repeatedly, whereas at test time we only need to make each prediction once. For example, suppose we want to train a loopy pairwise MRF. In piecewise estimation, what we will do is to train the parameters of each edge independently, as if each edge were a separate two-node MRF of its own. Finally, on test data, the parameters resulting from this local training become the parameters used to perform global inference, using some standard approximate inference algorithm.

Now I define the piecewise estimator more generally. Let the distribution $p(\mathbf{y})$ be defined by a factor graph, where $\Psi_a(\mathbf{y}_a, \theta)$ has the exponential form (2.3), and suppose that we wish to estimate θ. (To simplify notation, I describe piecewise estimation for generative models; the conditional case is exactly analogous.) I assume that the model's factors are divided into a set $\mathcal{P} = \{R_0, R_1 \ldots\}$ of pieces; each piece $R \in \mathcal{P}$ is a set of factors $R = \{\Psi_a\}$. The pieces need not be disjoint. For example, in a grid-shaped MRF with unary and pairwise factors, we might isolate each factor in its own piece, or alternatively we might choose one piece for each row and each column of the MRF, in which case each unary factor would be shared between its row piece and its column piece.

To train the pieces separately, each piece R has a local likelihood

$$\ell_R(\theta) = \sum_{a \in R} \sum_{k=1}^{K} \theta_{ak} f_{ak}(\mathbf{y}_a) - A_R(\theta). \tag{4.1}$$

where $A_R(\theta)$ is the *local log partition function* for the piece, that is,

$$A_R(\theta) = \log \sum_{\mathbf{y}_R} \exp\{\sum_{a \in R} \sum_{k=1}^{K} \theta_{ak} f_{ak}(\mathbf{y}_a)\}, \tag{4.2}$$

where $\mathbf{y}_R$ is the vector of variables used anywhere in piece R. This is the likelihood for the piece R if it were a completely separate graphical model. If the pieces are disjoint, and no parameters are shared between distinct factors, then we could train each piece by separately computing parameters $\theta^R_{\text{PW}} = \max_{\theta_R} \ell_R(\theta_R)$. But in fact we would like to handle both parameter tying and overlapping pieces. To do this, we instead perform a single optimization, maximizing the sum of all of the single-piece likelihoods. So for a set $\mathcal{P}$ of pieces, the piecewise likelihood becomes

$$\ell_{\text{PW}}(\theta) = \sum_{R \in \mathcal{P}} \sum_{a \in R} \theta_a \phi_a(\mathbf{y}_a) - \sum_{R \in \mathcal{P}} A_R(\theta). \tag{4.3}$$

For example, consider the special case of per-edge pieces in a pairwise MRF with no tied parameters. Then, for an edge (s,t), we have $A_{st}(\theta) = \log \sum_{y_s, y_t} \Psi(y_s, y_t)$, so that the piecewise estimator corresponds exactly to training independent probabilistic classifiers on each edge.

Now let us compare the approximate likelihood (4.3) to the exact likelihood. Recall that the true likelihood is

$$\ell(\theta) = \sum_k \theta_k \phi_k(\mathbf{y}_k) - A(\theta)$$
$$A(\theta) = \log \sum_{\mathbf{y}} \exp\{\sum_k \theta_k \phi_k(\mathbf{y}_k)\}.$$

Notice that the first summation contains exactly the same terms as in the exact likelihood. The only difference between the piecewise objective and the exact likelihood is in the second summation of (4.3). So $A_{\text{PW}}(\theta) = \sum_R A_R(\theta)$ can be viewed as an approximation of the log partition function.

A choice of pieces to which I devote particular attention is the factor-as-piece approximation, in which each factor in the model is assigned to its own piece. There is a potential ambiguity in this choice, because recall that the factors take the form

$$\Psi_a(\mathbf{y}_a) = \exp\{\sum_k \theta_{ak} f_{ak}(\mathbf{y}_a)\},$$

so that each factor has multiple parameters and sufficient statistics. But we could just as well place each sufficient statistic in a factor all to itself, that is,

$$\Psi_{ak}(\mathbf{y}_a) = \exp\{\theta_{ak} f_{ak}(\mathbf{y}_a)\}, \tag{4.4}$$

and define the pieces at that level of granularity. Such a fine-grained choice of pieces could be useful. For example, in a linear-chain model, we might choose to view the model as a weighted finite-state machine, and partition the state-transition diagram into pieces. For the purposes of this thesis, however, when I use the factor-as-piece approximation, I will not use the fine-grained factorization of (4.4), that is, I will assume that the graph has been constructed so that no two factors share exactly the same support.

Apart from its intuitive plausibility, another rationale for the piecewise estimator is provided by the following proposition:

Proposition 4.1. *For any set $\mathcal{P}$ of pieces, the piecewise partition function is an upper bound on the true partition function:*

$$A(\theta) \leq \sum_{R \in \mathcal{P}} A_R(\theta). \tag{4.5}$$

Proof. The bound is immediate upon expansion of $A(\theta)$.

$$A(\theta) = \log \sum_{\mathbf{x}} \exp \left\{ \sum_{a} \theta_a \phi_a(\mathbf{x}_a) \right\} \tag{4.6}$$

$$= \log \sum_{\mathbf{x}} \prod_{R \in \mathcal{P}} \exp \left\{ \sum_{a \in R} \theta_a \phi_a(\mathbf{x}_a) \right\} \tag{4.7}$$

$$\leq \log \prod_{R \in \mathcal{P}} \sum_{\mathbf{x}_R} \exp \left\{ \sum_{a \in R} \theta_a \phi_a(\mathbf{x}_a) \right\} \tag{4.8}$$

$$= \sum_{R \in \mathcal{P}} A_R(\theta). \tag{4.9}$$

The bound from (4.7) to (4.8) is justified by considering the expansion of the product in (4.8). The expansion contains every term of the summation in (4.7), and all terms are nonnegative. $\square$

Therefore, the piecewise likelihood is a lower bound on the true likelihood. If the graph is connected, however, then the bound is nowhere tight.

Although so far I have been using the notation of generative models for simplicity, estimation is especially well-suited for conditional random fields. As mentioned earlier, standard maximum-likelihood training for CRFs can require evaluating the instance-specific partition function $Z(\mathbf{x})$ for each training instance for each iteration of an optimization algorithm, which can be expensive even for linear chains. By using piecewise training, we need to compute only local normalization over small cliques, which for loopy graphs is potentially much more efficient.

4.1.1 The Node-split Graph

The piecewise likelihood (4.3) can be viewed as the exact likelihood in a transformation of the original graph. In the transformed graph, we split the variables, adding one copy of each variable for each factor that it participates in, as pictured in Figure 4.1. We call the transformed graph the *node-split graph*.

Formally, the splitting transformation is as follows. Given a factor graph G, create a new graph G' with variables $\{y_{as}\}$, where a ranges over all factors in G and s over

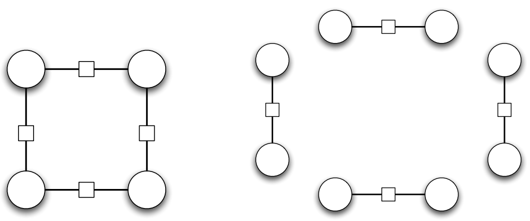


Figure 4.1. Example of node splitting. Left is the original model, right is the version trained by piecewise. In this example, there are no unary factors.

all variables in a. For any factor a, let π_a map variables in G to their copy in G', that is, $\pi_a(y_s) = y_{as}$ for any variable s in G. Finally, for each factor $\Psi_a(y_a, \theta)$ in G, add a factor Ψ'_a to G' as

$$\Psi'_a(\pi_a(y_a), \theta) = \Psi_a(y_a, \theta). \tag{4.10}$$

If we wish to use pieces that are larger than a single factor, then the definition of the node-split graph can be modified accordingly.

Clearly, piecewise training in the original graph is equivalent to exact maximum likelihood training in the node-split graph. This view of piecewise training will prove useful in the extensions presented in Section 4.5 and in Chapter 5.

4.1.2 The Belief Propagation Viewpoint

Another way of understanding piecewise training arises from belief propagation. Let $M = \{m_{ai}(y_i)\}$ be a set of BP messages, not necessarily converged. We view all of a factor's outgoing messages as approximating it, that is, we define $\tilde{\Psi}_a = \prod_{i \in a} m_{ai}$. Recall from Section 2.1.4 that the dual energy of belief propagation yields an approximate partition function

$$Z_{\text{BP}}(\theta, M) = \prod_a \left(\sum_{\mathbf{y}_a} \frac{\Psi_a(\mathbf{y}_a, \theta)}{\tilde{\Psi}_a(\mathbf{y}_a)} q(y_a) \right) \prod_i \left(\sum_{y_i} q(y_i) \right)^{1-d_i}, \tag{4.11}$$

where q denotes the unnormalized beliefs

$$q(\mathbf{y}) = \prod_a \tilde{\Psi}_a(\mathbf{y}_a) = \prod_a \prod_i m_{ai}(y_i), \tag{4.12}$$

with $q(\mathbf{y}_a) = \sum_{\mathbf{y}\backslash\mathbf{y}_a} q(\mathbf{y})$ and $q(y_i) = \sum_{\mathbf{y}\backslash y_i} q(\mathbf{y})$.

Now, let M_0 be the uniform message setting, that is, $m_{ai} = 1$ for all a and i. This is a common initialization for BP. Then the unnormalized beliefs are $q(\mathbf{y}) = 1$ for all $\mathbf{y}$, and the approximate partition function is

$$Z_{\text{BP}}(\theta, M_0) = \prod_a \left(C_a \sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \theta) \right) \prod_i C_i^{1-d_i}, \tag{4.13}$$

where C_a and C_i are constants that do not depend on θ. This approximate partition function is the same as that used by piecewise training with one factor per piece, up to a multiplicative constant that does not change the gradient. So another view is that piecewise training approximates the likelihood using belief propagation, except that we cut off BP after 0 iterations. This view informs the training methods that I introduce in Section 4.5 and in Chapter 6.

4.1.3 Pseudo-Moment Matching Viewpoint

Piecewise training is based on the intuition that if all of the local factors fit the data well, then the resulting global distribution is likely to be reasonable. An interesting way of formalizing this idea is by way of the *pseudo-moment matching* estimator of Wainwright et al. [142]. In this section, I show that there is a sense in which piecewise training can be viewed as an extension of the pseudo-moment matching estimator.

First, consider the case in which $p(\mathbf{y})$ factorizes according to a graph G with fully-parameterized tables, that is,

$$\Psi_a(\mathbf{y}_a) = \exp\{\sum_{\mathbf{y}'_a} \theta(\mathbf{y}'_a) \mathbf{1}_{\{\mathbf{y}_a = \mathbf{y}'_a\}}\} \tag{4.14}$$

Let $\tilde{p}(\mathbf{y})$ be the empirical distribution, that is, $\tilde{p}(\mathbf{y}) \propto \sum_i \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}}$.

The pseudo-moment matching estimator chooses parameters that maximize the BP likelihood (3.6) without actually computing any of the message updates. This estimator is

$$\begin{aligned}\hat{\theta}_a(\mathbf{y}_a) &= \log \frac{\tilde{p}(\mathbf{y}_a)}{\prod_{s \in a} \tilde{p}(y_s)} \\ \hat{\theta}_s(y_s) &= \log \tilde{p}(y_s).\end{aligned} \tag{4.15}$$

For these parameters, there exists a set of messages that (a) are a fixed-point of BP, and (b) the resulting beliefs q_a and q_s equal the empirical marginals. This can be seen using the reparameterization perspective of BP [141] described in Section 2.1.4, because with those parameters the belief-based updates of (2.26) yield a fixed point immediately.

In this thesis, however, we are interested in estimating the parameters of conditional distributions $p(\mathbf{y}|\mathbf{x})$. A simple generalization is to require for all inputs $\mathbf{x}$ with $\tilde{p}(\mathbf{x}) > 0$ that

$$\begin{aligned}\Psi_a(\mathbf{y}_a, \mathbf{x}) &= \frac{\tilde{p}(y_a|\mathbf{x})}{\prod_{s \in A} \tilde{p}(y_s|\mathbf{x})} \\ \Psi_s(y_s, \mathbf{x}) &= \tilde{p}(y_s|\mathbf{x}).\end{aligned} \tag{4.16}$$

However, we can no longer expect to find parameters that satisfy these equations in closed form. This is because the factor values of $\Psi_a(\cdot, \mathbf{x})$ ho not have an independent degree of freedom for each input value $\mathbf{x}$. Instead, to promote generalization across different inputs, the factors Ψ_a have some complex parameterization, such as the linear form (2.3), in which parameters are tied across different input values. A more useful generalization is to treat the equations (4.16) as a nonlinear set of equations to be solved. To do this, we optimize the objective function

$$\min_\theta \sum_a D(\Psi_a \| \tilde{p}_a) + \sum_s D(\Psi_s \| \tilde{p}_s), \tag{4.17}$$

where $D(\cdot\|\cdot)$ is a divergence measure. By a *divergence measure* $D(p\|q)$, I simply mean a nonnegative function that is 0 if and only if $p = q$. Then if a parameter setting θ exists such that the divergence is zero, then the equations have been solved exactly, and θ optimizes the BP likelihood.

This provides another view of piecewise training, because choosing using $\mathrm{KL}(\tilde{p}_a\|\Psi_a)$ for the divergence in (4.17) yields an equivalent optimization problem to the piecewise likelihood (4.3). This provides a justification of the intuition that fitting locally can lead to a reasonable global solution: it is not the case that fitting factors locally causes the true marginals to be matched to the empirical distribution, but it does cause the BP approximation to the marginals to be matched to the empirical distribution.

4.2 Reweighted Piecewise Training

In this section, I sketch another proof of Proposition 4.1, deriving it from the tree-reweighted bounds of Wainwright, Jaakkola, and Willsky [140], a connection which suggests generalizations of the simple piecewise training procedure. To simplify the exposition, in this section I assume the factor-as-piece approximation, but the ideas extend readily to more general disjoint pieces.

4.2.1 Tree-Reweighted Upper Bounds

Wainwright, Jaakkola, and Willsky [140] introduce a class of upper bounds on $A(\theta)$ that arise immediately from its convexity. The basic idea is to write the parameter vector θ as a mixture of parameter vectors of tractable distributions, and then apply Jensen's inequality.

Let $\mathcal{T} = \{T_R\}$ be a set of tractable subgraphs of $\mathcal{G}$. For concreteness, think of $\mathcal{T}$ as the set of all spanning trees of $\mathcal{G}$; this is in fact the case to which Wainwright, Jaakkola, and Willsky devote their attention. For each tractable graph $\mathcal{T}_R$, let $\theta(T_R)$ be an exponential parameter vector that has the same dimensionality as θ, but *respects* *the structure* of T_R. More formally, this means that the entries of $\theta(T_R)$ must be zero for factors that do not appear in T_R. Except for this, $\theta(T_R)$ is arbitrary; there is no requirement that on its own, it matches θ in any way.

Suppose we also have a distribution $\mu = \{\mu_R | T_R \in \mathcal{T}\}$ over the tractable subgraphs, such that the original parameter vector θ can be written as a combination of the per-tree parameter vectors:

$$\theta = \sum_{T_R \in \mathcal{T}} \mu_R \theta(T_R). \tag{4.18}$$

In other words, we have written the original parameters θ as a mixture of parameters on tractable subgraphs.

Then the upper bound on the log partition function $A(\theta)$ arises directly from Jensen's inequality:

$$A(\theta) = A\left(\sum_{T_R \in \mathcal{T}} \mu_R \theta(T_R)\right) \leq \sum_{T_R \in \mathcal{T}} \mu_R A(\theta(T_R)). \tag{4.19}$$

Because we have required that each graph T be tractable, each term on the right-hand side of (4.19) can be computed efficiently. If the size of $\mathcal{T}$ is large, however, then computing the sum is still intractable. We deal with this issue next.

A natural question about this bound is how to select θ so as to get the tightest upper bound possible. For fixed μ, the optimization over θ can be cast as a convex optimization problem:

$$\min_{\theta} \sum_{T_R \in \mathcal{T}} \mu_R A(\theta(T_R)) \tag{4.20}$$

$$\text{s.t. } \theta = \sum_{T_R \in \mathcal{T}} \mu_R \theta(T_R). \tag{4.21}$$

But this optimization problem can have astronomically many parameters, especially if $\mathcal{T}$ is the set of all spanning trees. The number of constraints, however, is much smaller, because the constraints are just one equality constraint for each element of θ. To collapse the dimensionality of the optimization problem, therefore, Wainwright, Jaakkola, and Willsky use the Lagrange dual of (4.20), which can then be optimized using either standard optimization techniques, or a message passing algorithm similar to to BP. For our present purposes, however, it suffices to consider only the primal problem in (4.20), which we use in the next section as a alternative derivation of piecewise bounds.

4.2.2 Application to Piecewise Upper Bounds

Now we discuss how the tree-reweighted upper bounds can be applied to piecewise training. As in the previous section, we will obtain an upper bound by writing the original parameters θ as a mixture of tractable parameter vectors $\theta(T)$. Consider the set $\mathcal{T}$ of tractable subgraphs induced by single edges of $\mathcal{G}$. Precisely, for each factor f_a in $\mathcal{G}$, we add a (non-spanning) tree T_R which contains only the factor f_a and its associated variables. With each tree T_R we associate an exponential parameter vector $\theta(T_R)$.

Let μ be a strictly positive probability distribution over factors. To use Jensen's inequality, we will need to have the constraint

$$\theta = \sum_R \mu_R \theta(T_R). \tag{4.22}$$

Now, each parameter θ_i corresponds to exactly one factor of $\mathcal{G}$, which appears in only one of the T_R. Therefore, only one choice of subgraph parameter vectors $\{\theta(T_R)\}$ meets the constraint (4.22), namely:

$$\theta(T_R) = \frac{\theta|_r}{\mu_R}, \tag{4.23}$$

where $\theta|_R$ is the restriction of θ to R; that is, $\theta|_R$ has the same entries and dimensionality as θ, but with zeros in all entries that are not included in the piece R.

Method	Overall F1
Piecewise	**91.2**
Pseudolikelihood	84.7
Per-edge PL	89.7
Exact	90.6

Table 4.1. Comparison of piecewise training to exact and pseudolikehood training on a linear-chain CRF for named-entity recognition. On this tractable model, piecewise methods are more accurate than pseudolikelihood, and just as accurate as exact training.

Therefore, using Jensen's inequality, we immediately have the bound

$$A(\theta) \leq \sum_R \mu_R A \left(\frac{\theta|_R}{\mu_R} \right). \tag{4.24}$$

This *reweighted piecewise* bound is clearly related to the basic piecewise bound in (4.5), because $A(\theta|_R)$ differs from $A_R(\theta)$ only by an additive constant which is independent of θ. In fact, a version of Proposition 4.1 can be derived by considering the limit of (4.24) as μ approaches a point mass on an arbitrary single piece R^*.

The connection to the Wainwright et al. work suggests at least two generalizations of the basic piecewise method. The first is that the reweighted piecewise bound in (4.24) can itself be minimized as an approximation to $A(\theta)$, yielding a variation of the basic piecewise method.

The second is that this line of analysis can naturally handle the case when pieces overlap. For example, in an MRF with both node and edge factors, we might choose each piece to be an edge factor with its corresponding node factors, hoping that this overlap will allow limited communication between the pieces which could improve the approximation. As long as there is a value of μ for which the constraint in (4.23) holds, then (4.24) provides a bound we can minimize in an overlapping piecewise approximation.

Method	Noun-phrase F1
Piecewise	**88.1**
Pseudolikelihood	84.9
Per-edge PL	86.5
BP	86.0

Table 4.2. Comparison of piecewise training to other methods on a two-level factorial CRF for joint part-of-speech tagging and noun-phrase segmentation.

Method	Token F1	
	location	speaker
Piecewise	**87.7**	75.4
Pseudolikelihood	67.1	25.5
Per-edge PL	76.9	69.3
BP	86.6	**78.2**

Table 4.3. Comparison of piecewise training to other methods on a skip-chain CRF for seminar announcements.

4.3 Experiments

The bound in (4.5) is not tight. Because the bound does not necessarily touch the true likelihood at any point, maximizing it is not guaranteed to maximize the true likelihood. We turn to experiments to compare the accuracy of piecewise training both to exact estimation, and to other approximate estimators. A particularly interesting comparison is to pseudolikelihood, because it is a related local estimation method.

On three real-world natural language tasks, we compare piecewise training to exact ML training, to approximate ML training using belief propagation, and to pseudolikelihood training. To be as fair as possible, we compare to two variations of pseudolikelihood, one based on nodes and a structured version based on edges. Pseudolikelihood in a generative model is normally defined as [8]:

$$\ell_{\text{PL}}(\theta) = \log \prod_s p(y_s | \mathbf{y}_{N(s)}), \tag{4.25}$$

where $N(s)$ are the set of variables that neighbor variable s. This per-variable pseudolikelihood function does not work well for sequence labeling, because it does not take into account strong interactions between neighboring sequence positions. In order to have a stronger baseline, we also compare to a per-edge version of pseudolikelihood:

$$\ell_{\text{EPL}}(\theta) = \log \prod_{st} p(y_s, y_t | \mathbf{y}_{N(s,t)}), \tag{4.26}$$

that is, instead of using the conditional distribution of each node, we use each edge, hoping to take more of the sequential interactions into account.

We evaluate piecewise training on three models: a linear-chain CRF (Section 2.3), a factorial CRF (Section 3.1), and a skip-chain CRF (Section 3.2). All of these models use a large number of input features such as word identity, part-of-speech tags, capitalization, and membership in domain-specific lexicons.

In all the experiments below, we optimize ℓ_{PW} using limited-memory BFGS. We use a Gaussian prior on weights to avoid overfitting. In previous work, the prior parameter had been tuned on each data set for belief propagation, and for the local models we used the same prior parameter without change. At test time, decoding is always performed using max-product belief propagation.

4.3.1 Linear-Chain CRF

First, we evaluate the accuracy of piecewise training on a tractable model, so that we can compare the accuracy to exact maximum-likelihood training. The task is named-entity recognition, that is, to find proper nouns in text. We use the CoNLL 2003 data set, consisting of 14,987 newswire sentences annotated with names of people, organizations, locations, and miscellaneous entities. We test on the standard development set of 3,466 sentences. Evaluation is done using precision and recall on the extracted chunks, and we report $F_1 = 2PR/P + R$. We use a linear-chain CRF, whose features are described in Table 4.4.

$w_t = w$
w_t matches `[A-Z][a-z]+`
w_t matches `[A-Z][A-Z]+`
w_t matches `[A-Z]`
w_t matches `[A-Z]+`
w_t contains a dash
w_t matches `[A-Z]+[a-z]+[A-Z]+[a-z]`
The character sequence $c_0 \dots c_n$ is a prefix of w_t (where $n \in [0, 4]$)
The character sequence $c_0 \dots c_n$ is a suffix of w_t (where $n \in [0, 4]$)
The character sequence $c_0 \dots c_n$ occurs in w_t (where $n \in [0, 4]$)
w_t appears in list of first names, last names, countries, locations, honorifics, etc.
$q_k(\mathbf{x}, t + \delta)$ for all k and $\delta \in [-2, 2]$

Table 4.4. Input features $q_k(\mathbf{x}, t)$ for the CoNLL named-entity data. In the above w_t is the word at position t, T_t is the POS tag at position t, w ranges over all words in the training data, and T ranges over all Penn Treebank part-of-speech tags. The "appears to be" features are based on hand-designed regular expressions that can span several tokens.

Piecewise training performs better than either of the pseudolikelihood methods. Even though it is a completely local training methods, piecewise training performs comparably to exact CRF training.

Now, in a linear-chain model, piecewise training has the same computational complexity as exact CRF training, so I do not mean to advocate the piecewise approximation for linear-chain graphs. Rather, that the piecewise approximation loses no accuracy on the linear-chain model is encouraging when we turn to loopy models, which we do next.

4.3.2 Factorial CRF

The first loopy model we consider is the *factorial CRF* introduced in Section 3.1. As in Chapter 3, we consider here the task of jointly predicting part-of-speech tags and segmenting noun phrases on the CoNLL 2000 data set. We report results here on subsets of 223 training sentences, and the standard test set of 2012 sentences. Results are averaged over 5 different random subsets. There are 45 different POS labels, and the three NP labels. We report F1 on noun-phrase chunks. The features used are described in Table 3.2.

In previous work, this model was optimized by approximating the partition function using belief optimization, but this was quite expensive. Training on the full data set of 8936 sentences required about 12 days of CPU time.[1]

Results on this loopy data set are presented in Table 4.2. Again, the piecewise estimator performs better both than either version of pseudolikelihood and than maximum-likelihood estimation using belief propagation. On this small subset, approximate ML training with BP requires 1.8 h, but piecewise training is still twice as fast, using 0.83 h.

4.3.3 Skip-chain CRF

Finally, we consider a model with many irregular loops, which is the skip chain model introduced in Section 3.2. This model incorporates certain long-distance dependencies between word labels into a linear-chain model for information extraction. We use the seminar extraction data set described in that section. Consistently with the previous work on this data set, we use 10-fold cross validation with a 50/50 training/test split. We report per-token F1 on the speaker and location fields, the most difficult of the four fields. The features used are described in Table 3.4. Most documents contain many crossing skip-edges, so that exact maximum-likelihood training using junction tree is completely infeasible, so instead we compare to approximate training using loopy belief propagation.

[1]This number should be taken with some skepticism, however, because it results from experiments using hardware and JVMs from 3 years ago, which are several times slower than those of today. Also, I have been developing the inference code continuously since then, so it is quite possible that the speed of my implementation has improved since then as well.

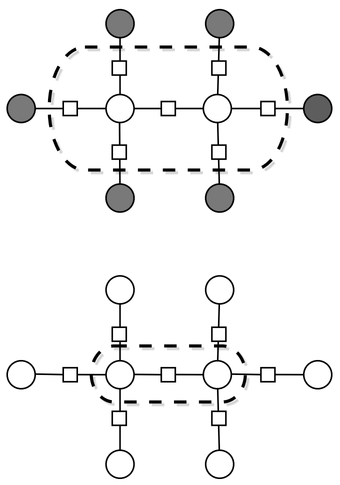


Figure 4.2. Schematic factor-graph depiction of the difference between pseudolikelihood (top) and piecewise training (bottom). Each term in pseudolikelihood normalizes the product of many factors (as circled), while piecewise training normalizes over one factor at a time.

Results on this model are given in Table 4.3. Pseudolikelihood performs particularly poorly on this model. Piecewise estimation performs much better, but worse than approximate training using BP.

Piecewise training is faster than loopy BP: in our implementation piecewise training used on average 3.5 hr, while loopy BP used 6.8 hr. To get these loopy BP results, however, we must carefully initialize the training procedure, as discussed in Section 3.2.2. For example, if instead we initialize the model to the uniform distribution, not only does loopy BP training take much longer, over 10 hours, but testing performance is much worse, because the convex optimization procedure has difficulty with noisier gradients. With uniform initialization, loopy BP does not converge for all training instances, especially at early iterations of training. Carefully initializing the model parameters seems to alleviate these issues, but this model-specific tweaking was unnecessary for piecewise training.

Model	Basic	Reweighted
Linear-chain	91.2	90.4
FCRF	88.1	86.4
Skip-chain (location)	87.7	75.5
Skip-chain (speaker)	75.4	69.2

Table 4.5. Comparison of basic piecewise training to reweighted piecewise bound with uniform μ.

4.3.4 Reweighted Piecewise Training

We also evaluate a reweighted piecewise training, a modification to the basic piecewise estimator discussed in Section 4.2, in which the pieces are weighted by a convex combination. The performance of reweighted piecewise training with uniform μ_R is presented in Table 4.5. In all cases, the reweighted piecewise method performs worse than the basic piecewise method. What seems be happening is that in each of these models, there are several hundred edges, so that the weight μ_R for each region is rather small, perhaps around 0.01. For each piece R, reweighted bound includes a term $A\left(\theta|_R/\mu_R\right)$. If μ_R is around 0.01, then this means that we multiply the log factor values by 100 before evaluating A. This multiplier is so extreme that the term $A\left(\theta|_R/\mu_R\right)$ is dominated by maximum-value weight in $\theta|_R$.

4.4 Three Views of Approximate Training Algorithms

In this section,[2] we explain the subgraph and BP viewpoints on local training algorithms. First, many local training algorithms are straightforwardly viewed as performing exact inference on a transformed graph that cuts the global dependencies in the model. For example, standard piecewise performs maximum-likelihood training in a *node-split graph* in which variables are duplicated so that each factor is in its own

[2]This section and the next originally appeared as the technical report Sutton and Minka [124], as explained in the Acknowledgments at the end of the chapter.

connected component. We refer to this viewpoint as the *neighborhood graph view* of a training algorithm.

Second, many local training algorithms can be interpreted as approximating $\log Z$ by the Bethe energy $\log Z_{\text{BP}}$ at a particular message setting. We call this the *pseudomarginal view*, because under this view, the estimated parameters are chosen to match the pseudomarginals to the empirical marginals. For any approximate partition function $\tilde{Z}$, the pseudomarginals are the derivatives $\partial \log \tilde{Z} / \partial \theta_{ak}$. To explain the terminology, suppose that the unary factors have the form $\Psi_i(y_i) = \exp\{\sum_{y_i'} \theta_{i,y_i'} \mathbf{1}_{\{y_i = y_i'\}}\}$. Then the derivative $\partial \log Z / \partial \theta_i(y_i)$ of the true partition function yields the marginal distribution, so the corresponding derivative of $\log \tilde{Z}$ is called a pseudomarginal.

As a third viewpoint, any approximate inference algorithm can be used to perform approximate ML training, by substituting the approximate beliefs for the exact marginals in the ML gradient. We call this the *belief view* of an approximate training algorithm. For example, this is the standard way of implementing approximate training using BP. Interestingly, although every approximate likelihood yields approximate gradients through the pseudomarginals, not all approximate gradients can themselves be obtained as the exact gradient of any single approximate objective function. An example of an approximate gradient that has no objective function is the one-step cutout method (Section 4.5.3). Recently, training methods that have a pseudomarginal interpretation—that is, those that can be described as numerically optimizing an objective function—have received much attention, but it is not clear if training methods that have a pseudomarginal interpretation should be preferred over ones that do not.

The pseudomarginal and belief viewpoints are distinct. To explain this, we need to make a distinction that is not always clear in the literature, between beliefs and pseudomarginals. By the *belief* of a node i, we mean its normalized product of messages, which is proportial to $q(y_i)$. By *pseudomarginal*, on the other hand, we mean the derivative of $\log \tilde{Z}$ with respect to θ_i. These quantities are distinct. For example, in standard piecewise, the pseudomarginal $\partial \log Z_{\text{PW}}/\partial\theta_i(y_i)$ equals $p_i(y_i)$, but the belief is proportional to $q(y_i) = \sum_{\mathbf{y}\backslash y_i} q(y_i) = 1$.

This point may be confusing for several reasons. First, when the messages m are a fixed point of BP, then the pseudomarginal always equals the belief. But this does not hold before convergence, and we shall be mainly concerned with intermediate message settings, before BP has converged. A second potential confusion arises because we define Z_{BP} using the dual Bethe energy [81] rather than the primal. In the primal Bethe energy [150], the pseudomarginal equals the belief at all message settings, but this is not true of the dual energy. We use the dual energy rather than the primal not only because it helps in interpreting local training algorithms, but also because at intermediate message settings it tends to be a better approximation to $\log Z$.

When calculating pseudomarginals $\partial \log \tilde{Z}/\partial\theta_i$, we must recognize that the message setting is often itself a function of θ. For example, suppose we stop BP after one iteration, that is, we take $\tilde{Z}(\theta) = Z_{\text{BP}}(\theta, m^{(1)}(\theta))$, where $m^{(1)}$ are the messages after one BP iteration. Then, because the message setting is clearly a function of θ, we need to take $\partial m^{(1)}/\partial\theta_i$ into account when computing the pseudomarginals of $\tilde{Z}$.

4.5 Extensions from BP Early Stopping

If piecewise training can be seen as running zero iterations of BP, it is natural to ask whether one or two iterations of BP might perform better. This leads to the algorithms that we explore in this section, namely *shared-unary piecewise* and *one-step cutout*. In this section, I assume that all factors are *weakly canonical* form, by which I mean that every variable i has a unary factor $\Psi_i(y_i, \mathbf{x}, \theta)$, and for every non-unary factor a and variable $i \in a$, the sum $\sum_{\mathbf{y}_a\backslash y_i} \Psi_a(y_a, \mathbf{x}, \theta)$ is uniform over y_i. For fixed $\mathbf{x}$, this transformation is always possible, and it means intuitively that none of the unary information is hidden within higher-way factors. Note that this is a weaker condition than the canonical form used in the Hammersley-Clifford theorem [7], in that we allow k-way factors ($k > 2$) to contain ($k - 1$)-way information; for example, a three-way factor may contain two-way information. I also assume that the unary factors are parameterized as

$$\Psi_i(y_i, \mathbf{x}, \theta) = \exp\{\theta_i(y_i)\}. \tag{4.27}$$

4.5.1 Shared-Unary Piecewise

One idea for improving the piecewise estimate of the unary gradient is to duplicate the unary factors over every non-unary piece that they neighbor. This yields a new approximate training method, that I call *shared-unary piecewise*. Recall that the standard piecewise Z_{PW} arises from Z_{BP} when all $\tilde{\Psi}_a = 1$. In shared-unary piecewise, rather than approximating the unary factors by $\tilde{\Psi}_i = 1$, we incorporate them exactly, that is, we take $\tilde{\Psi}_i(y_i) = \Psi_i(y_i, \mathbf{x}, \theta)$ for the unary factors, and $\tilde{\Psi}_a = 1$ otherwise. These messages are the result of one parallel BP iteration from uniform messages, so I call them *parallel BP(1) messages*. These messages yield the approximate partition function:

$$Z_{\text{PWU}} = \prod_{a \in NU} \left(\sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \prod_{i \in a} \Psi_i(y_i, \mathbf{x}, \theta) \right) \prod_i \left(\sum_{y_i} \Psi_i(y_i, \mathbf{x}, \theta) \right)^{2-d_i}, \tag{4.28}$$

where NU is the set of nonunary factors in the model. We can distribute terms in Z_{PWU} to yield a form similar to standard piecewise. First, we define a normalized version of the unary factors as

$$p_i(y_i) = \frac{\Psi_i(y_i, \mathbf{x}, \theta)}{\sum_{y'_i} \Psi_i(y'_i, \mathbf{x}, \theta)}. \tag{4.29}$$

Then we can distribute terms in (4.28) to obtain

Iterations	
0	$\tilde{\Psi}_a = 1$ for all a
1	$\tilde{\Psi}_a = 1$ for nonunary a; $\tilde{\Psi}_i = \Psi_i$ for variables i
2	$\tilde{\Psi}_a = \prod_{i \in a} m_{ai}^{(2)}(y_i)$, where $m_{ai}^{(2)}(y_i) = \sum_{y \backslash y_i} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \prod_{j \in N(a) \backslash i} \Psi_j(y_j, \mathbf{x}, \theta)$

Table 4.6. Message settings after zero, one, and two parallel iterations of BP. Recall that the nonunary factors are assumed to be weakly canonical.

$$Z_{\text{PWU}} = \prod_{a \in NU} \left(\sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \prod_{i \in N(a)} p_i(y_i) \right) \prod_i \sum_{y_i} \Psi_i(y_i, \mathbf{x}, \theta). \tag{4.30}$$

So shared-unary piecewise is the same as regular piecewise, except that we share normalized unary factors among all of the higher-way pieces that they neighbor. By normalizing the unary factors before spreading them across all the pieces, intuitively we avoid overcounting their sum.

4.5.2 Shared-Unary Pseudomarginals

In this section, we derive the pseudomarginals for shared-unary piecewise. Taking the derivative of $\log Z_{\text{PWU}}$ yields

$$\frac{\partial \log Z_{\text{PWU}}}{\partial \theta_i(y')} = \sum_{a \in NU(i)} \frac{\sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \left(\prod_{j \in N(a) \backslash i} p_j(y_j) \right) \cdot \frac{\partial p_i(y_i)}{\partial \theta_i(y')}}{\sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \left(\prod_{j \in N(a)} p_j(y_j) \right)} + p_i(y'), \tag{4.31}$$

where $NU(i)$ is set of all nonunary factors that neighbor variable i. Then substituting in

$$\frac{\partial p_i(y_i)}{\partial \theta_i(y')} = p(y_i)[1_{\{y'=y_i\}} - p_i(y')] \tag{4.32}$$

yields

Standard piecewise	
Neighborhood graph	node-split graph (Section 4.1.1)
Pseudomarginal view	Messages at zero iterations
Belief view	$b_a(\mathbf{y}_a) \propto \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta)$
Shared-unary piecewise	
Neighborhood graph	node-split graph with pieces Ψ_i for each variable i, and $\{\Psi_a\} \cup \{p_i \,|\, i \in a\}$ for each nonunary a
Pseudomarginal view	Messages after one parallel iteration
Belief view	Summation-hack approximation to BP beliefs after two parallel iterations.
Cutout (one-step)	
Neighborhood graph	node-split graph with pieces G_a for each factor a, where G_a defined as in (4.37)
Pseudomarginal view	none
Belief view	BP beliefs after two parallel iterations

Table 4.7. Viewpoints on local training algorithms discussed in this note. For each method, "Neighborhood graph" means the graph on which the method can be viewed as performing exact maximum likelihood training. "Pseudomarginal view" lists the message setting with which the method approximates $\log Z$ by $\log Z_{\text{BP}}$. "Belief view" gives the beliefs with which the method can be viewed as approximating the gradient of the true likelihood. For reference, the BP messages after zero, one, and two parallel iterations are given in Table 4.6.

$$\frac{\partial \log Z_{\text{PWU}}}{\partial \theta_i(y')} = p_i(y') + \left(\sum_{a \in NU(i)} \frac{\sum_{\mathbf{y}_a \setminus y'} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \left(\prod_{j \in N(a)} p_j(y_j) \right)}{\sum_{\mathbf{y}_a} \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \left(\prod_{j \in N(a)} p_j(y_j) \right)} - p_i(y') \right) \tag{4.33}$$

$$= p_i(y') \left[1 + \left(\sum_{a \in NU(i)} C_a m_{ai}^{(2)}(y') - 1 \right) \right], \tag{4.34}$$

where $m_{ai}^{(2)}$ are the unnormalized BP messages after two parallel updates. We introduce the notation C_a to represent the denominator in (4.33), which is not the normalizing constant of $m_{ai}^{(2)}$.

4.5.3 Cutout Method

The *cutout method* approximates the true gradient by performing exact inference on a subgraph. Each parameter is assigned a its own subgraph, but the subgraphs are allowed to overlap. Given a subgraph G_a for each factor a, we define the subgraph likelihood ℓ_a as the exact likelihood over the graph G_a. Let A be the set of factors in G_a and $F_A = \prod_{b \in A} \Psi_b$. Then the subgraph likelihood can be written

$$\ell_a(\theta_a) = \frac{F_A(\mathbf{y}_a, \mathbf{x}, \theta)}{\sum_{\mathbf{y}'_a} F_A(\mathbf{y}'_a, \mathbf{x}, \theta)} = \frac{F_A(\mathbf{y}_a, \mathbf{x}, \theta)}{Z_A}. \tag{4.35}$$

Then the parameter vector θ is selected to solve the system of equations $\partial \ell_a / \partial \theta_a = 0$ for all a. In general, there does not exist a single objective function $\ell(\theta)$ whose partial derivatives match all of the $\partial \ell_a / \partial \theta_a$, because the vector field defined by $\partial \ell_a / \partial \theta_a$ has nonzero curl. In two dimensions, the curl of a vector field $H(x_1, x_2) = [f_1(x_1, x_2) \; f_2(x_1, x_2)]$ is given by

$$\operatorname{curl} H = \frac{\partial f_2}{\partial x_1} - \frac{\partial f_1}{\partial x_2}. \tag{4.36}$$

It is a standard theorem of vector calculus that a piecewise continuous vector field over $\mathbb{R}^n$ is the gradient of a function if and only if it has zero curl. To see this, observe that if H has nonzero curl, it is the gradient of a function f only if $\partial f / \partial x_1 x_2 \neq \partial f / \partial x_2 x_1$, which is impossible. The cutout vector field has nonzero curl essentialy because each θ_a is used in many ℓ_b, but only one ℓ_a is used to compute its approximate gradient.

Here I focus on a special case of the cutout method, the *one-step cutout method.* In one-step cutout, we choose G_a to be all of the neighboring factors of f_a, plus their unaries. That is, G_a is a factor graph with factors

$$A = \{\Psi_b \,|\, \text{factor } \Psi_b \text{ is distance 2 or less from factor } \Psi_a\}. \tag{4.37}$$

(When counting distance between factors, we do not count variables, so that a path $a - i - b$ in the factor graph counts as one step.) In many situations, the cutout graph G_a is a tree, even when the original graph G is not, for example when G is a grid. If G_a is a tree, then we can compute $\partial \ell_a / \partial \theta_a$ exactly using two parallel iterations of BP on the original graph G. To see this, observe that because G_a is a tree of diameter 4, we can exactly compute Z_A by performing two parallel BP iterations on G_a. But the two-iteration messages on G_a are the same as the two-iteration messages on the original graph, which are given in Table 4.6.

The beliefs at the parallel BP(2) message setting are

$$b_a^{(2)}(\mathbf{y}_a) \propto \Psi_a(\mathbf{y}_a, \mathbf{x}, \theta) \prod_{i \in N(i)} m_{ai}^{(2)}(y_i). \tag{4.38}$$

So from the belief viewpoint, one-step cutout approximates the ML gradient by substituting the beliefs (4.38) for the marginal probabilities.

4.5.4 Shared Unary as an Approximation to Cutout

Shared-unary piecewise can be viewed as an approximation to the cutout method. A general way of approximating a product of terms is the "summation hack":

$$\prod_i \epsilon_i = \prod_i 1 + (\epsilon_i - 1) \approx 1 + \left(\sum_i \epsilon_i - 1 \right), \tag{4.39}$$

where the approximation arises from a first-order Taylor expansion around $\epsilon = 1$.

Applying the summation hack to the one-step cutout beliefs (4.38), we obtain

$$b_i^{(2)}(y_i) \propto \Psi_i(y_i, \mathbf{x}, \theta) \prod_{a \in NU(i)} C_a m_{ai}^{(2)}(y_i) \tag{4.40}$$

$$\approx \Psi_i(y_i, \mathbf{x}, \theta) \left[1 + \left(\sum_{a \in NU(i)} C_a m_{ai}^{(2)}(y_i) - 1 \right) \right], \tag{4.41}$$

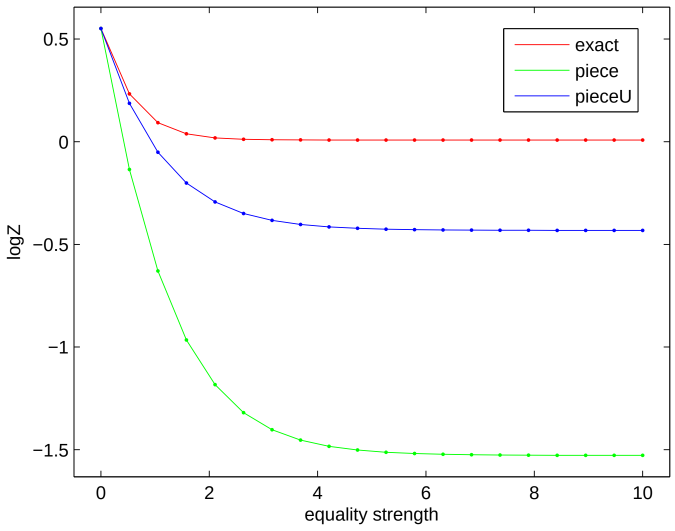
which are the same as the pseudomarginals (4.34) of Z_{PWU}, up to the proportionality constant of p_i. So we can view the shared-unary pseudomarginals either as the pseudomarginals after one BP iteration, or as an approximation to the beliefs after two iterations. This leads us to expect two sources of error in shared-unary piecewise: error may arise either from the summation hack, or because the model has long-distance interactions that cannot be propagated in two parallel BP iterations.

4.5.5 Simulations

These intuitions can be validated by simulation on a simple network. This data is generated from a three-node network of binary variables with pairwise factors

$$\Psi_a(\mathbf{y}_a) = \begin{pmatrix} 1 & e^{-s} \\ e^{-s} & 1 \end{pmatrix} \tag{4.42}$$

and unary factors $\Psi_i(y_i) = [1 \;\; e^{-u}]$. We transform the pairwise factors into a three-variable exclusive-or factor times a unary factor, so that from the perspective of the learning algorithm, all the factors are unary. We focus on how the approximations to $\log Z$ and its derivatives change as a function of the model parameters. This is useful to study because the log likelihood equals $\log Z$ plus a linear function of



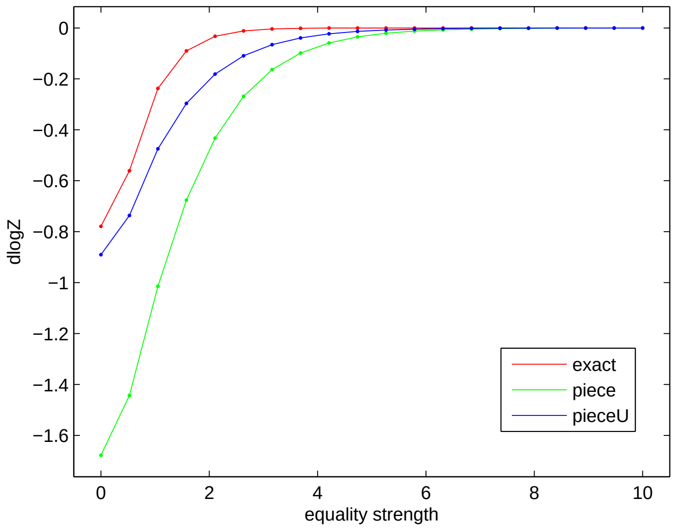
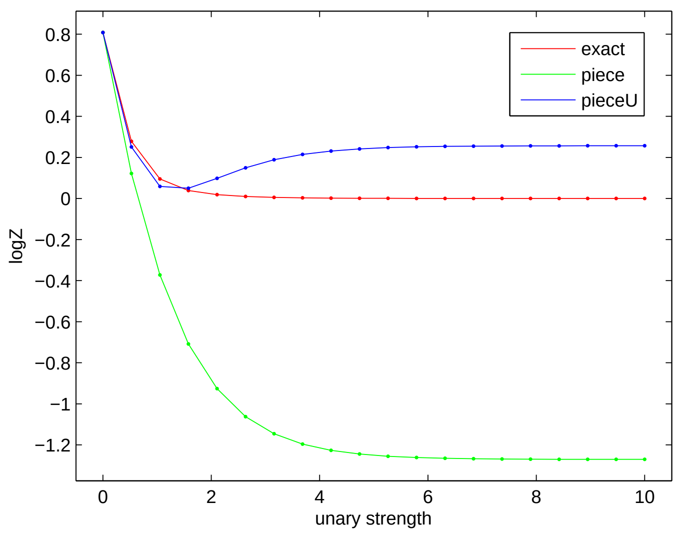


Figure 4.3. Comparison of piecewise and shared-unary piecewise approximations as a function of the equality strength s. Top, approximation to $\log Z$; bottom, approximation to its derivative.



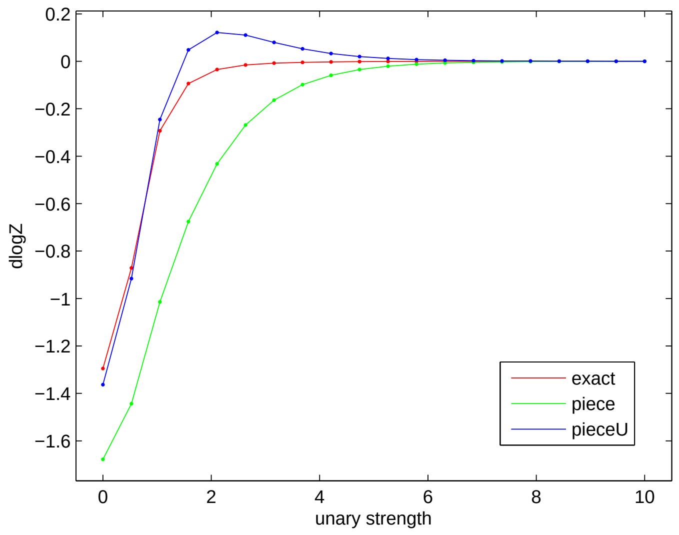


Figure 4.4. Approximation to $\log Z$ by piecewise and shared-unary piecewise as a function of the unary strength u. Top, approximation to $\log Z$; bottom, approximation to its derivative.

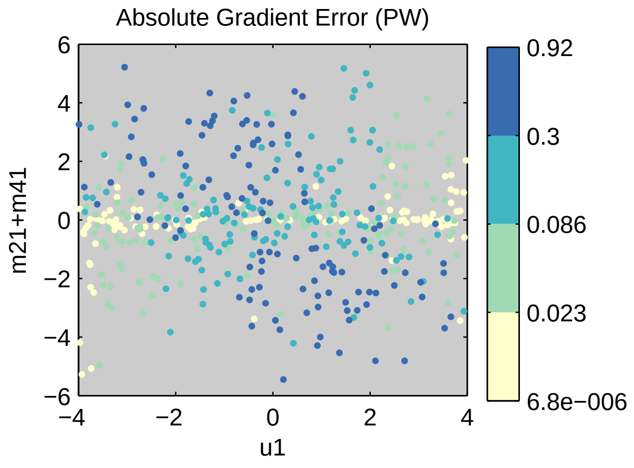


Figure 4.5. Absolute gradient error of standard piecewise as a function of the unary parameter and the sum of its incoming message strengths.

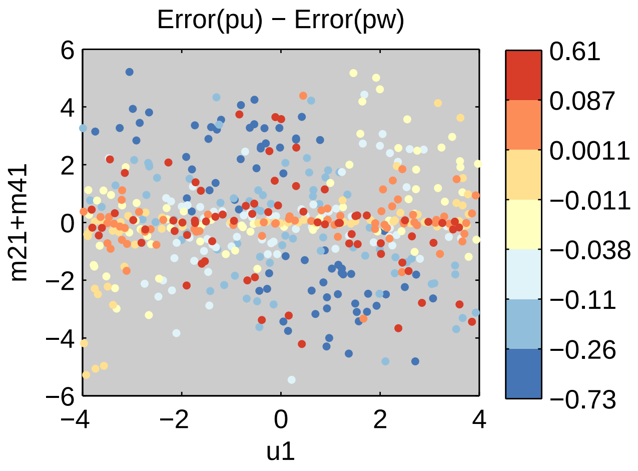


Figure 4.6. Difference in absolute gradient error between shared-unary piecewise and standard piecewise.

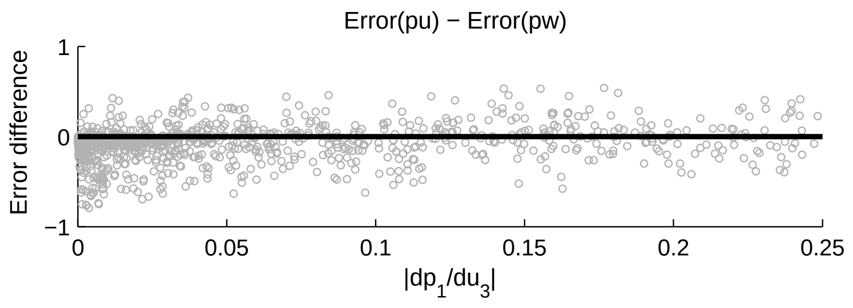


Figure 4.7. Difference in gradient error between shared-unary and standard piecewise. At left, when the magnitude of $\partial p(y_1)/\partial u_3$ is small, then shared-unary is superior. At right, where this derivative is large, shared-unary and standard piecewise are equivalent.

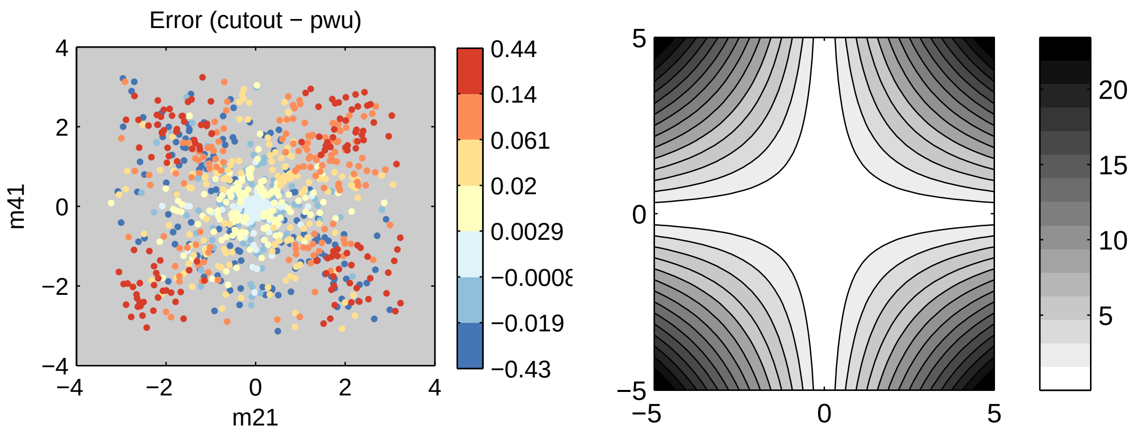


Figure 4.8. Difference in gradient error between cutout method and shared-unary piecewise as a function of the message strengths (left). At right, contours of the error of the “summation hack” Taylor expansion.

the parameters, so examining $\log Z$ alone gives insight into how the approximation performs for any data set [127].

First, we look at single-dimensional plots of the approximate gradients, in which all of the unary parameters are tied, and we vary either the unary strength u or the equality strength s. As we vary the equality strength, for a fixed, strong unary strength of $e^{-u} = 0.2$, then shared-unary piecewise provides a much better approximation to $\log Z$ as a function of the equality strength s (Figure 4.3). As s approaches 0, the pairwise factors drop out, so that both the piecewise approximations are exact. In both Figures 4.3 and 4.4, we subtract $\log Z(0)$ from $\log Z_{\text{PW}}$. Without that correction, Z_{PW} is an upper bound but a strong overestimate. Also, in both figures, the plotted derivative is the negative of the pseudomarginal, because of the parameterization we use.

When we vary the unary strengths, on the other hand, shared unary has less desirable behavior (Figure 4.4). This figure shows the approximations to $\log Z$ and its derivative for a fixed equality strength $e^{-s} = 0.2$. Of course, as u approaches 0, the unary factors drop out, so that shared unary becomes equivalent to standard piecewise. Elsewhere, however, we see that shared-unary piecewise is no longer convex, because of the per-node normalization, and in fact we see a large regime where Z_{PWU} is increasing while the true Z is decreasing. Consequently, the derivative of Z_{PWU} crosses zero at several points when the exact objective does not, which is undesirable in an approximation. In other words, the piecewise pseudomarginal is sometimes negative.

We can get a more precise sense of when the piecewise approximations break down by examining their approximation error as a function of the incoming messages from the rest of the network. To do this, we use a four-node Potts network of the form above, which is easier to interpret because there are no odd-length cycles. Also, it is helpful to leave the unary parameters untied, so that the model has five parameters

$[s, u_1, u_2, u_3, u_4]$. We generate models by sampling uniformly over all parameter values in the range $[-4, 4]$. We measure the error in the pseudomarginal of y_1 for both standard and shared-unary piecewise. We plot the error in the pseudomarginal as a function of the message strengths m_{21} and m_{41} of the incoming messages to y_1 from its neighbors y_2 and y_4. By message strength, we mean the log ratio of the message value at 0 over the message value at 1.

For standard piecewise (Figure 4.5), we see as expected that the approximation error is greatest when the incoming messages are both strong and in disagreement with the local unary parameter. For shared unary, on the other hand, we report the difference in gradient error between shared-unary and standard piecewise (Figure 4.6). First, shared unary improves greatly over standard piecewise in the areas where piecewise performs worst, that is, when the incoming messages disagree strongly with the local unary. But shared unary is not always better than standard piecewise, especially when the messages are weak. This may be surprising because shared-unary performs an extra iteration of BP. However, the BP view of shared unary suggests two possible sources of error. First, one BP iteration can actually be worse than zero iterations, if nearby potentials contradict stronger factors elsewhere in the network. Second, shared unary may be a bad approximation to the two-iteration BP beliefs because of the summation hack.

We can isolate these two sources of error. First, to see where one iteration of BP might actually hurt, we look at the derivative of $p(y_1)$, the exact marginal probability of variable y_1, taken with respect to u_3, the unary parameter opposite y_1 in the graph. If this derivative has large magnitude, then we expect that the parameter u_3 has a large impact on the marginal $p(y_1)$, so that one iteration can make the beliefs worse if y_2 and y_4 have an effect in the opposite direction as y_3. In Figure 4.7, we show the error difference between shared-unary and standard piecewise as a function of $|\partial p(y_1)/\partial u_3|$. As this argument predicts, when this derivative has large magnitude, then the information from u_3, which neither method considers, is most important, so that neither method dominates. When this derivative has small magnitude, then the local information is more important, so shared-unary piecewise dominates.

Second, we can examine the effects of the summation hack by plotting the difference in gradient error between shared unary and cutout. The summation hack is accurate near the axes, so if both messages are strong, we expect shared unary to have high error. In Figure 4.8 it can be seen that shared unary performs worse than one-step cutout at exactly the places where the summation hack predicts.

An interesting observation here is that the cutout method is itself not always better than shared-unary piecewise, even though cutout performs an extra iteration of BP. This happens in cases when u_3 has large magnitude, so neither method can do well. In these cases, it can happen that the summation hack error pushes the shared-unary marginal in the direction of the correct marginal, so that shared unary performs better.

In summary, we have seen that in many situtions, shared-unary piecewise provides a better approximation to $\log Z$ and its derivatives than standard piecewise. The occasions when standard piecewise performs better than shared unary occur when there is strong influence from outside the pieces, in which case neither piecewise method is probably advisable. We have also demonstrated in simulation two potential sources of error in shared-unary piecewise: strong iteractions from outside the piece, and the summation hack. A potentially serious drawback to shared-unary piecewise, however, it is not convex in the parameters. Indeed, an important limitation of these simulations is that we have looked only at error in the gradient, not error in the optimal parameter settings, so we cannot assess to what extent the loss of convexity makes it harder to find good parameters.

4.6 Related Work

Because the piecewise estimator is such an intuitively appealing method, it has been used in several scattered places in the literature, for tasks such as information extraction [147], collective classification [44], and computer vision [34]. In these papers, the piecewise method is reported as a successful heuristic for training large models, but its performance is not compared against other training methods. We are unaware of previous work systematically studying this procedure in its own right.

As mentioned earlier, the most closely related procedure that has been studied statistically is pseudolikelihood [8, 9]. The main difference is that piecewise training does not condition on neighboring nodes, but ignores them altogether during training. This is depicted schematically by the factor graphs in Figure 4.2. In pseudolikelihood, each locally-normalized term for a variable or edge in pseudolikelihood includes contributions from a number of factors that connect to the neighbors whose observed values are taken from labeled training data. All these factors are circled in the top section of Figure 4.2. In piecewise training, each factor becomes an independently, locally-normalized term in the objective function.

Also, in statistics there has been work on general families of surrogate likelihoods, called composite likelihoods, which are sums of marginal or conditional log likelihoods [63]. Such composite likelihoods are consistent and asymptotically normal under relatively general assumptions. An example of using a composite likelihood on structured models for natural-language data is Kakade et al. [51]. But these are designed for a different situation than ours, namely when the joint likelihoods are difficult to compute but marginal likelihoods are easier to work with. An example of this situation is the multivariate Gaussian. In our context, marginal likelihoods are difficult to compute, so composite likelihoods are not as useful. Piecewise estimation is not a type of composite likelihood, because in the likelihood of each piece, the contribution of the rest of the model is ignored, not marginalized out.

Independently, Choi et al. [18] present a node-splitting technique for upper bounds during inference, which is closely related to the technique in this chapter used for learning.

4.7 Conclusion

This chapter has presented piecewise training, an intuitively appealing procedure that separately trains factor subsets, called pieces, of a loopy graph. We show that this procedure can be justified as maximizing a loose bound on the log likelihood. On three real-world language tasks with different model structures, piecewise training outperforms several versions of pseudolikelihood, a traditional local training method. On two of the data sets, in fact, piecewise training is more accurate than global training using belief propagation.

Many properties of piecewise training remain to be explored. Our results indicate that in some situations piecewise training should replace pseudolikelihood as the local training method of choice. In particular, the experiments here all used conditional training, which make local training easier because of the large amount of information in the conditioning variables. In the data sets here, the local features, such as the word identity, provide almost. In generative training, there may be much less local information, making piecewise training much less effective. On the other hand, from the exponential family perspective, piecewise training does still match expected statistics of a subgraph to the empirical distribution, which still seems intuitively appealing. For this reason, it is hard to give a definitive characterization of when piecewise training is expected to work well or poorly.

A possible explanation for the performance of piecewise training is that it acts as a form of additional regularization, in that the objective function disfavors parameter settings that obtain good joint likelihood by using long-distance effects of weights.

For this reason, connections to generalization bound for feature selection, some of which take into account the amount of computation, may be interesting.

4.8 Acknowledgments

Although I happily acknowledge at the beginning of this thesis my debts to my friends and colleagues, those debts are particularly large in the case of this chapter, so here I describe them in more detail. I thank an anonymous reviewer, and independently Drew Bagnell and David Blei, for pointing out the simple proof of Proposition 4.1. An early draft of mine had only the more complicated proof sketched in Section 4.2. Also, I thank David Rosenberg for alerting me to a discrepancy that led to my finding an error in an earlier version of the experiments of this chapter. Finally, I thank Tom Minka for pointing out the connection between piecewise training and belief propagation. Sections 4.4 and 4.5, and particularly the ideas of shared-unary piecewise and one-step cutout, are the result of our collaboration when I was an intern at Microsoft Research Cambridge.

CHAPTER 5
PIECEWISE PSEUDOLIKELIHOOD

Piecewise training can be an effective training method when the model structure is intractable. If the variables have large cardinality, however, training can be computationally demanding even when the model structure is tractable. For example, consider a series of processing steps of a natural-language sentence [33, 125], which might begin with part-of-speech tagging, continue with more detailed syntactic processing, and finish with some kind of semantic analysis, such as relation extraction or semantic entailment. This series of steps might be modeled as a simple linear chain, but each variable has an enormous number of outcomes, such as the number of parses of a sentence. In such cases, even training using forward-backward is infeasible, because it is quadratic in the variable cardinality. Thus, we desire approximate training algorithms not only that are sub exponential in the model's treewidth, but also that scale well in the variable cardinality.

Pseudolikelihood (PL) [8] is a classical training method that addresses both of these issues, both because it requires no propagation and also because its running time is linear in the variable cardinality. Although in some situations pseudolikelihood can be very effective [90, 134], in other applications, its accuracy can be poor.

An alternative that has been employed occasionally throughout the literature is to divide the factors in the model into a set of *pieces*, and train each piece separately, in its own graphical model. In Chapter 4, I presented this *piecewise estimation* method, finding that it performs well when the local features are highly informative, as can be true in a lexicalized NLP model with thousands of features. As we saw, piecewise performs better than pseudolikelihood on certain types of data, sometimes by a very large amount. So piecewise training can have good accuracy, however, unlike pseudolikelihood it does not scale well in the variable cardinality.

In this chapter, I introduce and analyze a hybrid method, called *piecewise pseudolikelihood* (PWPL), that combines the advantages of both approaches. Essentially, while pseudolikelihood conditions each variable on all of its neighbors, PWPL conditions only on those neighbors within the same piece of the model, for example, that share the same factor. This is illustrated in Figure 5.2. Remarkably, although PWPL has the same computational complexity as pseudolikelihood, on real-world NLP data, its accuracy is significantly better. In other words, in testing accuracy PWPL behaves more like piecewise than like pseudolikelihood. The training speed-up of PWPL can be significant even in linear-chain CRFs, because forward-backward training is quadratic in the variable cardinality.

The chapter proceeds as follows. In Section 5.2.1, I describe PWPL in terms of the node-split graph, which was presented previously in Section 4.1.1. This viewpoint allows us to show that under certain conditions, PWPL converges to the piecewise solution in the asymptotic limit of infinite data (Section 5.2.2). In addition, it provides some insight into when PWPL may be expected to do well and to do poorly, an insight that we verify on synthetic data (Section 5.3.1). Finally, I evaluate PWPL on several real-world NLP data sets (Section 5.3.2), finding that it performs often comparably to piecewise training and to maximum likelihood, and on all of our data sets PWPL has higher accuracy than pseudolikelihood. Furthermore, PWPL can be as much as ten times faster than batch CRF training.

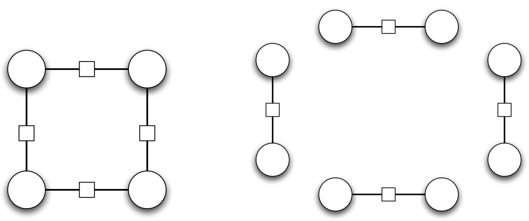


Figure 5.1. Example of node splitting. Left is the original model, right is the version trained by piecewise. In this example, there are no unary factors.

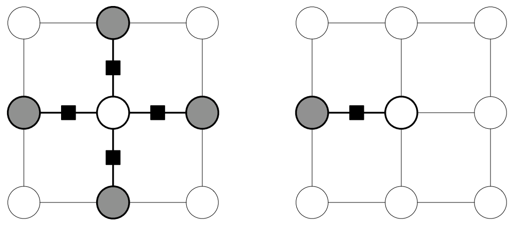


Figure 5.2. Illustration of the difference between piecewise pseudolikelihood (PWPL) and standard pseudolikelihood. In standard PL, at left, the local term for a variable y_s is conditioned on its entire Markov blanket. In PWPL, at right, each local term conditions only on the neighbors within a single factor.

5.1 Piecewise Training

5.1.1 Background

In this section, I make a few observations about pseudolikelihood and piecewise training that will be useful. As in the previous chapters, we are interested in estimating the parameters of a conditional random field $p(\mathbf{y}|\mathbf{x})$ of the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{a=1}^{A} \Psi_a(\mathbf{y}_a, \mathbf{x}_a), \tag{5.1}$$

where the factors have the exponential form

$$\Psi_a(\mathbf{y}_a, \mathbf{x}_a) = \exp\{\sum_{k=1}^{K} \theta_{ak} f_a(\mathbf{y}_a, \mathbf{x}_a)\}, \tag{5.2}$$

Recall from Chapter 4 that pseudolikelihood is a classical approximation that simultaneously classifies each node given its neighbors in the graph. For a variable s, let $N(s)$ be the set of all of its neighbors, not including s itself. Then the pseudolikelihood is defined as

$$\ell_{\text{PL}}(\Lambda) = \sum_s \log p(y_s | y_{N(s)}, \mathbf{x}),$$

where the conditional distributions are

$$p(y_s | y_{N(s)}, \mathbf{x}) = \frac{\prod_{a \ni i} \Psi_a(y_s, y_{N(s)}, \mathbf{x}_a)}{\sum_{y'_s} \prod_{a \ni s} \Psi_a(y'_s, y_{N(s)}, \mathbf{x}_a)}. \tag{5.3}$$

where $a \ni s$ means the set of all factors a that depend on the variable s. In other words, this is a sum of conditional log likelihoods, where for each variable we condition on the true values of its neighbors in the training data.

It is a well-known result that if the model family includes the true distribution, then pseudolikelihood converges to the true parameter setting in the limit of infinite data [41, 48]. One way to see this is that pseudolikelihood is attempting to match all of model conditional distributions to the data. If it succeeds in matching them all exactly, then a Gibbs sampler run on the model distribution will have the same invariant distribution as a Gibbs sampler run on the true data distribution.

In the previous chapter, I also presented piecewise training, based on the intuition that if each factor $\Psi(\mathbf{y}_a, \mathbf{x}_a)$ can on its own accurately predict $\mathbf{y}_a$ from $\mathbf{x}_a$, then the prediction of the global factor graph will also be accurate. Formally, piecewise training maximizes the objective function

$$\ell_{\text{PW}}(\Lambda) = \sum_a \log \frac{\Psi_a(\mathbf{y}_a, \mathbf{x}_a)}{\sum_{\mathbf{y}'_a} \Psi_a(\mathbf{y}'_a, \mathbf{x}_a)}. \tag{5.4}$$

The explanation for the name piecewise is that each term in (5.4) corresponds to a "piece" of the graph, in this case a single factor, and that term would be the exact likelihood of the piece if the rest of the graph were omitted. An important observation is that the denominator of (5.3) sums over assignments to a single variable, whereas the denominator of (5.4) sums over assignments to an entire factor, which may be a much larger set. This is why pseudolikelihood can be much more computationally efficient than piecewise when the variable cardinality is large.

5.2 Piecewise Pseudolikelihood

In this section, I define piecewise pseudolikelihood (Section 5.2.1), and describe its asymptotic behavior using well-known results about pseudolikelihood (Section 5.2.2).

5.2.1 Definition

The main motivation of piecewise training is computational efficiency, but in fact piecewise does not always provide a large gain in training time over other approximate methods. In particular, the time required to evaluate the piecewise likelihood at one parameter setting is the same as is required to run one iteration of belief propagation (BP). More precisely, piecewise training uses $O(m^K)$ time, where m is the maximum number of assignments to a single variable y_s and K is the size of the largest factor. Belief propagation also uses $O(m^K)$ time per iteration; thus, the only computational savings over BP is a factor of the number of BP iterations required. In tree-structured graphs, piecewise training is no more efficient than forward-backward.

To address this problem, we propose piecewise pseudolikelihood. Piecewise pseudolikelihood (PWPL) is defined as:

$$\ell_{\text{PWPL}}(\Theta; \mathbf{x}, \mathbf{y}) = \sum_{a} \sum_{s \in a} \log p_{\text{LCL}}(y_s | \mathbf{y}_{a \backslash s}, \mathbf{x}, \theta_a), \tag{5.5}$$

where $(\mathbf{x}, \mathbf{y})$ are an observed data point, the index a ranges over all factors in the model, the set $a \backslash s$ means all of the variables in the domain of factor a except for s, and p_{LCL} is a locally-normalized score similar to a conditional probability and defined below.

In other words, the piecewise pseudolikelihood is a sum of local conditional log-probabilities. Each variable s participates as the domain of a conditional once for each factor that it neighbors. As in piecewise training, the local conditional probabilities p_{LCL} are not the true probabilities according to the model, but are a quantity computed locally from a single piece (in this case, a single factor). The local probabilities p_{LCL} are defined as

$$p_{\text{LCL}}(y_s|\mathbf{y}_{a\backslash s}, \mathbf{x}, \theta_a) = \frac{\Psi_a(y_s, \mathbf{y}_{a\backslash s}, \mathbf{x}_a)}{\sum_{y'_s} \Psi_a(y'_s, \mathbf{y}_{a\backslash s}, \mathbf{x}_a)}. \tag{5.6}$$

Then given a data set $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, we select the parameter setting that maximizes

$$O_{\text{PWPL}}(\theta; D) = \sum_i \ell_{\text{PWPL}}(\theta; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \sum_a \frac{\|\theta_a\|^2}{2\sigma^2}, \tag{5.7}$$

where the second term is a Gaussian prior on the parameters to reduce overfitting. The piecewise pseudolikelihood is convex as a function of θ, and so its maximum can be found by standard techniques. In the experiments below, we use limited-memory BFGS [89].

For simplicity, we have presented PWPL for the case in which each piece contains exactly one factor. If larger pieces are desired, then simply take the summation over a in (5.5) to be over pieces rather than over factors, and generalize the definition of p_{LCL} appropriately.

Compared to standard piecewise, the main advantage of PWPL is that training requires only $O(m)$ time rather than $O(m^K)$. Compared to pseudolikelihood, the difference is that whereas in pseudolikelihood each local term conditions on the entire Markov blanket, in PWPL each local term conditions only on a variable's neighbors within a single factor. For this reason, the local terms in PWPL are not true conditional distributions according to the model. The difference between PWPL and pseudolikelihood is illustrated in Figure 5.2. In the next section, we discuss why in some situations this can cause PWPL to have better accuracy than pseudolikelihood.

5.2.2 Analysis

PWPL can be readily understood from the node-split viewpoint. In particular, the piecewise pseudolikelihood is simply the standard pseudolikelihood applied to the node-split graph. In this section, we use the asymptotic consistency of standard pseudolikelihood to gain insight into the performance of PWPL.

Let $p^*(\mathbf{y})$ be the true distribution of the data, after the node splitting transformation has been applied. Both PWPL and standard piecewise cannot distinguish this distribution from the distribution p_{NS} on the node-split graph that is defined by the product of marginals

$$p_{\text{NS}}(\mathbf{y}) = \prod_{a \in G'} p^*(y_a), \tag{5.8}$$

where G' is the node-split graph, and $p^*(y_a)$ is the marginal distribution of the variables in factor a according to the true distribution. By that we mean that the piecewise likelihood of any parameter setting θ when the data distribution is exactly the true distribution p^* is equal to the piecewise likelihood of θ when the data distribution equals the distribution p_{NS}, and similarly for PWPL.

So equivalently, we suppose that we are given an infinite data set drawn from the distribution p_{NS}. Now, the standard consistency result for pseudolikelihood is that if the model class contains the generating distribution, then the pseudolikelihood estimate converges asymptotically to the true distribution. In this setting, that implies the following statement. If the model family defined by G' contains p_{NS}, then piecewise pseudolikelihood converges in the limit to the same parameter setting as standard piecewise.

Because this is an asymptotic statement, it provides no guarantee about how PWPL will perform on real data. Even so, it has several interesting consequences that provide insight into the method. First, it may impact what sort of model is conducive to PWPL. For example, consider a Potts model with unary factors $\Psi(y_s) = [1 \quad e^{\theta_s}]^\top$ for each variable s, and pairwise factors

$$\Psi(y_s, y_t) = \begin{pmatrix} e^{\theta_{st}} & 1 \\ 1 & 1. \end{pmatrix}, \tag{5.9}$$

for each edge (s, t), so that the model parameters are $\{\theta_s\} \cup \{\theta_{st}\}$. Then the above condition for PWPL to converge in the infinite data limit will never be satisfied, because the pairwise piece cannot represent the marginal distribution of its variables. In this case, PWPL may be a bad choice, or it may be useful to consider pieces that contain more than one factor. In particular, shared-unary piecewise (see Section 4.5.1) may be appropriate.

Second, this analysis provides intuition about the differences between piecewise pseudolikelihood and standard pseudolikelihood. For each variable s with neighborhood $N(s)$, standard pseudolikelihood approximates the model marginal $p(y_{N(s)})$ over the neighborhood by the empirical marginal $\tilde{p}(y_{N(s)})$. We expect this approximation to work well when the model is a good fit, and the data is ample.

In PWPL, we perform the node-splitting transformation on the graph prior to maximizing the pseudolikelihood. The effect of this is to reduce each variable's neighborhood size, that is, the cardinality of $N(s)$.

This has two potential advantages. First, because the neighborhood size is small, PWPL may converge to piecewise faster than pseudolikelihood converges to the exact solution. Of course, the exact solution should be better than piecewise, so whether to prefer standard PL or piecewise PL depends on precisely how much faster the convergence is. Second, the node-split model may be able to exactly model the marginal of its neighborhood in cases where the original graph may not be able to model its larger neighborhood. Because the neighborhood is smaller, the pseudolikelihood con-

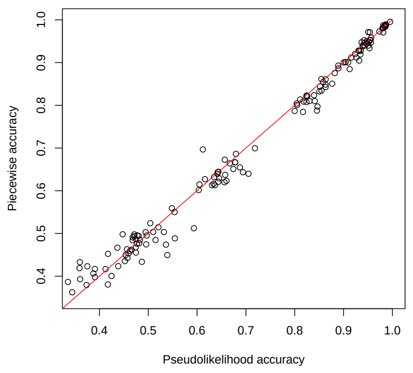


Figure 5.3. Comparison of piecewise to pseudolikelihood on synthetic data. Pseudolikelihood has slightly better accuracy on training instances than piecewise. (Piecewise and PWPL perform exactly the same; this is not shown.)

vergence condition may hold in the node-split model when it does not in the original model. In other words, standard pseudolikelihood requires that the original model is a good fit to the full distribution. In contrast, we expect piecewise pseudolikelihood to be a good approximation to piecewise when each individual piece fits the empirical distribution well. The performance of piecewise pseudolikelihood need not require the node-split model to represent the distribution across pieces.

Finally, this analysis suggests that we might expect piecewise pseudolikelihood to perform poorly in two regimes: First, if so much data is available that pseudolikelihood has asymptotically converged, then it makes sense to use pseudolikelihood rather than piecewise pseudolikelihood. Second, if features of the local factors cannot fit the training data well, then we expect the node-split model to fit the data quite poorly, and piecewise pseudolikelihood cannot possibly do well.

5.3 Experiments

5.3.1 Synthetic Data

In the previous section, we argued intuitively that PWPL may perform better on small data sets, and pseudolikelihood on larger ones. In this section we verify this in-

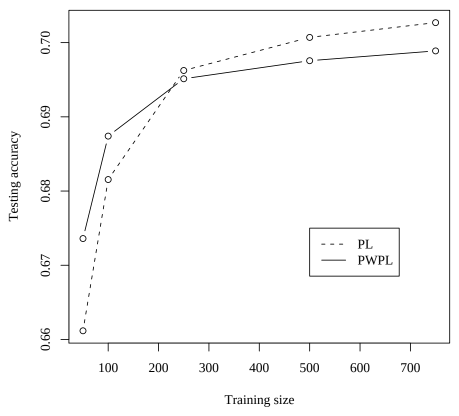


Figure 5.4. Learning curves for PWPL and pseudolikelihood. For smaller amounts of training data PWPL performs better than pseudolikelihood, but for larger data sets, the situation is reversed.

tuition in experiments on synthetic data. The general setup is replicated from Lafferty et al. We generate data from a second-order HMM with transition probabilities

$$p_\alpha(y_t|y_{t-1}, y_{t-2}) = \alpha p_2(y_t|y_{t-1}, y_{t-2}) + (1-\alpha)p_1(y_t|y_{t-1}) \tag{5.10}$$

and emission probabilities

$$p_\alpha(x_t|y_t, x_{t-1}) = \alpha p_2(x_t|y_t, x_{t-1}) + (1-\alpha)p_1(x_t|y_t). \tag{5.11}$$

Thus, for $\alpha = 0$, the generating distribution p_α is a first-order HMM, and for $\alpha = 1$, it is an autoregressive second-order HMM. We compare different approximate methods for training a first-order CRF. Therefore higher values of α make the learning problem more difficult, because the model family does not contain second-order dependencies. We use five states and 26 possible observation values. For each setting of α, we sample 25 different generating distributions. From each generating distribution we sample 1,000 training instances of length 25, and 1,000 testing instances. We use $\alpha \in \{0, 0.1, 0.25, 0.5, 0.75, 1.0\}$, for 150 synthetic generating models in all.

	ML	PL	PW	PWPL
POS				
Accuracy	94.4	94.4	94.2	94.4
Time (s)	33846	6705	23537	**3911**
Chunking				
Chunk F1	91.4	90.3	91.7	91.4
Time (s)	24288	1534	5708	**766**
Named-entity				
Chunk F1	90.5	85.1	90.5	90.3
Time (s)	52396	8651	6311	**4780**

Table 5.1. Comparison of piecewise pseudolikelihood to standard piecewise and to pseudolikelihood on real-world NLP tasks. Piecewise pseudolikelihood is in all cases comparable to piecewise, and on two of the data sets superior to pseudolikelihood.

First, we find that piecewise pseudolikelihood performs almost identically to standard piecewise training. Averaged over the 150 data sets, the mean difference in testing error between piecewise pseudolikelihood and piecewise is 0.002, and the correlation is 0.999.

Second, we compare piecewise to traditional pseudolikelihood. On this data, pseudolikelihood performs slightly better overall, but the difference is not statistically

	BP	PL	PW	PWPL
Start-Time	96.5	82.2	97.1	94.1
End-Time	95.9	73.4	96.5	90.4
Location	85.8	73.0	88.1	85.3
Speaker	74.5	27.9	72.7	65.0

Table 5.2. F1 performance of PWPL, piecewise, and pseudolikelihood on information extraction from seminar announcements. Both standard piecewise and piecewise pseudolikelihood outperform pseudolikelihood.

significant (paired t-test; $p > 0.1$). However, when we examine the accuracy as a function of training set size (Figure 5.4), we notice an interesting two-regime behavior. Both PWPL and pseudolikelihood seem to be converging to a limit, and the eventual pseudolikelihood limit is higher than PWPL, but PWPL converges to its limit faster. This is exactly the behavior intuitively predicted by the argument in Section 5.2.2: that PWPL can converge to the piecewise solution in less training data than pseudolikelihood to its (potentially better) solution.

Of course, the training set sizes considered in Figure 5.4 are fairly small, but this is exactly the case we are interested in, because on natural language tasks, even when hundreds of thousands of words of labeled data are available, this is still a small amount of data compared to the number of useful features.

5.3.2 Real-World Data

Now, we evaluate piecewise pseudolikelihood on four real-world NLP tasks: part-of-speech tagging, named-entity recognition, noun-phrase chunking, and information extraction.

For *part-of-speech tagging (POS)*, we report results on the WSJ Penn Treebank data set. Results are averaged over five different random subsets of 1911 sentences, sampled from Sections 0–18 of the Treebank. Results are reported from the standard development set of Sections 19–21 of the Treebank. We use a first-order linear chain CRF. There are 45 part-of-speech labels.

For the task of *noun-phrase chunking (chunking)*, we use a loopy model, the *factorial CRF* introduced in Section 3.1. As in that section, we consider here the task of jointly predicting part-of-speech tags and segmenting noun phrases in newswire text. Thus, the FCRF we use has a two-level grid structure. We report results here on subsets of 223 training sentences, and the standard test set of 2012 sentences. Results are averaged over 5 different random subsets. There are 45 different POS labels, and the three NP labels. We use the same features and experimental setup as previous work [119]. We report joint accuracy on (NP, POS) pairs; other evaluation metrics show similar trends.

In *named-entity recognition*, the task is to find proper nouns in text. We use the CoNLL 2003 data set, consisting of 14,987 newswire sentences annotated with names of people, organizations, locations, and miscellaneous entities. We test on the standard development set of 3,466 sentences. Evaluation is done using precision and recall on the extracted chunks, and we report $F_1 = 2PR/P+R$. We use a linear-chain CRF, whose features are described in Table 4.4.

Finally, for the task of *information extraction*, we consider a model with many irregular loops, which is the skip chain model introduced in Section 3.2. As i that section, the task is to extract information about seminars from email announcements from a standard data set [35]. We use the same features and test/training split as the previous work. The data is labeled with four fields—START-TIME, END-TIME, LOCATION, and SPEAKER—and we report token-level F1 on each field separately.

For all the data sets, we compare to pseudolikelihood, piecewise training, and conditional maximum likelihood with belief propagation. All of these objective functions are maximized using limited-memory BFGS. We use a Gaussian prior with variance $\sigma^2 = 10$.

Stochastic gradient techniques, such as stochastic meta-descent [110], would be likely to converge faster than the baselines we report here, because all our current results use batch optimization. However, stochastic gradient can be used with PWPL just as with standard maximum likelihood. Thus, although the training time of our baseline could likely be improved considerably, the same is true of our new approach, so that our comparison is fair.

5.3.3 Results

For the first three tasks—part-of-speech tagging, chunking, and NER—piecewise pseudolikelihood and standard piecewise training have equivalent accuracy both to each other and to maximum likelihood (Table 5.1). Despite this, piecewise pseudolikelihood is much more efficient than standard piecewise (Table 5.1). On the named-entity data, which has the fewest labels, PWPL uses 75% of the time of standard piecewise, a modest improvement. On the data sets with more labels, the difference is more dramatic: on the POS data, PWPL uses 16% of the time of piecewise and on the chunking data, PWPL needs only 13%. Similarly, PWPL is also between is 5 to 10 times faster than maximum likelihood.

The training times of the baseline methods may appear relatively modest. If so, this is because for both the chunking and POS data sets, we use relatively small subsets of the full training data, to make running this comparison more convenient. This makes the absolute difference in training time even more meaningful than it may appear at first. Also, it may appear from Table 5.1 that PWPL is faster than standard pseudolikelihood, but the apparent difference is due to low-level inefficiencies in our implementation. In fact the two algorithms have similar complexity.

On the skip chain data (Table 5.2), standard piecewise performs worse than exact training using BP, and piecewise pseudolikelihood performs worse than standard piecewise. Both piecewise methods, however, perform better than pseudolikelihood.

As predicted in Section 5.2.2, pseudolikelihood is indeed a better approximation on the node-split graph. In Table 5.1, PL performs much worse than ML, but PWPL performs only slightly worse than PW. In Table 5.2, the difference between PWPL and PW is larger, but still less than the difference between PL and ML.

5.4 Discussion and Related Work

Piecewise training and piecewise pseudolikelihood can both be considered types of *local training* methods, that avoid propagation throughout the graph. Such training methods have recently been the subject of much interest [1, 94, 134]. Of course, the local training method most closely connected to the current work is pseudolikelihood itself. We are unaware of previous variants of pseudolikelihood that condition on less than the full Markov blanket.

An interesting connection exists between piecewise pseudolikelihood and maximum entropy Markov models (MEMMs) [72, 100]. In a linear chain with variables $y_1 \ldots y_T$, we can rewrite the piecewise pseudolikelihood as

$$\ell_{\text{PWPL}}(\theta) = \sum_{t=1}^{T} \log p_{\text{LCL}}(y_t|y_{t-1}, \mathbf{x}) p_{\text{LCL}}(y_{t-1}|y_t, \mathbf{x}). \tag{5.12}$$

The first part of (5.12) is exactly the likelihood for an MEMM, and the second part is the likelihood of a backward MEMM. Interestingly, MEMMs crucially depend on normalizing the factors at both training and test time. To include local normalization at training time but not test time performs very poorly. But by adding the backward terms, in PWPL we are able to drop normalization at test time, and therefore PWPL does not suffer from label bias.

The current work also has an interesting connection to search-based learning methods [28]. Such methods learn a model to predict the next state of a local search procedure from a current state. Typically, training is viewed as classification, where the correct next states are positive examples, and alternative next states are negative examples. One view of the current work is that it incorporates backward training examples, that attempt to predict the *previous* search state given the current state.

Finally, stochastic gradient methods, which make gradient steps based on subsets of the data, have recently been shown to converge significantly faster for CRF training than batch methods, which evaluate the gradient of the entire data set before updating the parameters [136]. Stochastic gradient methods are currently the method of choice for training linear-chain CRFs, especially when the data set is large and redundant. However, as mentioned above, stochastic gradient methods can also be applied to piecewise pseudolikelihood. Also, in some cases, such as in relational learning problems, the data are not iid, and the model includes explicit dependencies between the training instances. For such a model, it is unclear how to apply stochastic gradient, but piecewise pseudolikelihood may still be useful. Finally, stochastic gradient methods do not address cases in which the variables have large cardinality, or when the graphical structure of a single training instance is intractable.

5.5 Summary

This chapter has presented piecewise pseudolikelihood (PWPL), a local training method that is especially attractive when the variables in the model have large cardinality. Because PWPL conditions on fewer variables, it can have better accuracy than standard pseudolikelihood, and is dramatically more efficient than standard piecewise, requiring as little as 13% of the training time.

CHAPTER 6

BELIEF PROPAGATION

Previously we have seen that many local training methods can be interpreted as approximate training using BP with early stopping. This raises the question of whether early stopping after larger numbers of iterations is generally useful. But we have also seen that early stopping interacts poorly with second-order optimization algorithms (Section 3.1.6.2), which are particularly useful when there are a large number of parameters. In this chapter, I try to make early stopping more useful by exploring the schedules used to prioritize messages. The hope is that early stopping may be most effective if we can get as much work as possible out of the messages that we have time to send.

Many popular approximate inference methods, such as belief propagation, its generalizations EP [77] and GBP [148], and structured mean-field methods [50], consist of a set of equations which are iterated to find a fixed point. The fixed-point updates are not usually guaranteed to converge. The schedule for propagating the updates can make a crucial difference both to how long the updates take to converge, and even whether they converge at all. Recently, *dynamic schedules*—in which the message values during inference are used to determine which update to perform next—have been shown to converge much faster on hard networks than static schedules [30]. In this chapter, I explore dynamic schedules both for inference and for learning.

First, I propose a new dynamic schedule the inference problem, which I call *residual BP with lookahead zero (RBP0L)* (Section 6.1). The idea behind the new schedule is to compute each message's priority cheaply, by the sum of how much its antecedents have changed. It can be shown, using arguments of Ihler et al. [49], that this priority is an upper bound on how much the message would change if the update were to be computed. On a natural-language data set, our schedule is more computationally efficient than the schedule presented previously by Elidan et al. [30].

Second, I present a first step toward the goal of using dynamic schedules to perform inference and learning in the same system of equations (Section 6.2). Combining both problems into a single system potentially allows for very flexible scheduling. The idea is that certain areas of parameter space should be easier for approximate inference than others, so we should run BP for longer in areas that are more difficult. Furthermore, inference in certain parts of the model may converge faster than others, so we should focus parameter updates on the parts that have not yet converged. On synthetic data, this yields a significant decrease in training time.

6.1 Dynamic Schedules for Inference

In this section, I introduce an efficient dynamic schedule for BP message updates. Previously, Elidan et al. proposed a schedule that propagates the message whose value has changed the most. I call this schedule *residual BP with lookahead one (RBP1L)*. Although this schedule was shown to be often more effective than static schedules, it has the difficulty that it determines a message's priority by actually computing it, which means that many message updates are "wasted", that is, they are computed solely for the purpose of computing their priority, and are never actually performed. A significant fraction of messages computed by RBP1L are wasted in this way. The main idea is that rather than *computing* the residual of each pending message update, it is far more efficient to *approximate* it. Recent work [49] has examined how a message error can be estimated as a function of its incoming errors. In our situation, the error arises because the incoming messages have been recomputed. The arguments from Ihler et al. apply also to the message residual, which leads to effective method for estimating the residual of a message, and to a dynamic schedule that is dramatically more efficient than RBP1L.

In this section, I first describe how the message residual can be upper-bounded by the residuals of its incoming messages (Section 6.1.2). I also describe a method for estimating the message residual when the factors themselves change (for example, from parameter updates), which leads to an intuitive method for initializing the residual estimates. Then I introduce a novel message schedule, called *residual BP with lookahead zero (RBP0L)* (Section 6.1.3). On several synthetic and real-world data sets, we show that RBP0L is as much as five times faster than RBP1L but still finds the same solution (Section 6.1.4). Finally, I examine how to what extent the distance that a message changes in a single update predicts its distance to its final converged value (Section 6.1.4.3). I measure distance in several different ways, including the dynamic range of the error and the Bethe energy. Surprisingly, the difference in Bethe energy has almost no predictive value for whether a message update is nearing convergence.

6.1.1 Background

In this section, I focus on generative models, so that we have a distribution $p(\mathbf{y})$ factorize according to an undirected factor graph G with factors $\{\Psi_a(\mathbf{y}_a)\}_{a=1}^{A}$. This choice is simply to lighten notation, and the inference algorithms of this section apply readily to conditional models as well. I use the indices a and b to denote factors of G, and the indices s and t to denote variables. By $\{s \in a\}$ I mean the set of all variables s in the domain of the factor Ψ_a, and conversely by $\{b \ni s\}$, I mean the set of all factors Ψ_b that have variable s in their domain.

Recall from Section 2.1.4 that belief propagation updates are given by

$$\begin{aligned} m_{ai}^{(k+1)}(y_s) &\leftarrow \kappa \sum_{\mathbf{y}_a \setminus y_s} \Psi_a(\mathbf{y}_a) \prod_{\{t \in a\} \setminus i} m_{ja}^{(k)}(y_t) \\ m_{ia}^{(k+1)}(y_s) &\leftarrow \kappa \prod_{\{b \ni t\} \setminus a} m_{bi}^{(k)}(y_s), \end{aligned} \tag{6.1}$$

which are iterated until a fixed point is reached. In the above, κ is a normalization constant to ensure the message sums to 1. The initial messages $m^{(0)}$ are set to some arbitrary value, typically a uniform distribution.

These message updates can be written in a generic fashion as

$$m_{cd}^{(k+1)}(y_{cd}) \leftarrow \kappa \sum_{y_c \backslash y_{cd}} \Psi_a(y_c) \prod_{\{b \in N(c)\} \backslash d} m_{bc}^{(k)}(y_c), \tag{6.2}$$

where c and d may be either factors or variables, as long as they are neighbors, $N(c)$ means the set of neighbors of c, and $\Psi_a(y_c)$ is understood to be the identity if c is a variable. This notation abstracts over whether a message is being sent from a factor or from a variable, which is convenient for describing message schedules.

In general, these updates may have multiple fixed points, and they are not guaranteed to converge. Convergent methods for optimizing the Bethe energy have been developed [145, 151], but they are not used in practice both because they tend to be slower than iterating the messages (6.1), and because when the BP updates do not converge, it has been observed that the Bethe approximation is bad anyway.

Now we describe in more detail how the iterations are actually performed in a BP implementation. This level of detail will prove useful in the next section for understanding the behavior of dynamic BP schedules. A vector $\mathbf{m} = \{m_{cd}\}$ is maintained of all the messages, which is initialized to uniform. Then until the messages are converged, we iterate: A message m_{cd} is selected according to the message update schedule. The new value m'_{cd} is computed from its dependent messages in $\mathbf{m}$, according to (6.1). Finally, the old message (c, d) in $\mathbf{m}$ is replaced with the newly computed value m'_{cd}.

The important part of this description is the distinction between when a message update is *computed* and when it is *performed*. When a message is computed, this means that its new value is calculated according to (6.1). When a message is performed, this means that the current message vector $\mathbf{m}$ is updated with the new value. Synchronous BP implementations compute all of the updates first, and then perform them all at once. Asynchronous BP implementations almost always perform an update as soon as it is computed, but it is possible to compute an update solely in order to determine its priority, and not perform the update until later. As we describe below, this is exactly the technique used by the Elidan et al. [30] schedule.

6.1.2 Estimating Message Residuals

In this section, I describe how to compute an upper bound on the error of a message, which will be used as a priority for scheduling messages. I define the *error* $e_{cd}(y_{cd})$ of a message $m_{cd}^{(k+1)}(y_{cd})$ as its multiplicative distance from its previous value $m_{cd}^{(k)}(y_{cd})$, so that

$$m_{cd}^{(k+1)}(y_{cd}) = e_{cd}(y_{cd}) m_{cd}^{(k)}(y_{cd}). \tag{6.3}$$

I define the *residual* of a message $m_{cd}^{(k+1)}(y_{cd})$ as the worst error over all assignments, that is,

$$r(m_{cd}^{(k+1)}) = \max_{y_{cd}} \left|\log e_{cd}(y_{cd})\right| = \max_{y_{cd}} \left|\log \frac{m_{cd}^{(k+1)}(y_{cd})}{m_{cd}^{(k)}(y_{cd})}\right|. \tag{6.4}$$

This corresponds to using the infinity norm to measure the distance between log message vectors, that is, $\| \log m_{cd}^{(k+1)} - \log m_{cd}^{(k)} \|_\infty$.

An alternative error measure is the *dynamic range* of the error, which has been studied by Ihler et al. [49]. This is

$$d(m_{cd}^{(k+1)}) = \max_{y_{cd}, y'_{cd}} \log \frac{e_{cd}(y_{cd})}{e_{cd}(y'_{cd})} \tag{6.5}$$

Later we compare the residual and the dynamic error range as priority functions for message scheduling.

In the rest of this section, we show how to upper-bound the message errors in two different situations: when the values of a message's dependents change, and when the factors of the model change.

First, suppose that we have available a previously-computed message value for $m_{cd}^{(k)}(y_d)$, so that

$$m_{cd}^{(k)}(y_d) = \kappa \sum_{y_{cd}} \Psi_c(y_c) \prod_{\{b \in N(c)\} \setminus d} m_{bc}^{(k)}(y_c), \tag{6.6}$$

and that now new messages $\{m_{bc}^{(k+1)}\}$ are available for the dependents. We wish to upper bound the residual $r(m_{cd}^{(k+1)})$ without actually repeating the update (6.6). Then the residual can be upper-bounded simply by the following:

$$r(m_{cd}^{(k+1)}) \leq \sum_{\{b \in N(c)\}} r(m_{bc}^{(k+1)}). \tag{6.7}$$

Proof. We show that the residual is both subadditive and contracts under the message update, following [49]. To show subadditivity, define the message product $M_{bc}^{(k+1)}(y_c) = \prod_{\{b \in N(c)\} \setminus d} m_{bc}^{(k+1)}(y_c)$, and define $M_{bc}^{(k)}$ similarly. Also, define the residual $r(M_{bc}^{(k+1)})$ as

$$r(M_{bc}^{(k+1)}) = \max_{y_c} \left| \log \frac{M_{bc}^{(k+1)}(y_c)}{M_{bc}^{(k)}(y_c)} \right| \tag{6.8}$$

Then we have

$$\begin{aligned} r(M_{bc}^{(k+1)}) &= \max_{y_c} \left| \sum_b \log \frac{m_{bc}^{(k+1)}(y_c)}{m_{bc}^{(k)}(y_c)} \right| \\ &\leq \sum_b \max_{y_c} \left| \log \frac{m_{bc}^{(k+1)}(y_c)}{m_{bc}^{(k)}(y_c)} \right| = \sum_b r(m_{bc}^{(k+1)}), \end{aligned}$$

which follows from the subadditivity of absolute value, and an increase in the degrees of freedom of the maximization.

To show contraction under the message update, we apply the fact that

$$\frac{f_1 + f_2}{g_1 + g_2} \leq \max\left\{\frac{f_1}{g_1}, \frac{f_2}{g_2}\right\}. \tag{6.9}$$

This directly yields

$$r(m_{cd}^{(k+1)}) = \max_{y_{cd}} \left| \log \frac{\sum_{y_c \setminus y_{cd}} \Psi_c(y_c) M_{bc}^{(k+1)}}{\sum_{y_c \setminus y_{cd}} \Psi_c(y_c) M_{bc}^{(k)}} \right| \tag{6.10}$$

$$\leq \max_{y_{cd}} \left| \log \max_{y_c \setminus y_{cd}} \frac{\Psi_c(y_c) M_{bc}^{(k+1)}}{\Psi_c(y_c) M_{bc}^{(k)}} \right| \tag{6.11}$$

$$\leq \max_{y_{cd}} \max_{y_c \setminus y_{cd}} \left| \log \frac{M_{bc}^{(k+1)}(y_c)}{M_{bc}^{(k)}(y_c)} \right| \tag{6.12}$$

$$= r(M_{bc}^{(k+1)}). \tag{6.13}$$

□

Now consider the second situation, when a factor Ψ_a changes. Define e_a to be the multiplicative error in the factor, so that

$$\Psi_a^{(k+1)}(\mathbf{y}_a) = e_a(\mathbf{y}_a)\Psi_a^{(k)}(\mathbf{y}_a). \tag{6.14}$$

Suppose we have already computed a message $m_{cd}^{(k)}$, so that in the current message vector

$$m_{cd}^{(k)}(y_d) = \sum_{y_{cd}} t_c^{(k)}(y_c) \prod_{\{b \in N(c)\} \setminus d} m_{bc}^{(k)}(y_c), \tag{6.15}$$

and as before we wish to upper bound $r(m_{cd}^{(k+1)})$. Then substitution into (6.4) yields

$$r(m_{cd}^{(k+1)}) \leq \max_{\mathbf{y}_a} \frac{\Psi_a^{(k+1)}(\mathbf{y}_a)}{\Psi_a^{(k)}(\mathbf{y}_a)}. \tag{6.16}$$

6.1.3 Dynamic BP Schedules

In this section, I describe the previously proposed schedule (RBP0L) and our new schedule (RBP1L).

Algorithm 6.2 RBP1L [30]

```
function RBP1L ()
 1: $\mathbf{m} \leftarrow$ uniform message array
 2: $q \leftarrow$ INITIALPQ()
 3: repeat
 4:    $m_{bc} \leftarrow$ DEQUEUE($q$)
 5:    $\mathbf{m}|_{bc} \leftarrow m_{bc}$ {Perform update.}
 6:    for all $d$ in $\{d \in N(c)\} \backslash b$ do
 7:       Compute update $m_{cd}$
 8:       Remove any pending update $m_{cd}^{(k)}$ from $q$
 9:       Add $m_{cd}$ to $q$ with priority $r(m_{cd})$
10:    end for
11: until messages converged
function INITIALPQ ()
 1: $q \leftarrow$ empty priority queue
 2: for all messages $(c, d)$ do {Initialize $q$}
 3:    Compute update $m_{cd}$
 4:    Add $m_{cd}$ to $q$ with priority $r(m_{cd})$
 5: end for
 6: return $q$
```

6.1.3.1 Residual BP with Lookahead (RBP1L)

Elidan et al. [30] call their algorithm *residual belief propagation*, but in the next section we introduce a different BP schedule that also depends on the message residual. Therefore, to avoid confusion we refer to the Elidan et al. algorithm by the more specific name of *residual BP with lookahead one (RBP1L)*.

The basic idea in RBP1L (Algorithm 6.2) is that whenever a message m_{cd} is pending for an update, the message is computed and placed on a priority queue to be performed. The priority of the message is the distance between its current value and its newly-computed value: the exact distance measure is not specified by Elidan et al., although they assume that it is based on a norm $\|m_{cd} - m_{cd}^{(k)}\|$ between the difference in message values. We use the residual (6.4) between log message values.

The problem with this schedule can be seen in Lines 7–9 of Algorithm 6.2. When an update m_{bc} is performed, each of its dependents m_{cd} is recomputed and placed in

Algorithm 6.3 RBP0L

```
function RBP0L ()
 1: $\mathbf{m} \leftarrow$ uniform message array
 2: $T \leftarrow$ total residuals; initialized to 0
 3: $q \leftarrow$ INITIALPQ()
 4: repeat
 5:     $m_{bc} \leftarrow$ DEQUEUE($q$)
 6:     Compute update $m_{bc}$ and residual $r = r(m_{bc})$
 7:     $\mathbf{m}|_{bc} \leftarrow m_{bc}$ {Perform update.}
 8:     For all $ab$, do $T(ab, bc) \leftarrow 0$
 9:     For all $cd$, do $T(bc, cd) \leftarrow T(bc, cd) + r$
10:     for all $d$ in $\{d \in N(c)\} \backslash b$ do
11:         $v \leftarrow \sum_a T(ac, cd)$
12:         Remove any pending update $(c, d)$ from $q$
13:         Add $m_{cd}$ to $q$ with priority $v$
14:     end for
15: until messages converged

function INITIALPQ ()
 1: $q \leftarrow$ empty priority queue
 2: for all messages $(c, d)$ do {Initialize $q$}
 3:     Compute update $m_{cd}$
 4:     $v \leftarrow \max_{y_c} |Y_c| \left|\log \Psi_c(y_c)\right|$
 5:     Add $m_{cd}$ to $q$ with priority $v$
 6: end for
 7: return $q$
```

the queue. If a previous update $m_{cd}^{(k)}$ was already pending in the queue, then that message is discarded. I refer to this as a "wasted" update. In Section 6.1.4, we see that this is a relatively common occurrence in RBP1L, so preventing this can yield significant gains in convergence speed.

6.1.3.2 Avoiding Lookahead (RBP0L)

In this section we present our dynamic schedule, *residual BP with lookahead zero (RBP0L)*. In Section 6.1.2 we saw that a residual can be upper-bounded by its sum of incoming residuals. The idea behind RBP0L is to use that upper bound as the message's priority, so that an update is never computed unless it will actually be performed. The full algorithm is given in Algorithm 6.3.

There are three fine points here. The first question is how to update the residual estimate when a message $m_{bc}(y_c)$ is updated twice before one of its dependents $m_{cd}(y_d)$ is updated even once. In the most general case, each dependent may have actually seen a different version of m_{bc} when it was last updated. Naively applying the bound (6.7) would suggest that we retain the version of m_{bc} as it was when each of its dependents last saw it. But this becomes somewhat expensive in terms of memory. Instead, for each pair of messages (b, c) and (c, d) we maintain a total residual $T(bc, cd)$ of how much the message m_{bc} has changed since m_{cd} was last updated. Estimates of the priority of m_{cd} are always computed using the total residual, rather than the single-update residual. (This preserves the upper-bound property of the residual estimates.)

The second question is how to initialize the residual estimates. Recall that the messages $\mathbf{m}$ are initialized to uniform. Imagine that those initial messages were obtained by starting with a factor graph in which all factors Ψ_a are uniform, running BP to convergence, and then modifying the factors to match those in the actual graph. From this viewpoint, the argument in Section 6.1.2 shows that an upper bound on the residual from uniform messages is

$$r(m_{cd}) \leq \max_{y_c} \left| \log \frac{\Psi_c(y_c)}{u_c(y_c)} \right|, \tag{6.17}$$

where u_c is a normalized uniform factor over the variables in y_c. Therefore, we use this upper bound as the initial priority of each update.

Finally, we need a way to approximate the residuals if damping is used. The important point here is that when a message m_{bc} is sent with damping, even after the update is performed, the residual m_{bc} is nonzero, because the full update has not been taken. To handle this, whenever a damped message m_{cd} is sent, the residual $r(m_{bc})$ is computed exactly and m_{bc} is added to the queue with that priority. (For simplicity, this is not shown in Algorithm 6.3.)

6.1.3.3 Application to Non-inference Domains

RBP1L has the advantage of being more general: it can readily be applied to any set of fixed-point equations, potentially ones that are very different than those used in approximate inference. On the other hand, RBP0L appears to be more specific to BP, because the residual bounds assume that BP updates are being used. For similar algorithms, such as max-product BP and GBP, it is likely that the same scheme would be effective. For a completely different set of fixed-point equations, applying RBP0L would require both designing a new method for approximating the update residuals, and designing an efficient way for initializing the residual updates. That said, our residual estimation procedure, which simply sums up the antecedent residuals, is fairly generic, and thus likely to perform well in a variety of domains.

6.1.4 Experiments

In this section, we compare the convergence speed of RBP0L and RBP1L on both synthetic and real-world graphs.

6.1.4.1 Synthetic Data

We randomly generate $N \times N$ grids of binary variables with pairwise Potts factors. Each pairwise factor has the form

$$\Psi_{ij}(y_s, y_t) = \begin{pmatrix} 1 & e^{-\alpha_{ij}} \\ e^{-\alpha_{ij}} & 1 \end{pmatrix}, \tag{6.18}$$

where the equality strength α is sampled uniformly from $[-C, C]$. Higher values of C make inference more difficult. The unary factors have the form $\Psi_s(y_s) = [1 \quad e^{-u_s}]$, where u_s is sampled uniformly from $[-C, C]$. We generate 50 distributions for $C = 5$. For smaller values of C, inference becomes so easy that all schedules performed equally well. For larger values of C, the same trend holds, but the the convergence rates are

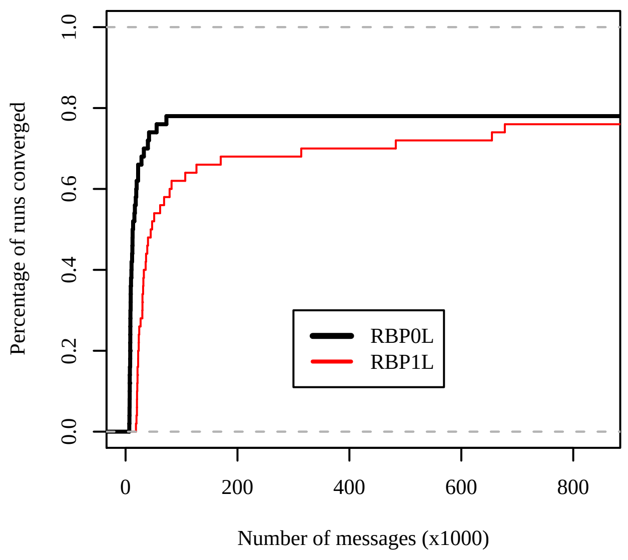


Figure 6.1. Convergence of RBP0L and RBP1L on synthetic 10×10 grids with $C = 5$. The x-axis is number of messages computed. RBP0L converges faster.

much lower. We use the grid size $N = 10$ so that exact inference is still feasible. We measure running time by the number of message updates computed. This measure closely matches the CPU time. Both algorithms are considered to have converged when no pending update has a residual of greater than 10^{-3}. The algorithms are considered to have diverged if they have not converged after the equivalent of 1000 complete sweeps of the graph.

The rate of convergence of the different schedules are shown in Figure 6.1. We see that RBP0L converges much more rapidly than RBP1L, although both eventually converge on the same percentage of networks.

Figure 6.2 shows the number of messages required for convergence for each sampled model. Each integer on the x-axis represents a different randomly-generated model, sorted by the number of messages required by RBP1L. Thus, the model at x-index 0 is the easiest model for RBP1L, and so on. Each curve is the number of messages

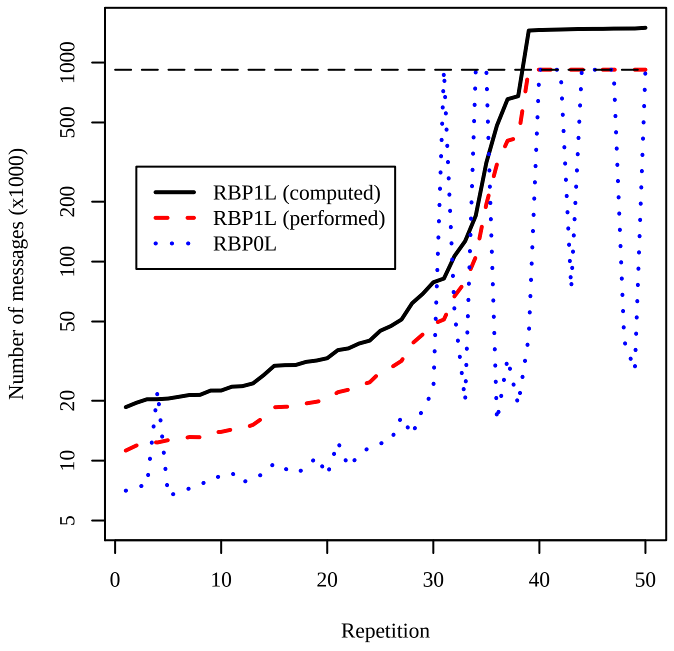


Figure 6.2. Updates performed by RBP0L and RBP1L on synthetic data. The horizontal line is the number-of-messages cutoff. The y-axis is logarithmic.

	Messages sent	Accuracy
TRP	3 079 570	**97.6**
RBP0L	**839 250**	97.4
RBP1L	2 685 702	97.3

Table 6.1. Performance of BP schedules on skip-chain test data.

required, as a function of this rank. The horizontal line is the number-of-messages cutoff, so points that exceed that line represent models for which BP did not converge. The y-axis is logarithmic.

RBP0L computes on average half as many messages as RBP1L. RBP0L uses fewer messages than RBP1L in 46 of the 50 sampled models. In three of the sampled models, RBP1L converges but RBP0L does not, which appear in Figure 6.2 as the peaks where the RBP0L curve is the only one that touches the horizontal line. In three other models, RBP0L converges but RBP1L does not, which appear as the valleys where RBP0L does not touch the horizontal line, but the other curves do. The dashed curve in the figure shows the number of updates actually performed by RBP1L. On average, 38% of the updates computed by RBP1L are never performed. Surprisingly, RBP0L performs fewer updates than RBP1L performs; that is, it is more efficient even if wasted updates are not counted against RBP1L. This may be a beneficial effect of our choice of initial residual estimates.

Finally, we measure the accuracy of the marginals for RBP0L and RBP1L. For both schedules, we measure the average per-variable KL from the exact distribution to the BP belief. When both schedules converge, the average per-variable KL is nearly identical: the mean absolute difference, averaged over the 50 random models, is 0.0038.

6.1.4.2 Natural-Language Data

Finally, we consider a model with many irregular loops, which is the skip chain conditional random field introduced in Section 3.2. This model incorporates certain long-distance dependencies between word labels into a linear-chain model for information extraction. The resulting networks contain many loops of varying sizes, and exact inference using a generic junction-tree solver is intractable. We evaluate on the seminars data set described in that section. The emails on average contain 273.1 tokens, but the maximum is 3062 tokens. The messages have an average of 23.5 skip edges, but the maximum is 2260, indicating that some networks are connected densely.

We generate networks as follows. Using ten-fold cross-validation with a 50/50 train/test split, we train a skip-chain CRF using TRP until the model parameters converge. Then we evaluate the RBP0L, RBP1L, and TRP on the test data, measuring the number of messages sent, the running time, and the accuracy on the test data. As in the last section, RBP0L and RBP1L are considered to have converged if no pending update has a residual of more than 10^{-3}. TRP is considered to have converged if no update performed on the previous iteration resulted in a residual of greater than 10^{-3}. In all cases, the trained model parameters are exactly the same; the inference algorithms are varied only at test time, not at training time.

Table 6.1 shows the performance of each of the message schedules, averaged over the 10 folds. RBP0L uses one-third of the messages as RBP1L, and one-fifth of the CPU time, but has essentially the same accuracy. Also, RBL0L uses 27% of the messages used by TRP.

In our implementation, the CPU time required per message update is much higher for the RBP schedules than for TRP. The total running time for RBP0L is 66s, compared to 110s for TRP and 321s for RBP1L. This is partially because of the overhead in maintaining the priority queues and residual estimates, but also this

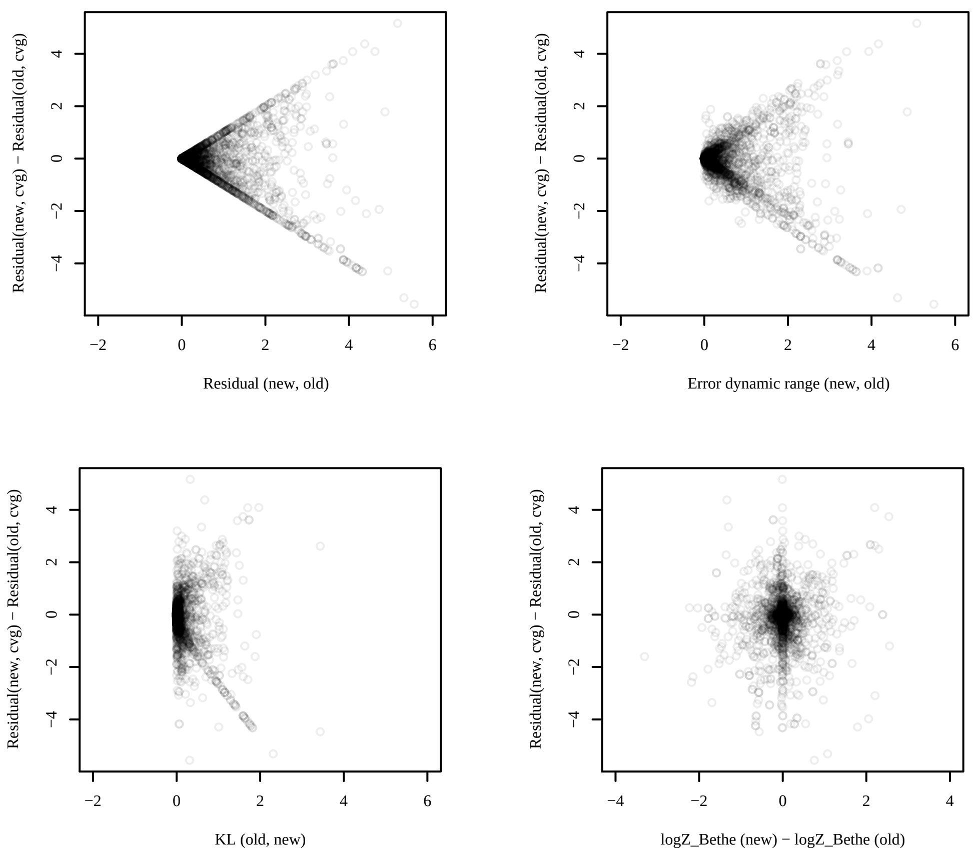


Figure 6.3. Comparison of error metrics in predicting the distance to convergence. (See text for explanation.)

is because our TRP implementation is a highly optimized one that we have used in much previous work, whereas our RBP implementations have more room for low-level optimization.

6.1.4.3 Error Estimates

The message residual is an intuitive error measure to use for scheduling, but there are many others that are conceivable. In this section, I compare different error measures to evaluate how reliable they are at predicting the next message to send. Ideally, we would evaluate a priority function for messages by whether higher priority messages actually reduces the computation time required for convergence. But it is extremely difficult to compute this, so we instead measure the distance to the converged message values, as follows.

We generate a synthetic grid as in Section 6.1.4.1. (The graphs here are from a single sampled model, but different samples result in qualitatively similar results.) Then, we run RBP0L on the grid to convergence, yielding a set of converged messages $\mathbf{m}^*$. Finally, we run RBP0L again on the same grid, without making use of $\mathbf{m}^*$. After each message update of RBP0L $m_{cd}^{(k)} \mapsto m_{cd}^{(k+1)}$, we measure:

a. The residual of the errors $e(m_{cd}^{(k)}, m_{cd}^{(k+1)})$, $e(m_{cd}^{(k)}, m_{cd}^*)$, and $e(m_{cd}^{(k+1)}, m_{cd}^*)$

b. The dynamic range of the same errors

c. The KL divergences $\mathrm{KL}(m_{cd}^{(k)} \| m_{cd}^{(k+1)})$, $\mathrm{KL}(m_{cd}^{(k)} \| m_{cd}^*)$, and $\mathrm{KL}(m_{cd}^{(k+1)} \| m_{cd}^*)$;

d. The change in Bethe energy $\log Z_{\mathrm{BP}}(\mathbf{m}^{(k+1)}) - \log Z_{\mathrm{BP}}(\mathbf{m}^{(k)})$.

Thus we can measure how well each of the error metrics predicts the distance to convergence $r(e(m_{cd}^{(k+1)}, m_{cd}^*)) - r(e(m_{cd}^{(k)}, m_{cd}^*))$. This is shown in Figure 6.3. Each plot in that figure shows a different distance measure between messages: from top left, they are message residual, error dynamic range, KL divergence, and difference in Bethe energy. Each point in the figures represents a single message update. In all figures, the x-axis shows the distance between the message $m_{cd}^{(k)}$ at the previous iteration and the value $m_{cd}^{(k+1)}$ at the current iteration. The y-axis shows the change in distance to the converged messages, that is, how much closer the update at $k+1$ brought the message to its converged value. We measure this as the difference between the residuals $e(m_{cd}^{(k+1)}, m_{cd}^*)$ and $e(m_{cd}^{(k)}, m_{cd}^*)$. Negative values of this measure are better, because they mean that the distance to the converged messages has decreased due to the update. An ideal graph would be a line with negative slope.

Both the message residual and the dynamic error range display a clear upper-bounding property on the absolute value. Also, the points are somewhat clustered along the diagonal, indicating some kind of a linear relationship between the single-message distance and the distance to convergence. The single-message distance does not seem to do well, however, at predicting *in which direction* the message will change, that is, closer or farther from its converged value. Qualitatively, the residual and the error range seem to perform similarly at predicting the distance to the converged messages, but in preliminary experiments, using the error range in a one-lookahead schedule seemed to converge slightly slower than using the residual.

The message KL also seems to do a poor job of predicting the distance to the converged message. More surprisingly, the difference in Bethe energy is almost completely uninformative about the distance to converged messages. This suggests an intriguing explanation of the slow converge of gradient methods for optimizing the Bethe energy: perhaps the objective function itself is simply not good at measuring what we care about. It is possible that the Bethe approximation may be accurate at convergence but still not be accurate outside of the constraint set, that is, when the messages are not locally consistent. This is precisely the situation that occurs during message scheduling. For this reason, it may be more revealing to look at the Lagrangian of the Bethe energy rather than the objective function itself.

6.2 Dynamic Schedules for Inference and Learning

In this section, I describe dynamic schedules for the combined inference and learning problem. Learning algorithms for structured models—such as maximum likelihood, max-margin methods [25, 129], and search-based approaches [28]—all require inference across a set of labeled training examples, which is then repeated for many settings of the model parameters. This repeated inference is intractable for general models and can be expensive even for tractable models when the training set is large.

Most practical methods for learning structured models can be abstractly described as follows. The object is to optimize some loss function $\ell(\theta)$ with respect to the model parameters θ. The parameters are updated iteratively, so that at iteration t, we take the current parameter setting $\theta^{(t)}$ and based on the training data, compute an update direction $\Delta\theta$, and set $\theta^{(t+1)} \leftarrow \theta^{(t)} + \alpha\Delta\theta$, where α is some step size. For example, in maximum likelihood training, ℓ is the likelihood, and $\Delta\theta$ the likelihood gradient with respect to θ. Often, computing $\Delta\theta$ will be intractable in general, so some approximation is used, such as a variational approximation for the likelihood, or a best-first search for a max-margin method.

This leads to two observations. First, not only is the parameter estimation algorithm iterative, but often the inference algorithm is iterative as well. Iterative inference algorithms include variational methods such as mean field and belief propagation, and search algorithms such as simulated annealing. The second observation is that some areas of the model may be more difficult than others for training or inference. If a certain area of the model is easy to train, we could save time by "locking" its parameters and focusing computational effort on the more difficult areas of the model.

In this section, we exploit both of these observations by defining a single set of fixed-point updates that integrate inference and learning. We focus on the case of training loopy Markov random fields using belief propagation (BP). We view the BP message updates and the likelihood gradient updates as a single set of fixed point equations, which we are free to iterate according to any schedule. Ordinarily, these are scheduled by running the belief equations to convergence, using those beliefs to compute an approximate gradient, and taking a step in that direction. But more efficient schedules are possible. In particular, dynamic schedules provide an especially great amount of flexibility:

- First, a dynamic scheduler can choose to make gradient updates before the inference updates have converged, thus providing a way to perform training using early stopping of BP.

- Second, just as a schedule can prioritize a message if its incoming messages change greatly, it can also prioritize a message whose local factor has changed greatly due to a parameter update. Thus, inference can focus on the areas of the model in which the parameters are changing most rapidly. In particular, a dynamic schedule is free to ignore regions of the model for which training has essentially converged.

- Finally, if the training set consists of iid examples, it can be presented to the scheduler as a single, disconnected graphical model. This means that when the scheduler is choosing which area of the model to perform inference in, it is also choosing which *training instance* to perform inference in. Because gradient updates occur can before performing inference on the entire training set, the resulting method is similar to stochastic gradient descent, except that the scheduler chooses the batch compositions automatically.

Also, Teh and Welling [132] have previously proposed an algorithm that integrates message updates and updates from iterative scaling. Since iterative scaling can also be used for parameter estimation in Markov random fields, their technique is in the same spirit as ours. Iterative scaling has repeatedly been shown to converge much slower than gradient updates for parameter estimation [65, 79, 112, 143], so a method that uses gradient updates has the potential to be a significant improvement.

In this section, I present an integrated system of equations for gradient and BP updates of Markov random fields (Section 6.2.1), and we describe how the updates may be iterated using a dynamic schedule. Then, we show that the integrated schedule converges significantly faster than running BP to convergence (Section 6.2.2), resulting in an almost three-fold decrease in training time with equivalent likelihood.

6.2.1 Combining Inference and Learning

Let $p(\mathbf{y})$ factorize according to a factor graph with factors $\{\Psi_a\}$, where as usual each Ψ_a has the exponential form $\Psi_a(\mathbf{y}_a) = \exp\{\sum_k \theta_{ak} f_{ak}(\mathbf{y}_a)\}$. We wish to estimate the parameters given data with empirical distribution $\tilde{p}$. In Section 3.1.4, we saw that the gradient of the BP likelihood is

$$\frac{\partial \ell_{\text{BP}}}{\partial \theta_a} \overset{\text{def}}{=} g_a(\mathbf{y}_a) = \sum_{\mathbf{y}_a} \tilde{p}(\mathbf{y}_a) f_a(\mathbf{y}_a) - \sum_{\mathbf{y}_a} q_a(\mathbf{y}_a) f_a(\mathbf{y}_a), \tag{6.19}$$

Also, we have seen that the beliefs are computed by finding fixed-points of the system of equations

$$m_{as}(y_s) \leftarrow \sum_{\mathbf{y}_a \backslash y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in N(a)\backslash s} m_{ta}(y_t), \tag{6.20}$$

$$m_{sa}(y_s) \leftarrow \prod_{b \in N(s)\backslash a} m_{bs}(y_s) \tag{6.21}$$

upon which the beliefs are calculated as

$$q_a(\mathbf{y}_a) = \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(y_s) \tag{6.22}$$

Thus, inference and learning can be viewed as attempting to find a fixed point of a single system of equations. This system is

$$\begin{aligned} m_{as}(y_s) &\leftarrow \sum_{\mathbf{y}_a \backslash y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in N(a)\backslash s} m_{ta}(y_t) \\ m_{sa}(y_s) &\leftarrow \prod_{b \in N(s)\backslash a} m_{bs}(y_s) \\ q_a(\mathbf{y}_a) &\leftarrow \Psi_a(\mathbf{y}_a) \prod_{s \in a} m_{sa}(y_s) \\ \theta_a &\leftarrow \theta_a + \alpha g_a(\theta_a) \end{aligned} \tag{6.23}$$

where α is a step size, and g_a is the approximate gradient (6.19).

Algorithm 6.4 IRBP

```
function IRBP (G)
 1: $q \leftarrow$ INITIALPQ($G$)
 2: repeat
 3:    $v_{lhs} \leftarrow$ DEQUEUE($q$)
 4:    Perform update indicated by $v_{lhs}$
 5:    for all equations $e$ that depend on $v_{lhs}$ do
 6:       ADD($q, e$)
 7:    end for
 8: until updates converged
function INITIALPQ (G)
 1: $q \leftarrow$ empty priority queue
 2: ADD($q, m_{sa}$)      $\forall (s, a) \in G$
 3: ADD($q, m_{as}$)      $\forall (s, a) \in G$
 4: ADD($q, \Psi_a$)      $\forall a \in G$
 5: return $q$
function ADD ($q, v_{lhs}$)
 1: $m \leftarrow$ Current value of variable $v_{lhs}$
 2: $m' \leftarrow$ Compute new value of $v_{lhs}$ from its associated equation
 3: Add $v_{lhs}$ to $q$ with priority $\|m' - m\|$
```

This is a system of fixed-point equations, in which the variables are $V = \{m_{as}\} \cup \{m_{sa}(y_s)\} \cup \{q_a(\mathbf{y}_a)\} \cup \{\theta_a\}$. In general, a fixed point of this system can be computed using a dynamic propagation schedule, by maintaining a priority queue of pending updates, and performing message updates that would cause the greatest residual. Let v_{lhs} denote some variable on the left-hand side of one of the update equations (6.23). Let $v_{lhs}^{(t)}$ be the value of some variable $v \in V$ at some iteration t. Then we define the *residual* of the update at iteration t as the vector $\|v_{lhs}^{(t+1)} - v_{lhs}^{(t)}\|$, where $\|\cdot\|$ is a norm. We use the infinity norm, so that the residual is the maximum absolute value.

However, the equations (6.23) are in fact not amenable to dynamic scheduling. The reason for this is that the gradient update on θ does not decrease its residual. Once a set of messages are fixed, the gradient update is linear in θ, and so has no maximum. There is nothing to stop an uniformed schedule from iterating the gradient update infinitely.

We can avoid this problem in two ways. First, adding regularization to the objective penalizes large parameter values directly. Second, we can cast the gradient update by in a different way. A descent step along the gradient (6.19) can be viewed as updating each factor Ψ_a by:

$$\begin{aligned} \Psi_a(\cdot;\theta_a) &\leftarrow \Psi_a(\cdot;\theta_a + \alpha g_a(\theta_a)) && (6.24) \\ &= \Psi_a\left(\cdot;\theta_a + \alpha(\sum_{\mathbf{y}_a} \tilde{p}(\mathbf{y}_a) - q_a(\mathbf{y}_a))\right) && (6.25) \end{aligned}$$

This leads to the system of equations

$$\begin{aligned} m_{as}(y_s) &\leftarrow \sum_{\mathbf{y}_a \backslash y_s} \Psi_a(\mathbf{y}_a) \prod_{t \in N(a) \backslash s} m_{ta}(y_t) \\ m_{sa}(y_s) &\leftarrow \prod_{b \in N(s) \backslash a} m_{bs}(y_s) \\ \Psi_a(\cdot;\theta_a) &\leftarrow \Psi_a\left(\cdot;\theta_a + \alpha\left[\sum_{\mathbf{y}_a} \tilde{p}(\mathbf{y}_a) - \Psi_a(\mathbf{y}_a)\prod_{s \in a} m_{sa}(y_s)\right]\right). \end{aligned} \qquad (6.26)$$

The main advantage to this viewpoint is that whenever two gradient updates are made in a row, then factor resulting from the new parameters are used to recompute the belief. This breaks the self-loop, because now altering Ψ_a causes q_a to better match the empirical belief, and thus the gradient to decrease. This also helps to prevent the factor updates from being selected too often.

Now this system of equations (6.26) can be iterated to find a fixed point. Any fixed point corresponds to a saddlepoint of the BP likelihood. In fact, this system of equations generalizes a typical method of training using loopy BP. Typically [e.g., 128], we iterate the BP iterations to convergence, then take a gradient step using a second-order method, and continue until convergence; this can be seen as a particular schedule for solving the system (6.26). Running BP to convergence may be wasteful, however, over several gradient steps, the beliefs may remain stable in some areas of the model, meaning that running BP to convergence in those parts of the model is wasteful. An alternative idea is to allow taking gradient steps before the BP steps have converged; that is, to schedule the equations (6.26) as a single set of fixed-point updates, in which the scheduler makes no distinction between gradient updates and BP updates. This is the approach that we take here.

Running BP to convergence is one schedule for propagating the updates (6.26), but other schedules may be more efficient. Recently, it has been shown that dynamic schedules for belief propagation—in particular, *residual belief propagation (RBP)*, which propagates messages with the largest residual first—can lead to a great improvement in inference time [30]. But this technique can be applied to any system of fixed-point equations, and here we apply it to the integrated inference/estimation system of equations (6.26). The potential advantage of this schedule is that the number of inference updates and gradient updates can be varied in different portions of the model, and in different regions of parameter space. We call this method *integrated residual belief/gradient propagation (IRBP)*. It is described in detail in Algorithm 6.4.

6.2.2 Experiments

In this section, we compare the convergence speed of IRBP to a gradient update in which BP is run to convergence. We randomly generate $N \times N$ grids of binary variables with pairwise Potts factors. Each pairwise factor has the form

$$\Psi_{ij}(y_s, y_t) = \begin{pmatrix} 1 & e^{-\alpha_{ij}} \\ e^{-\alpha_{ij}} & 1 \end{pmatrix}, \tag{6.27}$$

where the equality strength α is sampled uniformly from $[-C, C]$. Higher values of C mean that the constraints are stronger on average. The unary factors have the form $\Psi_s(y_s) = [1 \quad e^{-u_s}]$, where u_s is sampled uniformly from $[-C, C]$. I generate 50 distributions using $C = 4$. I use the grid size $N = 10$. For each sampled model, we

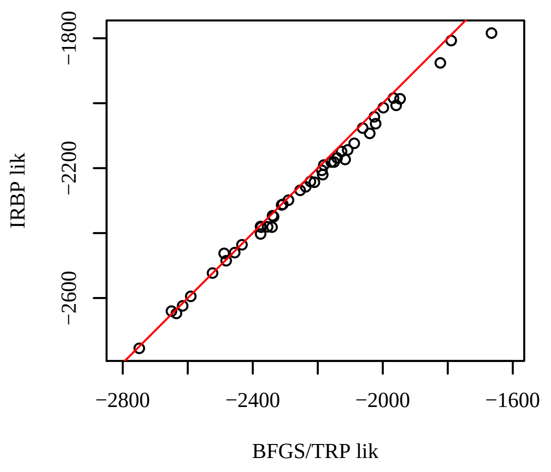


Figure 6.4. Approximate likelihood of final IRBP parameters compared to BFGS/TRP on generative synthetic model. The red line is $y = x$.

generate 100 training examples. I use a Gaussian prior on parameters with variance $\sigma^2 = 10$.

I compare the IRBP schedule to parameter estimation using BFGS, with the marginals computed by running TRP to convergence (BFGS/TRP). TRP has previously been shown to outperform synchronous and naive asynchronous BP schedules [30, 139]. IRBP converges dramatically faster than BFGS/TRP. Figure 6.5 shows the running time measured by floating point operations for each sampled model: on average BFGS/TRP requires 8 times as many floating point operations. The y-axis in Figure 6.5 shows the ratio of BFGS/TRP flops to IRBP flops. Figure 6.6 shows the CPU time used for each sampled model. Here the y axis shows the ration of running times of BFGS/TRP to IRBP. On average, IRBP uses 2.75 times less CPU time than BFGS/TRP. It is interesting that the speed difference is much more pronounced when measured in floating-point operations than wall-clock time. This indicates that the overhead of maintaining the update queue and the residuals is a significant portion of

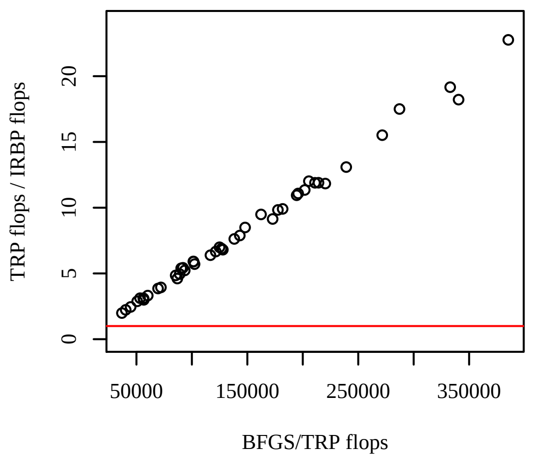


Figure 6.5. Running time of IRBP parameters and BFGS/TRP on generative synthetic model, measured in number of floating-point operations (flops). The red line is $y = 1$, which occurs when the two are equally fast.

the IRBP running time. (Our TRP algorithm has had significant amount of low-level optimization applied, while our IRBP implementation has not.)

In Figure 6.4, we show the training likelihood of the final parameter settings found by IRBP and BFGS/TRP. Because both schedules use BP, in both cases we approximate the likelihood using the Bethe energy. Both methods find equally good parameter settings.

Now, previous work has shown that a residual-based schedule for BP can greatly outperform TRP [30]. So it may be objected that the observed decrease in training time is due to using a better BP schedule, rather than interleaving the BP and gradient updates. To ensure that this is not the case, we measure the training time for simple gradient descent, where the gradient is approximated by running RBP to convergence (GD/RBP). To be clear, in GD/RBP, we use the residual schedule for the BP updates only, and never for the gradient updates. This schedule runs significantly slower than

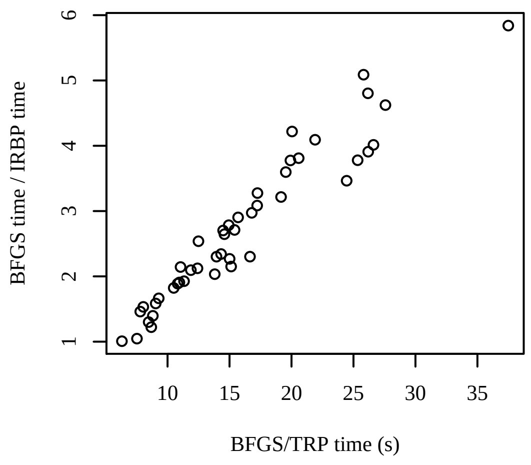


Figure 6.6. Running time of IRBP parameters and BFGS/TRP on generative synthetic model, measured in CPU time (s).

IRBP: the average training time for GD/RBP is 28.2s, while the average for IRBP is 10.4s.

6.2.3 Related Work

The closest work to ours is the unified propagation and scaling algorithm [132], which was introduced as a method for minimizing KL divergence from a reference distribution to an approximating family, which is generalization of the maximum likelihood problem. That algorithm is essentially a schedule for BP updates on the messages and iterative scaling updates on the model parameters. Thus, it can be seen as a kind of integrated inference and learning. However, it is well known that iterative scaling updates converge much slower than gradient updates for undirected parameter estimation [65, 79, 112, 143].

Bayesian methods, of course, make no distinction between inference and learning. Although theoretically attractive, integrating over the parameters becomes very difficult when the other variables in the model have complex connections of their own. For this reason, there has been very little work in Bayesian training of Markov random fields. Exceptions include Murray and Ghahramani [85] and Qi et al. [95].

For the fully-observed generative models that we consider here, parameters that maximize the BP likelihood may be found more quickly by pseudo-moment matching [142]. This does not make our techniques obsolete, however. The pseudo-moment matching estimator is unavailable when there are latent variables, or when the factors have a restricted form, such if they are restricted to a continuous exponential family, or if some parameters are tied. These situations often arise in practical models, so learning methods that compute the BP updates are still relevant.

6.3 Conclusion

In this chapter, I have explored dynamic schedules for message updates, both for the inference-only problem and the combined inference-and-learning problem. For the inference-only problem, I have presented RBP0L, a new dynamic schedule for belief propagation that schedules messages based on a upper-bound on their residual. On both synthetic and real-world data, RBP0L converges faster than both RBP1L, a recently-proposed dynamic schedule, and than TRP, with comparable accuracy. It would be interesting to explore whether the residual estimation technique in RBP0L is equally effective for other inference algorithms, such as EP, GBP, or whether the residual estimation technique would require significant adaptation. In continuous spaces, it may be that the message residual itself is not a good measure for scheduling, because it gives equal weight to all areas of the domain, even those with low probability. The KL divergence may be more appropriate.

For the inference-only problem, I have presented suggestive results that combining inference and learning into a single system of equations can lead to significant speed-ups in training time for undirected models, resulting in an almost three-fold improvement in training time with no loss in likelihood. Typically, training is performed by running an approximate inference algorithm such as BP to convergence, then using the resulting approximate marginals to approximate the gradient. But this is only one schedule for a more general system of equations, and I show that residual-based schedules can perform significantly faster. Of course, this leaves open whether this schedule would work as well on larger-scale networks, such as conditional random fields. Unfortunately, the heavy parameter tying that is typical in CRFs makes application of IRBP difficult.

CHAPTER 7
FUTURE DIRECTIONS

In this chapter, I mention some research directions in the area of approximate CRF training that may be useful for future work.

7.1 Bigger Pieces

A natural question is how readily the methods here extend to the case where pieces are treated as regions that are larger than a single factor. The difficulty seems to lie in how to handle the overlaps among larger pieces. But the connection to the Bethe energy is illuminating here. Just as the factor-as-piece likelihood arises from the dual Bethe energy with uniform messages, larger pieces can potentially be handled by using uniform messages in a more general free energy, such as region graph free energies [150]. That said, I do not believe that the models introduced in this thesis are complex enough to benefit from larger pieces.

Another question then becomes how to choose the pieces when they are larger than a single factor. Quite possibly the application itself would suggest a choice of pieces. There are also a few minimal consistency criteria that have been proposed—in particular, *maxent-normality* [150] and *non-singularity* [146]—but these, while useful, are fairly general, and do not place severe constraints on the region choice. Finally, there is a least one method in the literature for choosing regions in generalized BP [144]; however, this method chooses the regions based on properties of the entire distribution, that is, on the model parameters. It seems unwise to use the model parameters to choose the objective function that they are selected to optimize, so this techniques seems inapplicable. There is also a possible connection between methods for selecting pieces and feature selection methods.

7.2 Pseudolikelihood

Recall from Chapters 4 and 5 that pseudolikelihood [8] is defined as

$$\ell_{\text{PL}}(\theta) = \sum_s \log p(y_s | \mathbf{y}_{N(s)}, \mathbf{x}) \tag{7.1}$$

$$= \frac{\prod_{a \ni i} \Psi_a(y_s, \mathbf{y}_{N(s)}, \mathbf{x}_a)}{\sum_{y'_s} \prod_{a \ni i} \Psi_a(y'_s, \mathbf{y}_{N(s)}, \mathbf{x}_a)}. \tag{7.2}$$

Although on simple synthetic data sets, pseudolikelihood tends to perform well [90], on our benchmark NLP data it performs fairly poorly. This phenomenon warrants an explanation.

There exists a simple class of data sets in which pseudolikelihood performs pathologically. As an example, consider a two-node Boltzmann machine with binary variables. That is, each variable has a unary factor

$$\Psi_s(y_s) = [1 \quad e^{-\theta_s}] \tag{7.3}$$

and there is one binary factor

$$\Psi(y_0, y_1) = \begin{pmatrix} 1 & e^{\alpha_{ij}} \\ e^{\alpha_{ij}} & 1. \end{pmatrix} \tag{7.4}$$

Assume the data set contains two observations: $(0, 0)$ and $(1, 1)$. Then the maximum pseudolikelihood estimate becomes senseless: maximizing it will attempt to make the conditional distributions deterministic, with complete disregard for the per-variable marginals $p(y_0)$ and $p(y_1)$.

One way to understand this is that pseudolikelihood attempts to match the conditional distributions of the model to the conditional distributions of the data. But the conditional distributions given by the data need not determine a unique joint distribution; they do so only if the Gibbs sampler that they define is ergodic. This is not the case in the example above. It is unlikely, however, that this phenomenon explains the results that we see on the real-world data, however, because although pseudolikelihood performs significantly worse than piecewise ond maximum likelihood, it does not appear to be as pathologically bad as it would if this phenomenon were the culprit.

A different potential explanation is that pseudolikelihood may perform poorly in the presence of *data sparsity*, by which I mean that because there are a large number of input features, and the cardinality of the output variables is large, each input-output combination is observed only a few times, if at all. Sparsity is a ubiquitous feature of NLP applications, even when hundreds of thousands of words of labeled data are available. The reason this situation may be bad for pseudolikelihood is that if a neighborhood configuration $(y_s, \mathbf{y}_{N(s)})$ never occurs in the training data, then the objective function makes no attempt to set the model marginal of that configuration to 0.

One way to address this problem is to add a penalty to the pseudolikelihood that attempts to force such configurations to have zero probability. This is potentially a very fruitful approach in practice.

Another potential set of techniques to address this problem results from the following viewpoint. The pseudolikelihood gradient is given by

$$\frac{\partial \ell_{\text{PL}}}{\partial \theta_{ak}} = \sum_{\Psi_a \in G} d_a \sum_k f_{ak}(\mathbf{y}_a, \mathbf{x}_a) - \sum_{\Psi_a \in G} \sum_k \sum_{s \in a} \sum_{y'_s} f_{ak}(\tilde{\mathbf{y}}_a, \mathbf{x}_a) p(y'_s | \mathbf{y}_{N(s)}, \mathbf{x}) \tilde{p}(\mathbf{y}_{N(s)} | \mathbf{x}) \tag{7.5}$$

where d_a is the degree of factor a, that is, the number of variables in its domain; and $\tilde{p}$ is the empirical distribution. This can be actually be seen as an approximation to the exact likelihood gradient as follows:

$$\frac{\partial \ell}{\partial \theta_{ak}} = \sum_{\Psi_a \in G} \sum_k f_{ak}(\mathbf{y}_a, \mathbf{x}_a) - \sum_{\Psi_a \in G} \sum_k \sum_{\mathbf{y}'_a} f_{ak}(\mathbf{y}'_a, \mathbf{x}_a) p(\tilde{\mathbf{y}}_a | \mathbf{x}) \tag{7.6}$$

$$= \sum_{\Psi_a \in G} \sum_k f_{ak}(\mathbf{y}_a, \mathbf{x}_a) - \frac{1}{d_a} \sum_{\Psi_a \in G} \sum_k \sum_{\mathbf{y}'_a} \sum_{s \in a} f_{ak}(\mathbf{y}'_a, \mathbf{x}_a) p(y'_s | \mathbf{y}'_{N(s)}, \mathbf{x}) p(\mathbf{y}'_{N(s)} | \mathbf{x}) \tag{7.7}$$

$$\approx \sum_{\Psi_a \in G} \sum_k f_{ak}(\mathbf{y}_a, \mathbf{x}_a) - \frac{1}{d_a} \sum_{\Psi_a \in G} \sum_k \sum_{\mathbf{y}'_a} \sum_{s \in a} f_{ak}(\mathbf{y}'_a, \mathbf{x}_a) p(y'_s | \mathbf{y}'_{N(s)}, \mathbf{x}) \tilde{p}(\mathbf{y}'_{N(s)} | \mathbf{x}), \tag{7.8}$$

which is equivalent to the pseudolikelihood gradient. Thus, pseudolikelihood can be seen as approximating the model's neighborhood marginal $p(\mathbf{y}_{N(s)})$ by the empirical marginal $\tilde{p}(\mathbf{y}_{N(s)})$. Therefore, a potential way to improve pseudolikelihood to choose a different approximation for the neighborhood marginal, for example, using smoothing. Indeed, any approximate inference method may be used to generate the neighborhood approximation. Although I find this framework potentially appealing, it allows a wide range of learning methods, and in order to know what instance of the framework is most profitable, it would be useful to replicate the data sparsity failures in a synthetic domain, if indeed this is the culprit.

7.3 Other Directions

Other directions for future work include:

- *Online Updates.* As I mentioned in Section 1.2, for the exact CRF likelihood, online gradient updates have been shown to converge faster than second-order batch updates [42, 136]. Conceptually, there seems to be no obstacle to applying them to the piecewise likelihoods of this thesis. This is unlikely to present a significant research challenge, but verifying that this combination works could be practically important, because it could result in very fast training times.

- *Other applications.* In order to evaluate different training methods, I focus on a small set of benchmark NLP data sets, which are of practical interest, but in fact they are amenable to existing training methods. But really the point is to enable training of large, loopy CRFs which have not previously been feasible. Examples of applications that could benefit from such CRFs include cross-document information extraction and coreference, joint models for cascades of NLP tasks, and scoped learning for structured models.

- *Latent Variables.* Piecewise training and its variants assume fully labeled data, but models with latent variables are of considerable interest. The naive piecewise method is unlikely to work well with latent variables, because if the same latent variable occurs in multiple pieces, it needs to be constrained to have the same semantics in each piece, and the naive method does not do this. It is possible that the connections to BP can be used to devise a method that alternates between regular piecewise and message passing in order to handle latent variables.

- *Max-margin methods.* Both likelihood-based methods and max-margin methods require performing inference during training, so it is natural to wonder whether the methods in this thesis can be adapted to loopy max-margin models. A suggestive step in this direction is *factorized MIRA* [75], in which the margin constraints are required to hold only over single edges, rather than the entire prediction. On a dependency parsing task, this method had good accuracy, but it did not improve training time because the model had special structure that made it amenable to exact inference. It may be interesting to see whether analogs of piecewise methods work well for max-margin training on loopy models.

BIBLIOGRAPHY

[1] Pieter Abbeel, Daphne Koller, and Andrew Y. Ng. Learning factor graphs in polynomial time and sample complexity. In *Twenty-first Conference on Uncertainty in Artificial Intelligence (UAI05)*, 2005.

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

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

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

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

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

[7] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B*, 36(2):192–236, 1974.

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

[9] Julian Besag. Efficiency of pseudolikelihood estimation for simple Gaussian fields. *Biometrika*, 64(3):616–618, 1977.

[10] Jeff Bilmes. Graphical models and automatic speech recognition. In Mark Johnson, Sanjeev P. Khudanpur, Mari Ostendorf, and Roni Rosenfeld, editors, *Mathematical Foundations of Speech and Language Processing*. Springer-Verlag, 2003.

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

[12] Eric Brill. Some advances in rule-based part of speech tagging. In *National Conference on Artificial Intelligence (AAAI)*, 1994.

[13] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research*, 17, 2002.

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

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

[16] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *National Conference on Artificial Intelligence (AAAI)*, pages 328–334, 1999.

[17] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms using different performance metrics. Technical Report TR2005-1973, Cornell University, 2005. Available at `http://www.cs.cornell.edu/~alexn/`.

[18] Arthur Choi, Mark Chavira, and Adnan Darwiche. Node splitting: A scheme for generating upper bounds in Bayesian networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[19] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.

[20] Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *International Joint Conference on Artificial Intelligence (ICML)*, 2001.

[21] Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 103–110, Barcelona, Spain, July 2004.

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

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

[24] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

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

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

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

[28] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.

[29] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[30] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006.

[31] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

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

[33] Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines. In *Conference on Empirical Methods in Natural Language Proceeding (EMNLP)*, 2006.

[34] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):24–57, 2000.

[35] Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains.* PhD thesis, Carnegie Mellon University, 1998.

[36] Dayne Frietag and Andrew McCallum. Information extraction with HMMs and shrinkage. In *AAAI Workshop on Machine Learning for Information Extraction*, 1999.

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

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

[39] Zoubin Ghahramani and Michael I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29(2-3):245–273, 1997.

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

[41] Basilis Gidas. Consistency of maximum likelihood and pseudolikelihood estimators for Gibbs distributions. In Wendell Fleming and Pierre-Louis Lions, editors, *Stochastic Differential Systems, Stochastic Control Theory and Applications*. Springer, New York, 1988.

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

[43] Joshua 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.

[44] Russ Greiner, Yuhong Guo, and Dale Schuurmans. Learning coordination classifiers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

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

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

[47] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report 2000-004, Gatsby Computational Neuroscience Unit, 2000.

[48] Aapo Hyvarinen. Consistency of pseudolikelihood estimation of fully visible Boltzmann machines. *Neural Computation*, 18(10):2283–2292, 2006.

[49] Alexander T. Ihler, John W. Fisher III, and Alan S. Willsky. Message errors in belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[50] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. In Michael I. Jordan, editor, *Learning in Graphical Models*, pages 183–233. MIT Press, Cambridge, MA, 1999.

[51] Sham Kakade, Yee Whye Teh, and Sam Roweis. An alternative objective function for Markovian fields. In *In Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[52] Junhwan Kim and Ramin Zabih. Factorial Markov random fields. In *European Conference on Computer Vision (ECCV)*, pages 321–334, 2002.

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

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

[55] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Conference of the North American Chapter of the Association for Computation Linguistics (NAACL)*, 2001.

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

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

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

[59] Julia A. Lasserre, Christopher M. Bishop, and Thomas P. Minka. Principled hybrids of generative and discriminative models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 87–94, Washington, DC, USA, 2006. IEEE Computer Society.

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

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

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

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

[64] Yan Liu, Jaime Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. Segmentation conditional random fields (SCRFs): A new approach for protein fold recognition. In *ACM International conference on Research in Computational Molecular Biology (RECOMB05)*, 2005.

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

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

[67] David McAllester, Michael Collins, and Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

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

[69] Andrew McCallum and David Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *IJCAI'03 Workshop on Learning Statistical Models from Relational Data*, 2003.

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

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

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

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

[74] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the ACL*, pages 91–98, 2005.

[75] Ryan McDonald, Koby Crammer, and Fernando Pereira. Spanning tree methods for discriminative training of dependency parsers. Technical Report MS-CIS-05-11, University of Pennsylvania CIS, 2005.

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

[77] Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 362–369, 2001.

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

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

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

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

[82] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[83] Kevin Murphy and Mark A. Paskin. Linear time inference in hierarchical HMMs. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[84] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, U.C. Berkeley, July 2002.

[85] Iain Murray and Zoubin Ghahramani. Bayesian learning in undirected graphical models: Approximate MCMC algorithms. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[86] Ara V. Nefian, Luhong Liang, Xiaobo Pi, Liu Xiaoxiang, Crusoe Mao, and Kevin Murphy. A coupled HMM for audio-visual speech recognition. In *IEEE Int'l Conference on Acoustics, Speech and Signal Processing*, pages 2013–2016, 2002.

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

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

[89] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer-Verlag, New York, 1999. ISBN 0-387-98793-2.

[90] Sridevi Parise and Max Welling. Learning in Markov random fields: An empirical study. In *Joint Statistical Meeting (JSM2005)*, 2005.

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

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

[93] Leonid Peshkin and Avi Pfeffer. Bayesian information extraction network. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[94] Vasin Punyakanok, Dan Roth, Wen-tau Yih, and Dav Zimak. Learning and inference over constrained output. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129, 2005.

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

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

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

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

[99] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, 1995.

[100] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Proceeding (EMNLP)*, 1996.

[101] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006.

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

[103] Michal Rosen-Zvi, Alan L. Yuille, and Michael I. Jordan. The dlr hierarchy of approximate inference. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

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

[105] Dan Roth and Wen-tau Yih. Relational learning via propositional algorithms: An information extraction case study. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1257–1263, 2001.

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

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

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

[109] Kengo Sato and Yasubumi Sakakibara. RNA secondary structural alignment with conditional random fields. *Bioinformatics*, 21:ii237–242, 2005.

[110] Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Intl. Conf. Artificial Neural Networks (ICANN)*, pages 569–574, 1999.

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

[112] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Conference on Human Language Technology and North American Association for Computational Linguistics (HLT-NAACL)*, pages 213–220, 2003.

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

[114] Marios Skounakis, Mark Craven, and Soumya Ray. Hierarchical hidden Markov models for information extraction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.

[115] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, pages 233–272, 1999.

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

[117] Charles Sutton. Conditional probabilistic context-free grammars. Master's thesis, University of Massachusetts, 2004. URL `http://www.cs.umass.edu/~casutton/publications.html`.

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

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

[120] Charles Sutton and Andrew McCallum. Composition of conditional random fields for transfer learning. In *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.

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

[122] Charles Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient CRF training. In *International Conference on Machine Learning (ICML)*, 2007.

[123] Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

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

[125] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *International Conference on Machine Learning (ICML)*, 2004.

[126] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8:693–723, March 2007.

[127] Martin Szummer and Tom Minka. Analysis of extensions to piecewise training. Microsoft Research Internal Memo, 2006.

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

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

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

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

[132] Yee Whye Teh and Max Welling. The unified propagation and scaling algorithm. In *Advances in Neural Information Processing Systems*, volume 14, 2002.

[133] Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Mahadevan. Learning hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2001.

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

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

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

[137] Martin J. Wainwright. *Stochastic processes on graphs with cycles: geometric and variational approaches*. PhD thesis, MIT, 2002.

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

[139] Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. Tree-based reparameterization for approximate estimation on graphs with cycles. *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[140] Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. A new class of upper bounds on the log partition function. In *Uncertainty in Artificial Intelligence*, 2002.

[141] Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 45(9):1120–1146, 2003.

[142] Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. Tree-reweighted belief propagation and approximate ML estimation by pseudo-moment matching. In *Ninth Workshop on Artificial Intelligence and Statistics*, 2003.

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

[144] Max Welling. On the choice of regions for generalized belief propagation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[145] Max Welling and Yee Whye Teh. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, volume 17, 2001.

[146] Max Welling, Tom Minka, and Yee Whye Teh. Structured region graphs: Morphing EP into GBP. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[147] Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. An integrated, conditional model of information extraction and coreference with application to citation graph construction. In *20th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[148] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

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

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

[151] Alan L. Yuille and Anand Rangarajan. The concave-convex procedure (CCCP). In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1033–1040, Cambridge, MA, 2001. MIT Press.