Parse Reranking with WordNet Using a Hidden Variable Model

by

Terry Koo

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by . . . .
Michael Collins
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Parse Reranking with WordNet Using a Hidden Variable Model

by

Terry Koo

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

We present a new parse reranking algorithm that extends work in (Michael Collins and Terry Koo 2004) by incorporating WordNet (Miller et al. 1993) word senses. Instead of attempting explicit word sense disambiguation, we retain word sense ambiguity in a hidden variable model. We define a probability distribution over candidate parses and word sense assignments with a feature-based log-linear model, and we employ belief propagation to obtain an efficient implementation.

Our main results are a relative improvement of $\approx 0.97\%$ over the baseline parser in development testing, which translated into a $\approx 0.5\%$ improvement in final testing. We also performed experiments in which our reranker was appended to the (Michael Collins and Terry Koo 2004) boosting reranker. The cascaded system achieved a development set improvement of $\approx 0.15\%$ over the boosting reranker by itself, but this gain did not carry over into final testing.

Thesis Supervisor: Michael Collins
Title: Assistant Professor

Acknowledgments

I would like to thank Michael Collins for his innumerable ideas, suggestions, corrections, revisions, and quips. Throughout the course of this research, he has been unsurpassingly helpful and unwaveringly patient. Were it not for his guidance, the project could not have gotten far. I would also like to thank Regina Barzilay for her aid in revising this thesis and Kevin Murphy for helping me arrive at a clearer understanding of belief propagation, as well as the members of Mike's Student Group, who gave me many insightful suggestions during my recent talk, not a few of which have made their way into this document.

This research was supported by an NSF Graduate Research Fellowship and a supplementary MIT Research Assistantship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We present a novel method for parse reranking that is motivated by a desire to model meaningful dependency relationships. Numerous studies (Charniak 1997; Collins 1999) have shown that bigram headword statistics are helpful in resolving parse ambiguities. However, these lexical bigram statistics suffer from problems of data sparseness, and correlations between bare words only approximate the meaningful underlying dependencies. In an attempt to resolve sparseness issues and to more closely model the underlying relationships, we substitute bigrams of WordNet (Miller et al. 1993) word senses for bigrams of bare headwords.

Our reranker is an extension of the reranking approach introduced in (Michael Collins and Terry Koo 2004). As with any other system that uses word senses, we must address the word sense disambiguation problem. However, we are placed at a disadvantage because our data sets are not word sense-annotated, so our word sense inferences are essentially unsupervised. Rather than attempt to assign explicit word senses, a task that would be a labor in itself, we retain the word sense ambiguity, capturing it in a hidden variable model. We treat each candidate parse as a Markov Random Field in which the hidden variables are the unknown word senses. A probability distribution is defined over the space of candidate parses and word sense assignments using a feature-based log-linear model. By summing out the hidden variables, we obtain a distribution that reranks the candidate parses.

The choice of a log-linear probability model offers us flexibility in the choice of features. In addition, feature sets for log-linear models are easy to redefine, allowing our reranker to be repurposed quickly and with minimal human effort. For reasons of efficiency, however, we accept a pairwise restriction on the features; Section 3.2 describes this restriction and explains how it allows us to make use of the efficient belief propagation algorithm (Yedidia et al. 2002).

Our main result is a relative improvement of $\approx 0.97\%$ over the baseline parser in development testing, which translated into a $\approx 0.5\%$ relative improvement in final testing. We also performed cascaded reranking experiments, in which our reranker was appended to the (Michael Collins and Terry Koo 2004) boosting reranker. On the development set, the cascaded system achieved a relative improvement of $\approx 0.15\%$ over the boosting reranker by itself, but this gain did not carry over into final testing.

The remainder of this chapter explains the motivations behind our reranking approach and mentions some difficulties in the application of WordNet. Afterward, Chapter 2 surveys background material and related research, Chapter 3 presents our reranking algorithm, Chapter 4 describes our experiments and results, and Chapter 5 wraps up by pointing out some directions for future work.

1.1 Motivations Behind Our Approach

A considerable body of research (Collins and Brooks 1995; Charniak 1997; Collins 1999; Ratnaparkhi 1999) holds that lexical headword statistics are helpful in resolving syntactic ambiguities. This accords with linguistic intuition: since the headword of a constituent is the linguistically most important word in the constituent, we should expect that interactions between constituents are governed by correlations between their headwords.

For instance, consider the sentence "He walked the dog to the park", in which the prepositional phrase "to the park" may attach to either "walked" or "dog", as shown in Figure 1-1. The former case is the intuitive attachment, implying that the walking action was conducted along the way to the park; the meaning of the

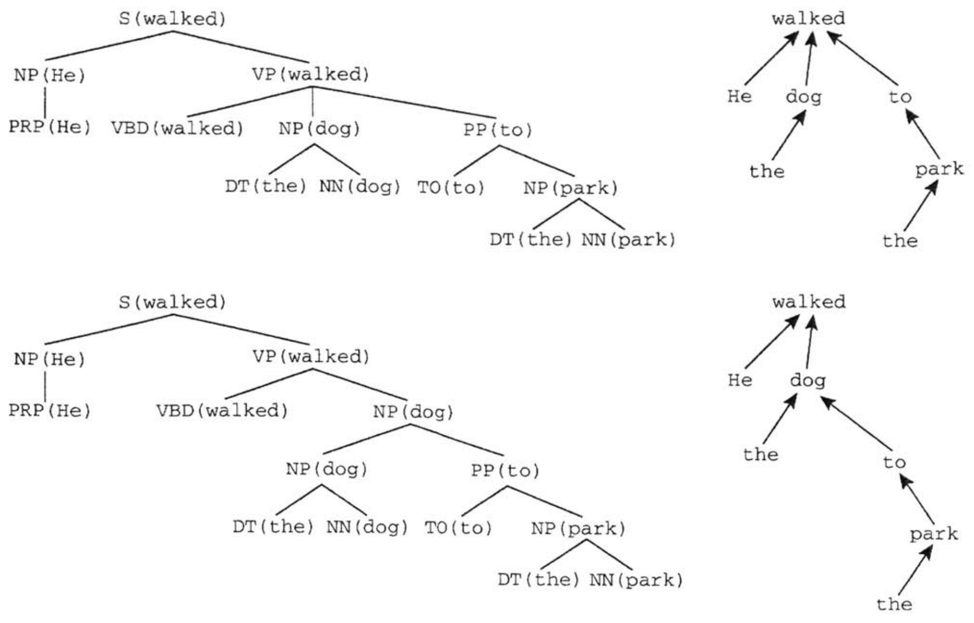


Figure 1-1: A pair of syntactic structures that depict a case of PP-attachment; both phrase structure trees and the dependency trees are shown. This ambiguity could be resolved through observation of the headword dependencies.

latter attachment is unclear, though we might imagine that the dog is somehow extending toward the park, akin to "path" in "the path to the park". The use of headword bigram statistics could disambiguate this PP-attachment since "to" modifies "walked" far more often than it modifies "dog".

Unfortunately, headword statistics are quite sparse, and it is unlikely that a training corpus will ever be made that overcomes this sparseness problem. Even if such a corpus did exist, lexicalized bigrams would still fall short of the mark. Consider the following pair of sentences

1. John charged across the battlefield.

2. John charged the defendants across the table with murder.

Bigram statistics collected from the first sentence would encourage attachment of "across" to the verb "charged", while statistics from the second sentence would discourage the verb attachment. Note that these opposed statistics are *not* a product of PP-attachment ambiguity, but of word sense ambiguity: the first sentence uses "charged" in the sense of "ran" while the second sentence uses "charged" in the sense of "accused". When the two senses are taken separately, each sense has a fairly unambiguous preference for the preposition "across"; nevertheless, a seeming PP-attachment ambiguity is observed because the bare word "charged" conflates the two senses.

Our intuition, therefore, is that dependencies between word senses will give a more meaningful measure of plausibility than dependencies between headwords. Moreover, the use of word senses will allow us to address data sparseness issues by sharing statistics among semantically related senses. For instance, we might not have collected any statistics for the noun "chihuahua". However, we can still reason about this unseen noun through statistics collected from semantically similar nouns such as "rottweiler" or "dachshund". The particular methods by which we bring about this information-sharing effect are described in greater detail in Section 4.2.

1.2 Difficulties in Using WordNet

The use of word senses introduces several difficulties. Most obviously, there is the problem of word sense ambiguity. As we have described above, we use word senses as a means of alleviating data sparseness and accessing deeper relationships. However, neither of these benefits can be achieved unless the sense ambiguity of a word is reduced. Therefore, even though we abstain from full word sense disambiguation, sense ambiguity will pose a problem for us.

Another problem is the intractability of adjective senses: unlike nouns and verbs, WordNet adjective senses are not organized under a hierarchical relationship[1] such as hypernymy, nor are they categorized into supersenses. However, as we will see in Section 4.2, our information-sharing techniques depend on either hypernyms or supersenses. We decided not to make use of adjective senses at all.

[1]In fact, it is unclear how, if at all, such a relationship could be established (Miller et al. 1993).

Chapter 2

Related Research

This chapter reviews some related research that has influenced our own research. We begin by describing efforts in statistical parsing, continue with an overview of parse reranking techniques, and finish by presenting some related projects that also make use of WordNet (Miller et al. 1993).

2.1 Statistical Parsing Approaches

The syntactic ambiguity inherent in natural language makes it difficult to design robust parsers. Attempting to resolve ambiguities deterministically is a downward spiral: augmenting and refining rule sets leads to an explosive growth in the grammar and often introduces new ambiguities. Statistical parsers, which use statistical techniques to make informed decisions in the face of parse ambiguity, have come to prominence in recent years, with the appearance of large parsed corpora such as the Penn Treebank (Marcus et al. 1994).

The simplest form of statistical parsing model is the probabilistic context-free grammar (PCFG), which simply assigns a frequency-based probability to each rule expansion in a fixed grammar. (Charniak 1997) achieves significant improvement over a plain PCFG by augmenting the probability model with headword bigram statistics. The lexical statistics are backed off by applying a word-clustering technique to the headwords; however, the results indicate that the clustering had only a small effect. One of the drawbacks of the (Charniak 1997) parser is the use of a fixed treebank grammar, which is extracted from the training corpus; if a new sentence requires a rule which was not seen in the training set, then it is impossible for the parser to produce the correct structure.

The more complex parsers of (Collins 1999) and (Charniak 2000) make use of generative probability models that employ a Markov grammar. An n^{th}-order Markov grammar models a rule expansion as a generative process that first creates the rule's head constituent and then generates the left and right modifiers with n^{th}-order Markov processes. A Markov grammar is able to outperform a fixed treebank grammar because of the Markov independence assumptions that are applied when generating left and right modifiers: the reduced conditioning context allows the parameters of the grammar to be estimated more accurately, and the modeling of rule expansions as sequential processes extends the grammar to unseen rules effortlessly.

The (Collins 1999) parser actualizes the complement-adjunct distinction through the additional complement nonterminals `NP-C`, `S-C`, `SBAR-C`, and `VP-C`. The generative model is adjusted so that left and right subcategorization frames are chosen after generating the head constituent, but before generating the right and left modifiers. These frames name the complements that the head expects to find to its left and right. Introducing the complement nonterminals is useful in itself, since the meaning of a parse cannot be recovered without first determining the complements. The use of complement nonterminals also aids parsing performance in two ways: first, the complement nonterminals allow separate distributions to be learned by complements and adjuncts, and second, the subcategorization frames allow the parser to avoid certain impossibilities, such as an intransitive verb with two objects.

An alternative statistical parsing approach is that of (Ratnaparkhi 1999), which uses a feature-based maximum-entropy probability model. The parser first performs part-of-speech tagging and low-level chunking, then combines the chunks into progressively larger constituents through a series of parsing decisions that resemble the actions of a shift-reduce parser. Each decision is assigned a probability by the maximum-entropy model, where the features of the model can access the context of the earlier decisions. One of the advantages of this approach is that the probability model is completely unspecific, whereas the probability models of the (Collins 1999) and (Charniak 2000) parsers have been specially crafted for the parsing task. Therefore, while the maximum-entropy parser achieves slightly lower performance than its more heavily tuned counterparts, Ratnaparkhi claims that significantly less work was required to create it.

2.2 Parse Reranking Approaches

Parse reranking is one of the most interesting recent developments in statistical parsing. A parse reranker is a model that reanalyzes the output of some base parser; the reranker accepts a set of the top N candidate parses created by the base parser, together with the parser's initial ranking, and induces a new ranking over these parses. The motivation is that the reranker can make use of information unavailable to the base parser, since it has access to complete parse trees and is free from the independence assumptions of the parser.

For example, a reranker can make use of long-range features, such as "there is an auxiliary verb inside this relative clause", or global features, such as "there is an SBAR in this parse tree". A parser, on the other hand, must evaluate the plausibility of a parse as the parse is being constructed, so it cannot feasibly access long-range or global information.

The reranker presented in (Michael Collins and Terry Koo 2004) uses feature selection to optimize an exponential loss function. The reranker also integrates the base parser's ranking though a special feature that gives the parser's log-probability for each candidate parse. The optimal weight for this special feature is determined ahead of time, before entering the optimization phase. This general approach to incorporating the base parser's ranking is one that we borrow for our own reranker, as described in Section 3.4.1.

Appending the (Michael Collins and Terry Koo 2004) reranker to the (Collins 1999) parser yields a 13% relative reduction in error over the base parser. This improvement matches the gains made by the (Charniak 2000) parser, which uses a similar set of features. The equivalent improvements imply that new features should yield the same influence, whether incorporated directly into the parser or appended in a reranker. A reranker, however, is simpler to implement and allows freedom in the choice of features.

2.3 WordNet-Based Approaches

WordNet word senses are applied to the prepositional phrase attachment ambiguity problem in (Stetina and Nagao 1997). Instances of PP-ambiguity are represented as quadruples of words: the preposition, the noun argument of the preposition, and the noun and verb between which the attachment is ambiguous. When evaluating a test quadruple, it will often be the case that the exact quadruple was unseen in the training set. Backed-off models, such as (Collins and Brooks 1995), collect statistics based on lower-order tuples to address this issue. However, each level of backoff obtains progressively lower performance; for instance, (Collins and Brooks 1995) obtained 92.6% performance for full quadruple matches, 87.8% for triples, and so forth.

The approach of (Stetina and Nagao 1997), therefore, is to increase the number of high-order matches by equating unseen test words with words from the training set. For instance, the test quadruple "buy magazines for children" could be matched to the observed quadruple "buy books for children" by equating "books" and "magazines". The criteria that determines when two words can be equated is the *semantic distance* between the senses of the two words, where the semantic distance between two senses is related to the distance between each word sense and their lowest common ancestor in the WordNet hypernymy graph.

The action of our hypernym feature set, described at greater length in Section 4.2.4, resembles the semantic distance metric of (Stetina and Nagao 1997). Nevertheless, our reranker does not actually compute semantic distances, nor does it explicitly attenuate correlations between word senses as their semantic distance increases. However, it is possible for our reranker to simulate the use of a semantic distance measure, since it may learn to pay less attention to features involving higher-level hypernyms during training.

A system that simultaneously parses and performs word sense disambiguation is described in (Bikel 2000). The parser makes use of a generative probability model based on a 1st-order Markov grammar, in the language of Section 2.1. For every rule, the head constituent is generated first, and then left and right modifiers are generated outward. For each constituent, the nonterminal label is created first, followed by the part-of-speech tag of the headword, WordNet word sense of the headword, and actual headword. Note that the word senses are generated *before* the words themselves. Bikel gives the motivation for this decision as a desire for the generative probability model to match the cognitive model of language generation. WordNet word senses correspond to internal concepts, while bare words correspond to external artifacts such as speech or text; therefore word senses should be generated before words. In spite of the added burden of disambiguating word senses, parsing performance was not significantly affected by the introduction of word senses, and moreover, the word sense disambiguation performance was high.

Many aspects of the (Bikel 2000) project are similar to our own: the use of WordNet senses as a means of accessing underlying meanings, the use of sense-to-sense dependencies, the use of WordNet hypernymy to combat data sparseness. However, the project differs from our own in a few important ways. First, instead of creating or modifying a standalone parser, our project employs a reranking approach. In addition, the goal of our reranker is to increase parse performance *only*; word senses are useful inasmuch as they aid us in resolving parse ambiguities, but they are not an aim in themselves. We also train on a corpus that has no word sense-annotations. On the other hand, the (Bikel 2000) project is aimed at creating both word senses and syntactic structures, and the parser is trained on a word sense-annotated corpus.

Finally, high-level semantic tagging of named entities has been explored by (Johnson and Ciaramita 2003). This project introduces the concept of WordNet "supersenses", an idea which we borrow for our supersense feature set, as described in Section 4.2.3. In explanation, every WordNet sense is processed from source material in a lexicographer file. WordNet noun senses originate from 26 such files, each of which pertains to some high-level semantic category. The filenames, therefore, define the supersense categories (a listing of the 26 noun supersenses is given in Table 4.1 on page 52).

Chapter 3

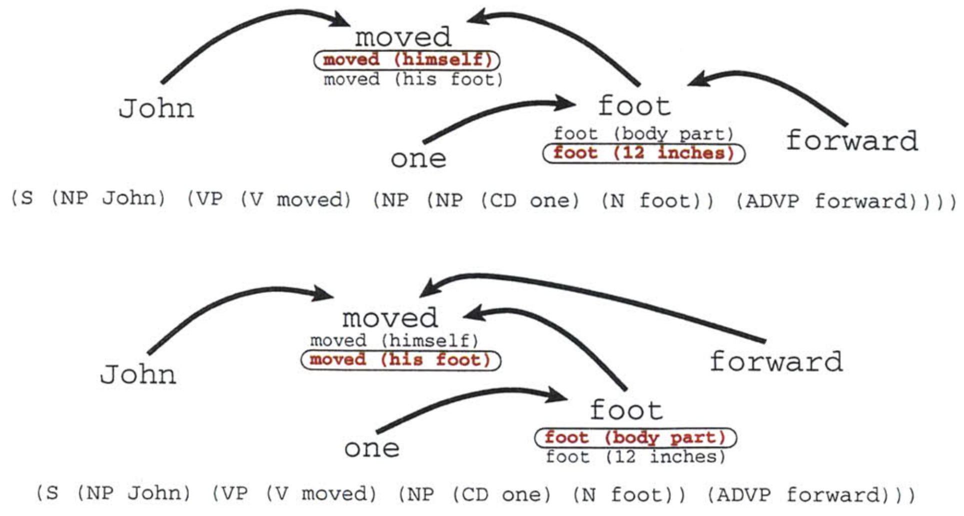
Structure of the Reranking Model

In this chapter, we present our reranking algorithm. We begin by outlining the algorithm at a high level, and in later sections we describe how we obtain an efficient implementation by imposing and exploiting limitations on the feature space. We continue by showing how we optimize our model using stochastic gradient descent, and we finish by describing our method for incorporating the rankings of other models into our reranker.

3.1 High-Level Description

Our hidden variable model is a simple extension of the head-driven statistical models introduced in (Collins 1999) and (Charniak 1997). From each parse tree we produce a dependency tree, which captures the dependency relationships between headwords in the parse. Our main innovation is to treat this dependency tree as a pairwise Markov Random Field (MRF); each word in the dependency tree is given a hidden word sense, and these hidden senses interact along dependency arcs. Figure 3-1 illustrates the interaction of hidden word senses.

Our reranking algorithm is characterized by four elements. First, we make use of a feature vector representation, which reduces each MRF into a high-dimensional vector of feature counts. Second, we use a matched vector of parameters to define a probability distribution over the MRFs. Third, we define a loss function that



```
(S (NP John) (VP (V moved) (NP (NP (CD one) (N foot)) (ADVP forward))))
```

```
(S (NP John) (VP (V moved) (NP (CD one) (N foot)) (ADVP forward)))
```

Figure 3-1: The sentence "John moved one foot forward" gives rise to the two alternate dependency trees above, each of which prefers a different word sense assignment, as circled above. In the top structure, the adjectival attachment of "forward" creates an affinity for the "unit of length" sense of "foot", which in turn promotes the reflexive sense of "moved". In the bottom structure, the adverbial attachment of "forward" has the opposite effect.

approximates the training error of the algorithm. Fourth, we optimize the parameter vector using the gradient of the loss function. The remainder of this section explains each of these four elements in greater detail.

3.1.1 Notation

We begin by defining notation and formally restating the problem. We are given a corpus of m training examples $x_1, \ldots, x_m$, where each example contains of a set of m_i candidate dependency trees $t_{i,1}, \ldots, t_{i,m_i}$. Each candidate tree is scored according to its adherence to the gold standard parse, and in each group of candidates, the highest-scored tree is specially labeled as $t_{i,1}$.

Each word in every tree $t_{i,j}$ holds a hidden word sense. We define tree-wide *assignments* of word senses to all nodes. Let $m_{i,j}$ give the number of possible assignments for tree $t_{i,j}$; let the assignments themselves be $a_{i,j,1}, \ldots, a_{i,j,m_{i,j}}$. Note that $m_{i,j}$ is exponential in the size of the tree; if there are n words, each of which has s_i possible senses, then there are $m_{i,j} = \prod_{i=1}^{n} s_i$ possible sense assignments. Figure 3-2 arranges the various entities associated with a given data example x_i.

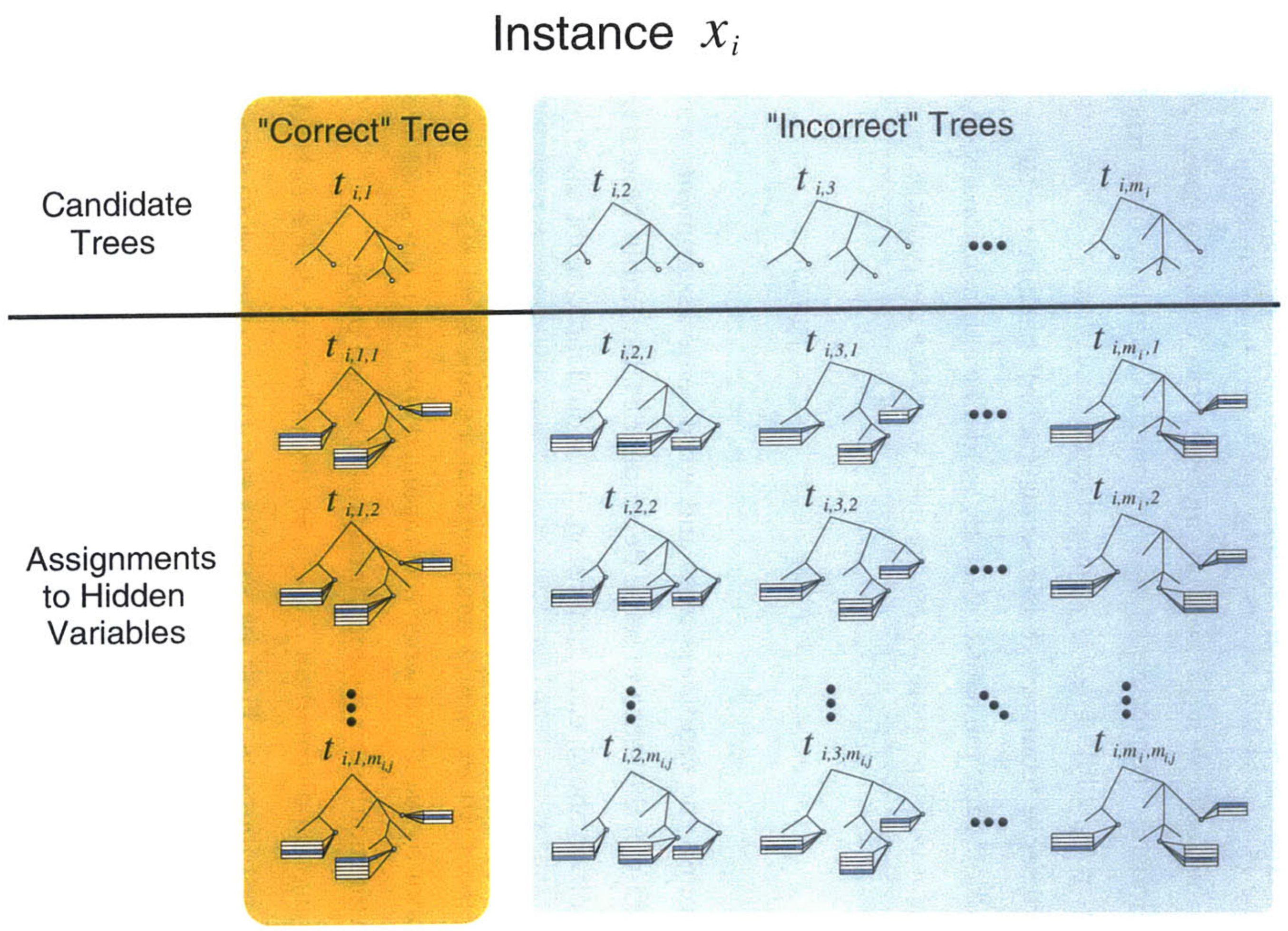


Figure 3-2: A table depicting the elements that make up a single reranking example x_i. The top row enumerates the candidate dependency trees $t_{i,j}$; note that the best-scored tree $t_{i,1}$ is labeled as "correct" tree, while the other trees are labeled "incorrect" trees. The assignments of hidden word senses are enumerated in columns below each candidate tree.

3.1.2 Feature-Based Probability Model

A dependency tree $t_{i,j}$ together with some sense assignment $a_{i,j,k}$ is represented with a high-dimensional feature vector. We define the function $\Phi(i, j, k)$, which gives the feature vector representing tree $t_{i,j}$ with assignment $a_{i,j,k}$. Each dimension of the vector $\Phi(i, j, k)$ contains an occurrence count for some salient structure or relationship. For example, $\Phi_1(i, j, k)$ might count how often rule `S` $\Rightarrow$ `NP VP` appears in $t_{i,j}$, while $\Phi_{100}(i, j, k)$ might count how often `S` $\Rightarrow$ `NP VP` appears with the word sense "company (business institution)" heading the `NP`.

To accompany the feature vectors, we define a dimensionally-matched parameter vector Θ. These parameters are used to induce the probability distribution

$$p(j, k \mid i, \Theta) \quad = \quad \frac{e^{\Phi(i,j,k)\cdot\Theta}}{\sum\limits_{j',k'} e^{\Phi(i,j',k')\cdot\Theta}}$$

which ranks the candidate trees and word sense assignments according to the plausibility of the sense-to-sense dependencies they imply. Note that according to the definition above, each dimension in Θ would indicate the discriminative ability of the related feature. Continuing our example from above, if $\Theta_1 = 0.0000001$ and $\Theta_{100} = 0.5$, then we would conclude that the rule `S` $\Rightarrow$ `NP VP` is not a good discriminator by itself, but when accompanied with the noun sense "company (business institution)", it is a strong indicator of a good parse.

We use the distribution $p(j, k \mid i, \Theta)$ from above to define two additional probability distributions. First, by summing out the hidden word sense assignments, we obtain the distribution

$$p(j \mid i, \Theta) \quad = \quad \sum_k p(j, k \mid i, \Theta)$$

which measures the overall plausibility of dependency tree $t_{i,j}$, independent of any particular sense assignment. Second, by dividing the two previous distributions, we obtain a conditional probability distribution over the word sense assignments for a given dependency tree

$$p(k \mid i, j, \Theta) = \frac{p(j, k \mid i, \Theta)}{p(j \mid i, \Theta)}$$

Finally, before we move on, note that our probability model does not place any *inherent* constraints on the features it can use; that is, our reranking model can accept arbitrarily-defined features. However, in Section 3.2, we will impose an *artificial* constraint on the features; namely, we will restrict the features to involve either a single word sense or a pair of senses involved in a dependency relationship. This new constraint allows us to bring powerful dynamic-programming methods to bear on the problem and thereby achieve an efficient implementation. Nevertheless, for the time being we shall continue to treat our probability model as unconstrained.

3.1.3 Loss Function and Gradient

Ideally, the probability distribution p should satisfy the following property:

$$\forall i,\ \forall j > 1 \qquad p(1 \mid i, \Theta) > p(j \mid i, \Theta)$$

so that the best-scoring candidate tree $t_{i,1}$ is awarded the greatest share of the probability mass. Accordingly, the goal of the training phase of our algorithm is to generate parameters Θ that maximize the probability mass awarded to trees $t_{1,1}, t_{2,1}, \ldots, t_{m,1}$. This subsection captures this goal formally by defining a *loss function* $\mathcal{L}(\Theta)$ that equals the negative log-probability of the best-scored candidate trees:

$$\mathcal{L}(\Theta) = -\sum_i \log p(1 \mid i, \Theta)$$

Substituting our definitions for the various probability distributions yields

$$\begin{aligned}
\mathcal{L}(\Theta) &= -\sum_i \log \sum_k p(1, k \mid i, \Theta) \\
&= -\sum_i \left[\log \sum_k \frac{e^{\Phi(i,1,k)\cdot\Theta}}{\sum\limits_{j',k'} e^{\Phi(i,j',k')\cdot\Theta}} \right] \\
&= \sum_i \left[-\log \left(\sum_k e^{\Phi(i,1,k)\cdot\Theta} \right) + \log \left(\sum_{j',k'} e^{\Phi(i,j',k')\cdot\Theta} \right) \right]
\end{aligned}$$

Our next step is to find an expression for the gradient $\frac{\partial\mathcal{L}(\Theta)}{\partial\Theta}$. For the sake of simplicity, we rewrite $\mathcal{L}$ in terms of functions F_i and G_i as follows:

$$\begin{aligned}
\mathcal{L}(\Theta) &= \sum_i (-F_i(\Theta) + G_i(\Theta)) \\
F_i(\Theta) &= \log \sum_k e^{\Phi(i,1,k)\cdot\Theta} \\
G_i(\Theta) &= \log \sum_{j,k} e^{\Phi(i,j,k)\cdot\Theta}
\end{aligned}$$

The gradient $\frac{\partial F_i(\Theta)}{\partial\Theta}$ is given by

$$\begin{aligned}
\frac{\partial F_i(\Theta)}{\partial\Theta} &= \frac{\partial}{\partial\Theta} \log \sum_k e^{\Phi(i,1,k)\cdot\Theta} \\
&= \frac{\sum\limits_k \Phi(i,1,k) e^{\Phi(i,1,k)\cdot\Theta}}{\sum\limits_{k'} e^{\Phi(i,1,k')\cdot\Theta}} \\
&= \sum_k \Phi(i,1,k) \left(\frac{e^{\Phi(i,1,k)\cdot\Theta}}{\sum\limits_{k'} e^{\Phi(i,1,k')\cdot\Theta}} \right) \\
&= \sum_k \Phi(i,1,k) \left(\frac{e^{\Phi(i,1,k)\cdot\Theta} \Big/ \sum\limits_{q,r} e^{\Phi(i,q,r)\cdot\Theta}}{\sum\limits_{k'} e^{\Phi(i,1,k')\cdot\Theta} \Big/ \sum\limits_{q',r'} e^{\Phi(i,q',r')\cdot\Theta}} \right)
\end{aligned}$$

$$\begin{aligned}
&= \sum_k \Phi(i,1,k)\left(\frac{p(1,k \mid i,\Theta)}{\sum_{k'} p(1,k' \mid i,\Theta)}\right) \\
&= \sum_k \Phi(i,1,k)\left(\frac{p(1,k \mid i,\Theta)}{p(1 \mid i,\Theta)}\right) \\
&= \sum_k \Phi(i,1,k)p(k \mid i,1,\Theta) \\
&= \mathbf{E}_p[\Phi(i,1,k)]
\end{aligned}$$

where $\mathbf{E}_p[\Phi(i,1,k)]$ is the expected feature vector produced by candidate tree $t_{i,1}$ under the probability distribution $p(k \mid i,1,\Theta)$ (and the generalization $\mathbf{E}_p[\Phi(i,j,k)]$ follows immediately). Graphically, $\mathbf{E}_p[\Phi(i,1,k)]$ can be thought of as the weighted average of the feature vectors produced by the leftmost "Correct Tree" column of Figure 3-2. We now turn our attention to $\frac{\partial G_i(\Theta)}{\partial \Theta}$:

$$\begin{aligned}
\frac{\partial G_i(\Theta)}{\partial \Theta} &= \frac{\partial}{\partial \Theta} \log \sum_{j',k'} e^{\Phi(i,j',k')\cdot\Theta} \\
&= \frac{\sum_{j,k} \Phi(i,j,k)e^{\Phi(i,j,k)\cdot\Theta}}{\sum_{j',k'} e^{\Phi(i,j',k')\cdot\Theta}} \\
&= \sum_{j,k} \Phi(i,j,k)\frac{e^{\Phi(i,j,k)\cdot\Theta}}{\sum_{j',k'} e^{\Phi(i,j',k')\cdot\Theta}} \\
&= \sum_{j,k} \Phi(i,j,k)p(j,k \mid i,\Theta) \\
&= \sum_{j,k} \Phi(i,j,k)p(k \mid i,j,\Theta)p(j \mid i,\Theta) \\
&= \sum_j p(j \mid i,\Theta) \sum_k \Phi(i,j,k)p(k \mid i,j,\Theta) \\
&= \sum_j p(j \mid i,\Theta)\mathbf{E}_p[\Phi(i,j,k)]
\end{aligned}$$

Therefore, $G_i(\Theta)$ equals the expected feature vector produced by the *entire* example x_i. Referring back to Figure 3-2 once more, we can think of $G_i(\Theta)$ as the

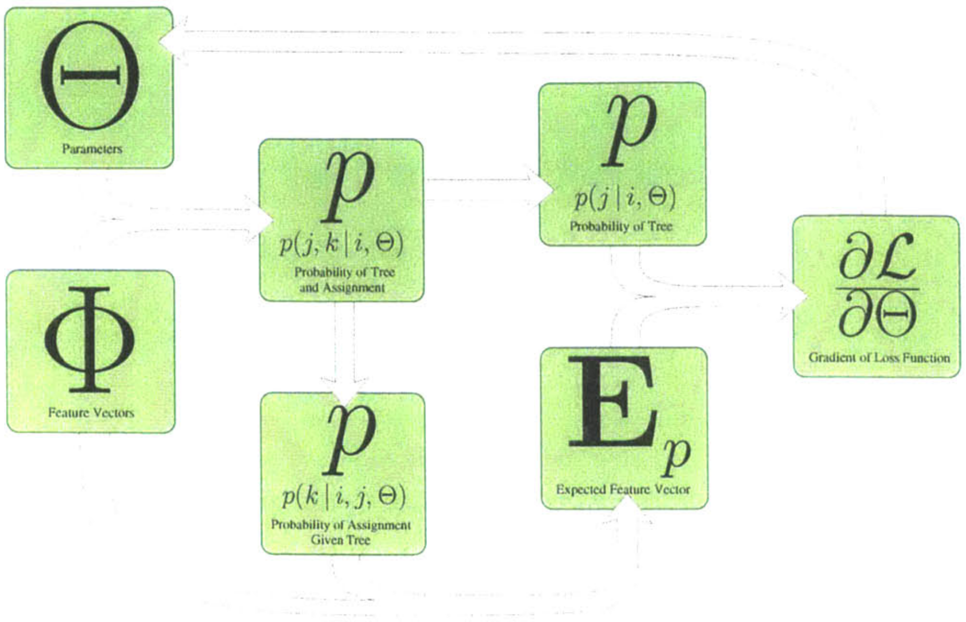


Figure 3-3: A flowchart depicting how the major elements of the reranking algorithm interact with each other.

weighted average of the feature vectors produced by the entire table. Finally, we substitute our partial results back into the expression for $\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta}$:

$$\begin{aligned}\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} &= \sum_i \left(-\frac{\partial F_i(\Theta)}{\partial \Theta} + \frac{\partial G_i(\Theta)}{\partial \Theta} \right) \\ &= \sum_i \left(-\mathbf{E}_p[\Phi(i,1,k)] + \sum_j p(j \mid i,\Theta) \mathbf{E}_p[\Phi(i,j,k)] \right)\end{aligned}$$

Note that $\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta}$ is completely expressed in terms of the two functions $p(j \mid i,\Theta)$ and $\mathbf{E}_p[\Phi(i,j,k)]$. This property figures importantly in Section 3.2, where we will show how p and $\mathbf{E}_p$, and hence the gradient, can be computed efficiently.

3.1.4 Summary

In summary, our reranking algorithm proceeds as follows. We first produce feature vectors from every dependency tree $t_{i,j}$ and sense assignment $a_{i,j,k}$. We initialize the parameters Θ, creating a probability distribution over trees and assignments.

The gradient $\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta}$ allows us to optimize Θ, using an algorithm such as stochastic gradient descent (LeCun et al. 1998) or conjugate gradient descent (M. Johnson and Riezler 1999). The flowchart in Figure 3-3 lays out the high-level operation of the algorithm.

3.2 Incorporating Belief Propagation

The reranking algorithm as described in the previous section contains some glaring inefficiencies. The gradient must be recomputed on each iteration of gradient descent, which means that the algorithm must repeatedly evaluate

$$\mathrm{E}_p[\Phi(i,j,k)] \quad \text{and} \quad p(j \mid i, \Theta) \qquad \forall i,j$$

The formulae for $\mathrm{E}_p[\Phi(i,j,k)]$ and $p(j \mid i, \Theta)$ given in Section 3.1 require an enumeration over all possible assignments to hidden variables, a combinatorially complex task. In addition, feature vectors must be produced for every tree $t_{i,j}$ and sense assignment $a_{i,j,k}$; again, a combinatorially-sized task. To overcome these obstacles, we impose the following constraint on our features:

(i) A feature can involve at most two hidden word senses,

(ii) If a feature involves two senses, these two must be involved in a dependency relationship; that is, they must be linked by an edge in the dependency tree.

In this section, we show how these restrictions allow the use of belief propagation as a module that computes $\mathrm{E}_p[\Phi(i,j,k)]$ and $p(j \mid i, \Theta)$ in linear time (Yedidia et al. 2002). Additionally, the restrictions permit a decomposed feature vector representation that sidesteps the problem of enumerating a combinatorially-sized set of feature vectors.

The remainder of this section is divided into four subsections: the first subsection introduces new notation, the second subsection describes the belief propagation algorithm, the third subsection explains how we use belief propagation to create p and $\mathbf{E}_p$, and the fourth subsection summarizes our modifications to the reranking algorithm.

3.2.1 Notation

We narrow the scope of our analysis to a single dependency tree $t_{i,j}$. Let n be the number of nodes in the tree, and number the nodes $1, 2, \ldots, n$. Let the edges in the tree be given by the set $E_{i,j}$, which is defined such that $(u, v) \in E_{i,j}$ iff $u < v$ and nodes u and v neighbor each other in $t_{i,j}$; therefore, $|E_{i,j}| = n - 1$. For convenience, let $N(u)$ give the set of nodes neighboring u; that is, $v \in N(u)$ iff $(u, v) \in E_{i,j}$ or $(v, u) \in E_{i,j}$.

Next, let the hidden word sense of node u be $s_u \in S_u$, where S_u is the domain of word senses that the word at node u may carry. We also define the constant $S = \max_u |S_u|$, which bounds the size of the word sense domains. We continue to refer to a complete assignment of senses to all nodes with the notation $a_{i,j,k}$, but we define new selector functions $a_u(a_{i,j,k}) \in S_u$ such that $a_u(a_{i,j,k})$ returns the word sense assigned to node u in assignment $a_{i,j,k}$.

The input to belief propagation is a set of node weights $\alpha_u \in S_u \mapsto \mathbb{R}$ and edge weights $\beta_{u,v} \in (S_u \times S_v) \mapsto \mathbb{R}$ such that $\alpha_u(s_u)$ gives a measure of the appropriateness of sense s_u being assigned to the word at node u, and $\beta_{u,v}(s_u, s_v)$ gives a measure of the appropriateness of values s_u and s_v appearing on edge (u, v) in conjunction[1]. These weight functions define the following probability distribution over assignments $a_{i,j,k}$:

$$p_{i,j}\left(a_{i,j,k}\right) = \frac{1}{Z_{i,j}} \prod_{w} \alpha_w(a_w(a_{i,j,k})) \prod_{(u,v)\in E_{i,j}} \beta_{u,v}(a_u(a_{i,j,k}), a_v(a_{i,j,k}))$$

where $Z_{i,j}$ is a normalizing constant that ensures $\sum_k p_{i,j}(a_{i,j,k}) = 1$. Figure 3-4 depicts the functions α_u and $\beta_{u,v}$ for a small sample graph of four nodes.

[1]Note that we require the edge weights to be undirected and defined in both directions, so that $\beta_{u,v}(s_u, s_v) = \beta_{v,u}(s_v, s_u)$ holds. This property is important because the message passing algorithm of Section 3.2.2 will use both $\beta_{u,v}(s_u, s_v)$ or $\beta_{v,u}(s_v, s_u)$ to refer to the same value.

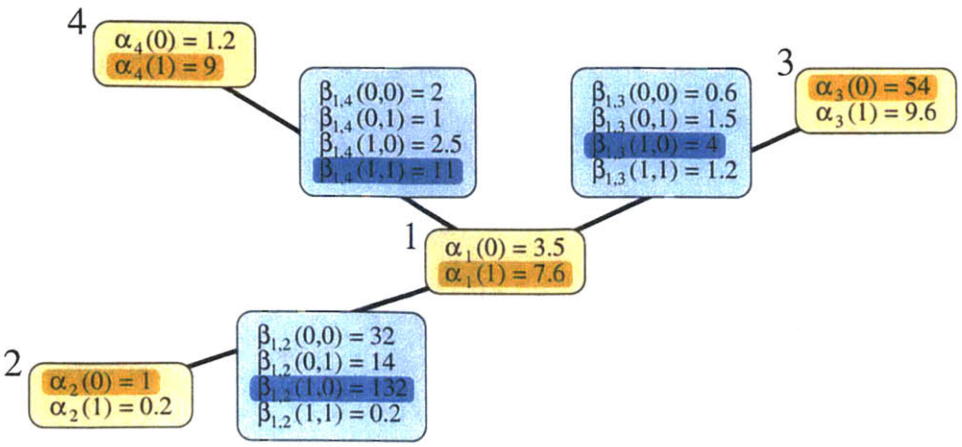


Figure 3-4: An example graph over binary-valued hidden variables in which the values of the weight functions α_u and $\beta_{u,v}$ have been filled in. As the highlighted values indicate, the probability distribution $p_{i,j}(a_{i,j,k})$ generated over this four-node graph would favor the binary sense assignment $a_{i,j,k} = \{s_1 = 1, s_2 = 0, s_3 = 0, s_4 = 1\}$.

The belief propagation algorithm produces three items as output. The first output is a set of node *beliefs* $b_u \in S_u \mapsto \mathbb{R}$ that satisfy the following properties:

1. $b_u(s_u)$ indicates the appropriateness of assigning sense s_u to the word at node u, and

2. $\sum_{s_u \in S_u} b_u(s_u) = 1$

In fact, because we are performing belief propagation on a tree, the node beliefs b_u will actually give the exact marginalized probability distribution at each node u; that is,

$$b_u(x_u) = \sum_{a_{i,j,k} \,|\, a_u(a_{i,j,k}) = x_u} p_{i,j}(a_{i,j,k})$$

where the notation $a_{i,j,k} \,|\, a_u(a_{i,j,k}) = x_u$ restricts the summation to assignments where node u is given word sense x_u; that is, the summation is restricted to the set $\{a_{i,j,k} \,|\, a_u(a_{i,j,k}) = x_u\}$. The second output of belief propagation is a set of pairwise beliefs $b_{u,v} \in (S_u \times S_v) \mapsto \mathbb{R}$ which, in the tree case, give the exact marginalized distributions over pairs of nodes; that is:

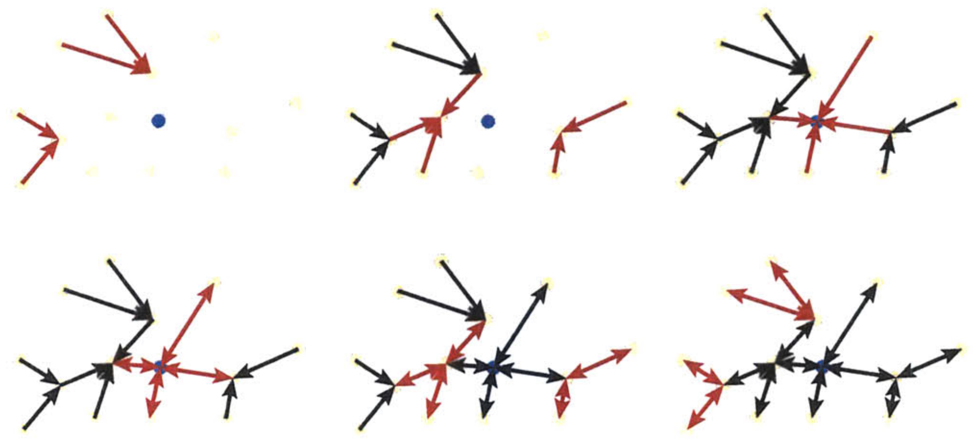


Figure 3-5: The steps of the message-passing algorithm on a small example graph. The dark node is an arbitrarily chosen root, and the messages are shown as arrows.

$$b_{u,v}(x_u, x_v) = \sum_{a_{i,j,k} \,|\, a_u(a_{i,j,k})=x_u, a_v(a_{i,j,k})=x_v} p_{i,j}(a_{i,j,k})$$

where the notation $a_{i,j,k} \,|\, a_u(a_{i,j,k}) = x_u, a_v(a_{i,j,k}) = x_v$ indicates summation over assignments where senses x_u and x_v are assigned to nodes u and v, respectively. Proof that the node and edge beliefs are equal to the marginal probability distributions, in the tree case, can be found in Appendix A.1. Finally, the third output of belief propagation is the normalization constant $Z_{i,j}$ associated with $p_{i,j}(a_{i,j,k})$; this constant is essential, as it allows us to compute $p(j \,|\, i, \Theta)$ efficiently (see Section 3.2.3).

3.2.2 Mechanics of Belief Propagation

The core of belief propagation is the dynamic-programming technique known as the *message passing* algorithm, which allows linear-time[2] computation of all three outputs of belief propagation: node beliefs, pairwise beliefs, and normalization factor. We will describe only the essentials here, leaving the details of a linear-time implementation and the necessary complexity analysis to Appendix A.2.

[2]Time linear in the number of nodes in the tree; the dependence is quadratic in S, the bound on the size of the word sense domains.

In the message passing algorithm, every node u transmits a message $m_{u\to v} \in S_v \mapsto \mathbb{R}$ to each of its neighbors v, where $m_{u\to v}(s_v)$ gives an indication of how strongly node u "thinks" node v should have sense s_v. These messages are determined recursively by the following:

$$m_{u\to v}(s_v) = \sum_{s_u \in S_u} \alpha_u(s_u)\beta_{u,v}(s_u, s_v) \prod_{w\in N(u)\setminus v} m_{w\to u}(s_u)$$

so that the message from u to v is a combination of the messages received from u's other neighbors (that is, $w \in N(u) \setminus v$). In the case of a tree, the following scheme suffices to calculate the messages: pick an arbitrary root node, then compute messages from the leaves in towards the root and then from the root back out to the leaves. To see why, consider a tree in which no messages have been calculated. Each leaf node ℓ has only one neighbor: its parent node p. Therefore, the set $w \in N(\ell) \setminus p$ is empty, and the message from ℓ to p can be created immediately, without waiting for any incoming messages. Once all of the leaf messages have been computed, the parent nodes p can compute their own upstream messages, and so forth. Figure 3-5 depicts the full upstream-downstream process for an example tree. After the messages have been computed, the node beliefs b_u and pairwise beliefs $b_{u,v}$ are given by

$$\begin{aligned} b_u(s_u) &= \frac{1}{z_u}\alpha_u(s_u) \prod_{v\in N(u)} m_{v\to u}(s_u) \\ b_{u,v}(s_u, s_v) &= \frac{1}{z_{u,v}}\alpha_u(s_u)\alpha_v(s_v)\beta_{u,v}(s_u, s_v) \prod_{s\in N(u)\setminus v} m_{s\to u}(s_u) \prod_{t\in N(v)\setminus u} m_{t\to v}(s_v) \end{aligned}$$

where z_u and $z_{u,v}$ are normalizing factors which ensure that

$$\sum_{s_u\in S_u} b_u(s_u) = 1 \qquad \text{and} \qquad \sum_{s_u\in S_u,\, s_v\in S_v} b_{u,v}(s_u, s_v) = 1$$

We now turn our attention to the third output of belief propagation: the normalization factor $Z_{i,j}$. Fortunately, it turns out that

$$\begin{aligned} \forall u \quad z_u &= Z_{i,j} \\ \forall (u,v) \in E_{i,j} \quad z_{u,v} &= Z_{i,j} \end{aligned}$$

so that any of the node or pairwise normalization constants can serve as the tree-wide normalization constant. This equivalence is a side effect of the proof that tree beliefs are equal to marginal probabilities; see the end of Appendix A.1, page 75.

3.2.3 Computing p and E_p

The inputs to belief propagation must be carefully prepared so that its outputs give rise to $p(j \mid i, \Theta)$ and $\mathrm{E}_p[\Phi(i,j,k)]$. Recall that our features are restricted to either single word senses or pairs of senses joined by a dependency. Therefore, we can decompose the tree-wide feature vector $\Phi(i,j,k)$ into a set of node feature vectors $\phi_u(s_u)$ and pairwise feature vectors $\phi_{u,v}(s_u, s_v)$:

$$\Phi(i,j,k) = \left(\sum_u \phi_u(a_u(a_{i,j,k})) \right) + \left(\sum_{(u,v) \in E_{i,j}} \phi_{u,v}(a_u(a_{i,j,k}), a_v(a_{i,j,k})) \right)$$

where $a_u(a_{i,j,k})$, as before, gives the word sense assigned to node u by assignment $a_{i,j,k}$. If $S = \max_u |S_u|$ bounds the size of the word sense domains, then there are only $O(nS + nS^2) = O(nS^2)$ decomposed feature vectors $\phi_u(s_u)$ and $\phi_{u,v}(s_u, s_v)$, while the number of full feature vectors $\Phi(i,j,k)$ is $O(S^n)$. We use the decomposed feature vectors to define the node and pairwise weight functions[3]:

$$\begin{aligned} \alpha_u(s_u) &= e^{\phi_u(s_u) \cdot \Theta} \\ \beta_{u,v}(s_u, s_v) &= e^{\phi_{u,v}(s_u, s_v) \cdot \Theta} \end{aligned}$$

[3]Note that for notational compatibility with $\beta_{u,v}$, we require that $\phi_{u,v}(s_u, s_v) = \phi_{v,u}(s_v, s_u)$

so that the probability distribution $p_{i,j}(a_{i,j,k})$ can be rewritten as follows:

$$\begin{aligned}
p_{i,j}(a_{i,j,k}) &= \frac{1}{Z_{i,j}} \prod_{w} e^{\phi_w(a_w(a_{i,j,k}))\cdot\Theta} \prod_{(u,v)\in E_{i,j}} e^{\phi_{u,v}(a_u(a_{i,j,k}),a_v(a_{i,j,k}))\cdot\Theta} \\
&= \frac{1}{Z_{i,j}} e^{\left(\sum_w \phi_w(a_w(a_{i,j,k}))\right)\cdot\Theta} e^{\left(\sum_{(u,v)\in E_{i,j}} \phi_{u,v}(a_u(a_{i,j,k}),a_v(a_{i,j,k}))\right)\cdot\Theta} \\
&= \frac{1}{Z_{i,j}} e^{\Phi(i,j,k)\cdot\Theta}
\end{aligned}$$

From the role of $Z_{i,j}$ as a normalizer, we can infer that $Z_{i,j} = \sum_k e^{\Phi(i,j,k)\cdot\Theta}$, and this new definition of $Z_{i,j}$ allows us to compute $p(j \mid i, \Theta)$ in $O(1)$ time per candidate tree:

$$\begin{aligned}
Z_i &= \sum_j Z_{i,j} &= \sum_{j,k} e^{\Phi(i,j,k)\cdot\Theta} \\
p(j \mid i, \Theta) &= \frac{\sum_k e^{\Phi(i,j,k)\cdot\Theta}}{\sum_{j',k'} e^{\Phi(i,j',k')\cdot\Theta}} &= \frac{Z_{i,j}}{Z_i}
\end{aligned}$$

which is a considerable improvement over the $O(S^n)$ time that a naive implementation would require. Note that this definition of $Z_{i,j}$ also implies that

$$p_{i,j}(a_{i,j,k}) = \frac{1}{Z_{i,j}} e^{\Phi(i,j,k)\cdot\Theta} = p(k \mid i, j, \Theta)$$

Next, we consider the quantity $\mathbf{E}_p[\Phi(i,j,k)]$. Beginning from its original definition, we replace $p(k \mid i, j, \Theta)$ with the equivalent $p_{i,j}(a_{i,j,k})$ and substitute in the decomposed feature vectors:

$$
\begin{aligned}
&\mathbf{E}_p[\Phi(i,j,k)] \\
&= \sum_k p(k \mid i,j,\Theta)\Phi(i,j,k) \\
&= \sum_k p_{i,j}(a_{i,j,k}) \left(\left(\sum_w \phi_w(a_w(a_{i,j,k})) \right) + \left(\sum_{(u,v)\in E_{i,j}} \phi_{u,v}(a_u(a_{i,j,k}), a_v(a_{i,j,k})) \right) \right) \\
&= \left(\sum_w \sum_k p_{i,j}(a_{i,j,k})\phi_w(a_w(a_{i,j,k})) \right) + \\
&\quad \left(\sum_{(u,v)\in E_{i,j}} \sum_k p_{i,j}(a_{i,j,k})\phi_{u,v}(a_u(a_{i,j,k}), a_v(a_{i,j,k})) \right)
\end{aligned}
$$

Note that $\phi_w(s_w)$ is only sensitive to the word sense s_w, so for the set of assignments $\{a_{i,j,k} \mid a_w(a_{i,j,k}) = x_w\}$, the value of $\phi_w(s_w)$ will be fixed at $\phi_w(x_w)$. Accordingly, we break the set of possible assignments into equivalence classes for which the sense s_w remains constant, giving rise to the following simplification:

$$
\begin{aligned}
\sum_w \sum_k p_{i,j}(a_{i,j,k})\phi_w(a_w(a_{i,j,k})) &= \sum_w \sum_{x_w \in S_w} \phi_w(x_w) \left(\sum_{a_{i,j,k} \mid a_w(a_{i,j,k})=x_w} p_{i,j}(a_{i,j,k}) \right) \\
&= \sum_w \sum_{x_w \in S_w} \phi_w(x_w) b_w(x_w)
\end{aligned}
$$

and similar reasoning applies to the pairwise feature vectors:

$$
\begin{aligned}
&\sum_{(u,v)\in E_{i,j}} \sum_k p_{i,j}(a_{i,j,k})\phi_{u,v}(a_u(a_{i,j,k}), a_v(a_{i,j,k})) \\
&\quad= \sum_{(u,v)\in E_{i,j}} \sum_{x_u \in S_u,\, x_v \in S_v} \phi_{u,v}(x_u, x_v) \sum_{a_{i,j,k} \mid a_u(a_{i,j,k})=x_u,\, a_v(a_{i,j,k})=x_v} p_{i,j}(a_{i,j,k}) \\
&\quad= \sum_{(u,v)\in E_{i,j}} \sum_{x_u \in S_u,\, x_v \in S_v} \phi_{u,v}(x_u, x_v) b_{u,v}(x_u, x_v)
\end{aligned}
$$

Therefore, if there are n nodes in the tree and the size of the word sense domains is bounded by S, the expected feature vector $\mathbf{E}_p$ can be computed in $O(nS^2)$ time by the following expression:

$$\begin{aligned}\mathbf{E}_p[\Phi(i,j,k)] &= \left(\sum_{w}\sum_{x_w\in S_w}\phi_w(s_w)b_w(s_w)\right) + \\ &\quad \left(\sum_{(u,v)\in E_{i,j}}\sum_{x_u\in S_u,\, x_v\in S_v}\phi_{u,v}(s_u,s_v)b_{u,v}(s_u,s_v)\right)\end{aligned}$$

Once again, this is a vast improvement over the $O(S^n)$ time required by a naive algorithm.

3.2.4 Summary

In summary, we have seen how restricting features to pairs of senses allows us to apply the belief propagation algorithm, a powerful dynamic-programming technique. Whereas a naive algorithm would require time exponential in the size of the dependency trees, belief propagation requires only linear time. The flowchart in Figure 3-6 updates Figure 3-3 to show how belief propagation replaces several items in the high-level algorithm.

3.3 Stochastic Gradient Descent

We optimize our model using a form of stochastic gradient descent (LeCun et al. 1998). We initialize the parameters Θ to small randomized values, in order to break symmetries that would trap the model in metastable local optima. We then iterate through the training corpus, and for each example x_i we calculate the gradient arising from that example alone:

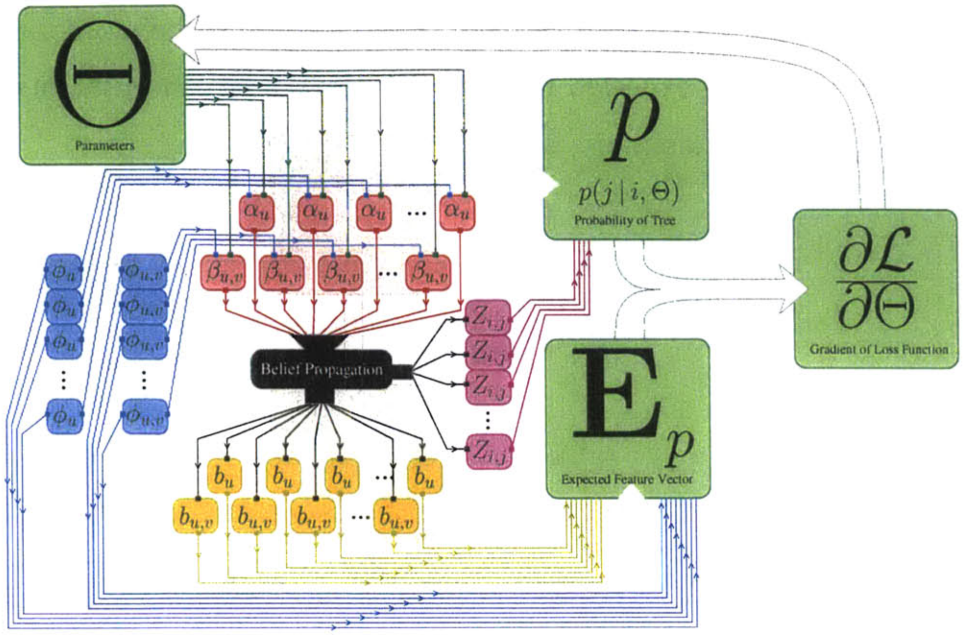


Figure 3-6: A revised version of the flowchart in Figure 3-3 that depicts use of belief propagation in the implementation of the reranking algorithm. Several high-level elements from the original flowchart have been overwritten with belief propagation items. Although these high-level elements no longer exist in the implementation, they still exist in the abstract, implicitly defined by the belief propagation machinery that replaces them. For instance, the feature vector Φ is decomposed into partial feature vectors ϕ_u and $\phi_{u,v}$, the distribution $p(j, k \mid i, \Theta)$ is represented by the weight functions α_u and $\beta_{u,v}$, and the conditional distribution $p(k \mid i, j, \Theta)$ is represented by the beliefs b_u and $b_{u,v}$.

$$\frac{\partial \mathcal{L}_i(\Theta)}{\partial \Theta} = -\mathbf{E}_p[\Phi(i,1,k)] + \sum_j p(j \mid i, \Theta)\mathbf{E}_p[\Phi(i,j,k)]$$

The parameters are then perturbed a small amount along the direction of $\frac{\partial \mathcal{L}_i(\Theta)}{\partial \Theta}$, where the magnitude of the perturbation is determined by a learning rate η:

$$\begin{aligned} \eta(t) &= \frac{a_0}{1+ct} \\ \Theta &\leftarrow \Theta - \eta(mp+i)\frac{\partial \mathcal{L}_i(\Theta)}{\partial \Theta} \end{aligned}$$

Note that we move in the opposite direction of the gradient, as we wish to minimize $\mathcal{L}(\Theta)$. The learning rate $\eta(t)$ decays as t, the number of examples that have been processed, increases. The quantity $mp+i$ above gives the number of examples processed when the i^{th} example is encountered on the p^{th} pass through the training set, and a_0 and c are parameters of the learning rate, which are chosen through validation on a development set.

In order to confirm that stochastic gradient descent is a viable optimization algorithm for parse reranking, we performed experiments using the features selected by the (Michael Collins and Terry Koo 2004) boosting reranker. The results of optimization through stochastic gradient descent were indistinguishable from the results of the boosting optimization.

3.4 Incorporating Other Rankings

In the preceding descriptions of the reranking algorithm, we have always discussed the reranker as a stand-alone entity. However, we have found it useful to integrate our reranker with the rankings of other models. The remainder of this section describes how we integrate the rankings of the (Collins 1999) base parser and the (Michael Collins and Terry Koo 2004) boosting reranker with our own reranker.

3.4.1 Integrating the Base Parser

The base parser's probability model creates an initial ranking over the candidate parses. Although our reranker replaces that initial ranking, by no means is the ranking completely discarded; our aim in reranking is to supplement, rather than supplant, the original base model. We establish a special per-tree feature $\phi_{\text{logp}}(i,j)$, which simply gives the log-probability assigned by the base parser to tree $t_{i,j}$. This feature is assigned a parameter θ_{logp}, and our probability model and loss function are adjusted to include this new feature as follows:

$$
\begin{aligned}
p(j \mid i, \Theta, \theta_{\text{logp}}) &= \frac{e^{\phi_{\text{logp}}(i,j)\theta_{\text{logp}}} \sum\limits_{k} e^{\Phi(i,j,k)\cdot\Theta}}{\sum\limits_{j'} e^{\phi_{\text{logp}}(i,j')\theta_{\text{logp}}} \sum\limits_{k'} e^{\Phi(i,j',k')\cdot\Theta}} \\
\mathcal{L}(\Theta, \theta_{\text{logp}}) &= -\sum_{i} \log p(1 \mid i, \Theta, \theta_{\text{logp}})
\end{aligned}
$$

Unlike the normal features, however, we do not optimize the parameter θ_{logp} during stochastic gradient descent. Intuitively, the polarity of the log-probability feature will change frequently from example to example: in examples where the base parser's ranking is correct, the gradient $\frac{\partial \mathcal{L}_i(\Theta,\theta_{\text{logp}})}{\partial \theta_{\text{logp}}}$ will be strongly positive, but when the base parser makes a mistake, the gradient $\frac{\partial \mathcal{L}_i(\Theta,\theta_{\text{logp}})}{\partial \theta_{\text{logp}}}$ will be strongly negative. Since $\phi_{\text{logp}}(i,j)$ is present in every candidate tree, it is a powerful feature, and the back-and-forth oscillations in the value of θ_{logp} would hinder the training process. Experimental evidence (Brian Roark et al. 2004a,b) confirms this intuition. Therefore, in our experiments we initialize θ_{logp} to a static value before training and keep it constant throughout. The value of θ_{logp} which we used in our final testing was chosen through validation on the development test set.

3.4.2 Integrating the Boosting Reranker

A good point of comparison for our reranking model is the (Michael Collins and Terry Koo 2004) boosting reranker. So that we can accurately gauge the amount of additive improvement our reranker offers over the boosting reranker, we incorporate the features selected by the boosting reranker into our own reranker. Like the base parser's log-probability, the boosting reranker's features are per-tree features that are insensitive to word sense assignments. Let $\Phi_{\text{boost}}(i,j)$ give the boosting feature vector for tree $t_{i,j}$, and let Θ_{boost} be the matched vector of parameters. We incorporate the boosting features into our probability model and loss function as follows:

$$\begin{aligned}
p(j \mid i, \Theta, \Theta_{\text{boost}}, \theta_{\text{logp}}) &= \frac{e^{\phi_{\text{logp}}(i,j)\theta_{\text{logp}}} e^{\Phi_{\text{boost}}(i,j)\cdot\Theta_{\text{boost}}} \sum_k e^{\Phi(i,j,k)\cdot\Theta}}{\sum_{j'} e^{\phi_{\text{logp}}(i,j')\theta_{\text{logp}}} e^{\Phi_{\text{boost}}(i,j')\cdot\Theta_{\text{boost}}} \sum_{k'} e^{\Phi(i,j',k')\cdot\Theta}} \\
\mathcal{L}(\Theta, \Theta_{\text{boost}}, \theta_{\text{logp}}) &= \sum_i \log p(1 \mid i, \Theta, \Theta_{\text{boost}}, \theta_{\text{logp}})
\end{aligned}$$

and the example-wise gradient $\frac{\partial \mathcal{L}_i(\Theta, \Theta_{\text{boost}}, \theta_{\text{logp}})}{\partial \Theta_{\text{boost}}}$ is given by

$$\frac{\partial \mathcal{L}_i(\Theta, \Theta_{\text{boost}}, \theta_{\text{logp}})}{\partial \Theta_{\text{boost}}} = -\Phi_{\text{boost}}(i,1) + \sum_j p(j \mid i, \Theta, \Theta_{\text{boost}}, \theta_{\text{logp}}) \Phi_{\text{boost}}(i,j)$$

We took two approaches to optimizing the boosting parameters. First, we attempted to optimize Θ and Θ_{boost} simultaneously. To accomplish this, we altered our stochastic gradient descent algorithm so that on the i^{th} example of the p^{th} pass, it performed:

$$\begin{aligned}
\Theta &\leftarrow \Theta - \eta(mp+i) \frac{\partial \mathcal{L}_i(\Theta, \Theta_{\text{boost}}, \theta_{\text{logp}})}{\partial \Theta} \\
\Theta_{\text{boost}} &\leftarrow \Theta_{\text{boost}} - \eta_{\text{boost}}(mp+i) \frac{\partial \mathcal{L}_i(\Theta, \Theta_{\text{boost}}, \theta_{\text{logp}})}{\partial \Theta_{\text{boost}}}
\end{aligned}$$

Note that we applied different learning rates to each parameter vector. From our experiences in training each set of features in isolation, it was clear that the boosting features performed best with a more aggressive learning rate than the newer features[4]. Therefore, when optimizing both sets of parameters simultaneously, we decided to keep the learning rates separate.

Unfortunately, selecting parameters for the two learning rates proved to be a

[4]In fact, even among the various types of new features we experimented with (these are described further in Section 4.2), each feature set required a different learning rate.

difficult task. Simply reusing the learning rates that worked best for each set of features in isolation produced poor results; we presume that interactions between the two feature sets were invalidating the old learning rates. However, with two independent learning rates, exploring the possible space of learning rate parameters became prohibitively expensive.

Accordingly, we turned to a second, simpler integration method. We trained each reranker in isolation and then combined the two rankings with a weighted average. Therefore, our probability model effectively became

$$p(j \mid i, \Theta, \Theta_{\text{boost}}, \theta_{\text{logp}}) = \frac{e^{\phi_{\text{logp}}(i,j)\theta_{\text{logp}}} e^{C_{\text{boost}}\Phi_{\text{boost}}(i,j)\cdot\Theta^*_{\text{boost}}} \sum_k e^{C\Phi(i,j,k)\cdot\Theta^*}}{\sum_{j'} e^{\phi_{\text{logp}}(i,j')\theta_{\text{logp}}} e^{C_{\text{boost}}\Phi_{\text{boost}}(i,j')\cdot\Theta^*_{\text{boost}}} \sum_{k'} e^{C\Phi(i,j',k')\cdot\Theta^*}}$$

where Θ^* and Θ^*_{boost} are the optimized parameters from the isolated training runs, and C and C_{boost} are the parameters of the weighted average. The particular Θ^*, Θ^*_{boost}, C, and C_{boost} that we used in our final testing were chosen by validation on the development data set.

Chapter 4

Experimental Setup and Results

This chapter describes the experiments we conducted and presents the results achieved by our reranker. We begin by mentioning our data sets and continue with descriptions of the feature sets we made use of. We finish by presenting our test results and suggesting a number of factors that may have hindered performance.

4.1 Data Sets

The data sets consist of parsed output produced by the (Collins 1999) parser on the Penn Treebank, with an average of roughly 30 candidate parses per training example. Section 23 was held out as a final test set, and the remainder of the sentences were divided into a training corpus of 35,540 examples and a development-test corpus of 3,712 examples.

These training, devtest, and final test corpuses are the same that were used to produce the (?) boosting reranker. In order to measure how much improvement our own reranker can provide beyond the boosting reranker, lists of the features that the boosting reranker used during training, development testing, and final testing were obtained. There were a total of 11,673 distinct boosting features with an average of roughly 40 features per tree. The next section discusses how we used these features in combination with our own reranker.

4.2 Feature Sets

We trained and tested our reranker using several different feature sets. All of our feature sets, however, make use of WordNet (Miller et al. 1993) and have the same basic composition. The remainder of this section first describes the basic structure of our feature sets and discusses some of the problems of this basic model. We then describe each feature set, explaining how it addresses particular problems with the basic model.

4.2.1 Basic Feature Composition

Each feature set is divided into two kinds of features: single-sense features that are aimed at establishing a prior probability distribution over word sense assignments, and pairwise features that attempt to capture head-modifier relationships between word senses. We currently only retrieve WordNet word senses for nouns; we discuss some of the implications of this decision later on in Section 4.3.3.

Often, there may not be any WordNet word senses available: the word might not be a noun, or it might not appear in WordNet's noun index. When we fail to obtain a word sense, we simply substitute the bare word for the missing sense and treat the node as having a single word sense. Each node feature is a tuple consisting of four elements:

Word	The bare word at that node
Sense	The word sense assigned to the word.
POS	The part-of-speech tag of the word.
Label	The nonterminal label that the word receives as it modifies its target; i.e. the label of the highest nonterminal to which this word propagates as a headword.

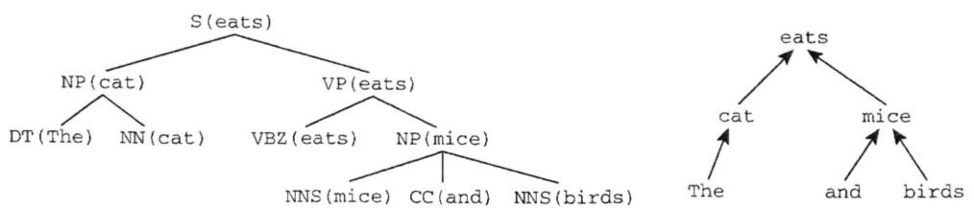


Figure 4-1: A sample parse tree and the dependency tree derived from it.

For example, in Figure 4-1, the dependency node "cat" would have Word "cat", Sense "cat (feline)", POS NN, and Label NP, since the word "cat" propagates up as a headword until the NP. On the other hand, the node "eats" would have Label S, since "eats" propagates all the way to the top of the phrase structure tree.

Including the word as well as the sense gives our reranker a means of manipulating the prior probability distribution over word senses. The POS and Label are also included as they can sometimes influence the word sense distribution. For instance, consider noun part of speech tags, which indicate plurality. The words "sense" and "senses": the senses of "sense" as in "the five senses" or "word senses" are likely to be used in singular or plural, but the senses of "sense" as in "common sense" or "sense of security" will generally only be found in the singular.

The node feature 4-tuples can be quite specific, so we implement several levels of backoff. In particular, for every node u and word sense s_u, we produce the following node features:

$$\phi_u(s_u) = \left\{ \begin{array}{llll} (\text{Word}_u, & s_u, & \text{POS}_u, & \text{Label}_u) \\ (\text{Word}_u, & s_u, & \text{POS}_u &) \\ (\text{Word}_u, & s_u, & & \text{Label}_u) \\ (\text{Word}_u, & s_u & &) \\ (& s_u, & \text{POS}_u, & \text{Label}_u) \\ (& s_u, & \text{POS}_u &) \\ (& s_u, & & \text{Label}_u) \end{array} \right\}$$

Recall that when a word sense is nonexistent, we substitute the bare word for the sense. Therefore, when the word sense is unavailable, we would not generate the first four of the features above, as they would redundantly include the bare word twice: once as Word_u and again as s_u. Moving on, the pairwise features are tuples consisting of the following elements:

Modifier Sense	The word sense of the modifier in the dependency relationship.
Modifier POS	The part-of-speech tag of the modifier.
Head Sense	The word sense of the head in the dependency relationship.
Head POS	The part-of-speech tag of the head.
Production Label	The nonterminal label of the constituent produced by this head-modifier interaction.
Modifier Label	The nonterminal label of the head.
Head Label	The nonterminal label of the modifier.
Dominates Conjunction	True if the Production Label dominates a word with POS tag CC.
Adjacency	True if the modifier's constituent neighbors the head's constituent.
Left/Right	Whether the modifier is on the left or right of the head.

For example, referring to Figure 4-1 again, the dependency between "cat" and "ate" would be assigned Head Label VP, Modifier Label NP, and Production Label S, because when the NP headed by "cat" and the VP headed by "ate" meet, the VP is chosen as the head, and the constituent they produce is an S. Since the NP appears directly to the left of the VP, the dependency would also have Adjacency equal to true and Left/Right equal to Left; however, as the Produced Label S does not dominate a CC, the Dominates Conjunction is set to false. On the other hand, the dependency between "birds" and "mice" would yield Adjacency equal to false, Left/Right equal to Left, and Dominates Conjunction equal to true.

Note that the triple of nonterminal labels provides information about the kind of dependency. For instance, a subject-verb relationship would be reflected by the triple (PLbl, MLbl, HLbl) = (S, NP, VP) while the argument of a transitive verb might produce the triple (VP, NP, VBZ). The additional information provided by the Adjacency, Left/Right, and Dominates Conjunction values help to further determine the type of dependency relationship. As with the node features, the pairwise features can be overly specific, so we generate backed-off features by first removing the two part of speech tags, and then removing the three binary-valued elements. We also generate features where the either the head or modifier sense is removed, leaving behind only the part of speech tag. The full suite of pairwise features is:

$$\phi_{u,v}(s_u, s_v) = \left\{ \begin{array}{llllllllll} \big(s_u, & \text{POS}_u, & s_v, & \text{POS}_v, & \text{PLbl}, & \text{MLbl}, & \text{HLbl}, & \pm\text{CC}, & \pm\text{ADJ}, & \text{L/R}\big) \\ \big(s_u, & & s_v, & & \text{PLbl}, & \text{MLbl}, & \text{HLbl}, & \pm\text{CC}, & \pm\text{ADJ}, & \text{L/R}\big) \\ \big(s_u, & & s_v, & & \text{PLbl}, & \text{MLbl}, & \text{HLbl} & & & \big) \\ \big(s_u, & & & \text{POS}_v, & \text{PLbl}, & \text{MLbl}, & \text{HLbl}, & \pm\text{CC}, & \pm\text{ADJ}, & \text{L/R}\big) \\ \big(s_u, & & & \text{POS}_v, & \text{PLbl}, & \text{MLbl}, & \text{HLbl} & & & \big) \\ \big(& \text{POS}_u, & s_v, & & \text{PLbl}, & \text{MLbl}, & \text{HLbl}, & \pm\text{CC}, & \pm\text{ADJ}, & \text{L/R}\big) \\ \big(& \text{POS}_v, & s_v, & & \text{PLbl}, & \text{MLbl}, & \text{HLbl} & & & \big) \end{array} \right\}$$

4.2.2 Issues with the Basic Model

First of all, there is the issue of choosing which senses to use when producing features. WordNet provides an index that maps words to sets of senses; we call these directly-mapped senses the "literal" word senses for a word. Unfortunately, these literal senses are much too fine-grained for our usage, even with the aid of backed-off features. Consider the word "chocolate", to which WordNet assigns the following three senses:

1. A beverage prepared from cocoa, milk, and sugar, as in "hot chocolate",

2. A solid substance made from roasted ground cacao beans, as in "chocolate bar" or "chocolate chips", and

3. A deep brown color.

The literal senses are clearly informative, but they provide far too much specificity; given the limited size of our training set, it is difficult to imagine features with literal senses being any more informative or less sparse than plain lexicalized features.

Our intuition is that the important information lies somewhere above these lowest-level senses. While the knowledge that "chocolate" can refer to a chocolate-based beverage, chocolate-based solid food, and chocolate-colored color is almost useless, knowing that "chocolate" can refer to a beverage, solid food, or color in general is quite powerful. We might imagine our reranker learning, for instance, that the verb "eat" has an affinity for "solid food" arguments, while "drink" has an affinity for "beverage" arguments. The next two subsections describe our attempts to recover this kind of knowledge using WordNet supersenses and hypernyms.

Another issue, though one that we have begun to address only recently, is the interposition of function words at key points in the dependency tree. Most notably, the headword of a prepositional phrase is chosen as the preposition. Consider Figure 4-2, which displays a dependency tree containing a PP-attachment ambiguity.

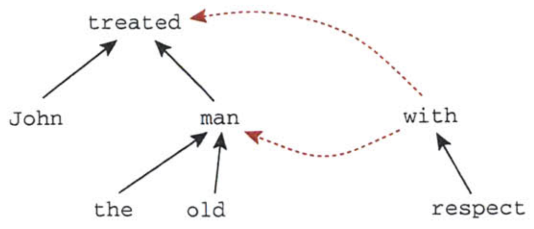


Figure 4-2: A dependency tree for the sentence "John treated the old man with respect."

Note that the preposition "with" interposes between its noun argument "respect" and the potential targets "treated" and "man". The function word "with" is not a noun, so it is only given a single sense, and as we explained in Section 3.1 our features are restricted to pairs of neighboring word senses. Therefore, no information can flow through the interposing node "with" in our hidden variable model, and the PP argument "respect" is unable to affect the PP-attachment preference at all. In fact, under our current model, the prepositional phrase "with respect" would be treated no differently than the phrase "with arthritis", which has the opposite attachment preference. Many studies (Ratnaparkhi and Roukos 1994; Collins and Brooks 1995) have shown that the PP argument is essential in resolving PP-attachment ambiguities.

We explored two possibilities for resolving this problem. First, we ran preliminary experiments in which each preposition was given two word senses. Variation of this binary word sense allows a measure of information to pass through the preposition, so that the PP argument can exert some influence over the attachment preferences. Our experiments showed that this method yielded some increase in performance; however, the gains were by no means large. Recently, we have tried a second approach that involves transforming the dependency tree to bring the PP argument into direct contact with the target that the PP attaches to. A fuller description of our efforts in this direction is left to Section 5.2.5 in the next chapter.

4.2.3 Supersenses

As we mentioned earlier, one of the issues with our basic feature model is the over-specified nature of literal word senses. One method by which we access higher-

`noun.Tops`	`noun.act`	`noun.animal`
`noun.artifact`	`noun.attribute`	`noun.body`
`noun.cognition`	`noun.communication`	`noun.event`
`noun.feeling`	`noun.food`	`noun.group`
`noun.location`	`noun.motive`	`noun.object`
`noun.person`	`noun.phenomenon`	`noun.plant`
`noun.possession`	`noun.process`	`noun.quantity`
`noun.relation`	`noun.shape`	`noun.state`
`noun.substance`	`noun.time`	

Table 4.1: A listing of the 26 WordNet noun lexicographer filenames, which we use as "supersenses."

level information is through the use of WordNet "supersenses", which are described in greater detail in Section 2.3. Briefly, WordNet supersenses categorize word senses into broad, top-level semantic categories; the full set of 26 noun supersenses is given in Table 4.1.

For every word, we first retrieve its literal senses, and from each sense we derive its supersense, discarding the original senses. For example, the noun "crate" has two literal senses: a box-like object ("a wooden crate"), or the quantity held in a crate ("a crate of toys"), which give rise to the supersenses `noun.artifact` and `noun.quantity`. However, note that this process may reduce the number of senses the word is assigned. For instance, the noun "car" has five literal senses, but all five are members of `noun.artifact`, so we treat "car" as having only a single word sense.

The small number of supersenses yield a fairly compact feature set, making for more dependable training and generalization. However, the coarse nature of the supersenses means that each supersense carries less information than a more specific word sense might.

4.2.4 Hypernyms

A second way in which we access higher-level information is through the use of WordNet *hypernymy*. Beyond the word to sense index and the supersenses, WordNet contains a great deal of information about the relationships between word

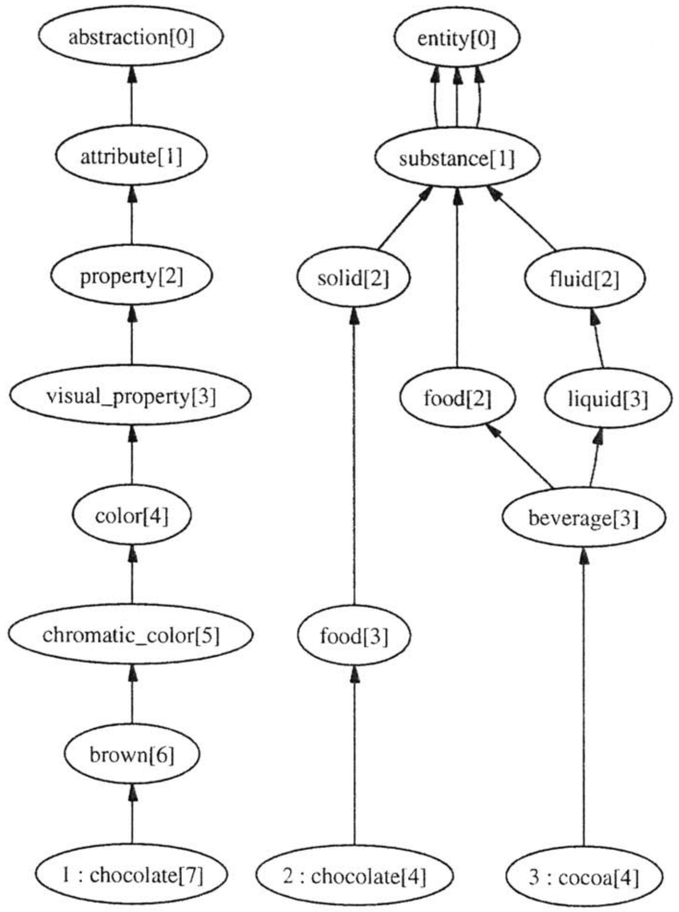


Figure 4-3: A depiction of the hypernym graph that arises from the literal senses of the word "chocolate." Each word sense is annotated with the minimum distance between it and the top-level hypernyms. Note that the two nodes marked "food" above denote different word senses; the "food" at depth 2 indicates a nutrient-bearing substance in general, while the "food" at depth 3 indicates solid food in particular, as in "food and drink".

senses. Noun hypernymy is a WordNet relation that organizes noun senses according to the hierarchical[1] "is-a" relationship; for instance, "brown" is a hypernym of "chocolate" because chocolate "is-a" type of brown.

By repeatedly following hypernym pointers, a series of progressively broader senses can be established, starting from the overspecified literal senses to a set of 9 top-level[2] hypernyms (see Figure 4-3 for an example). We believe that useful

[1] Actually, WordNet hypernymy does not define a true hierarchy, as some senses may have more than one hypernym; for instance, the word sense "beverage" in Figure 4-3 has two hypernyms: "food" and "liquid". For simplicity, however, we will continue to refer to hypernymy as a hierarchical relationship.

[2]Note that these top-level hypernyms are *not* the same as the supersenses described above; the supersenses derive from the 26 noun lexicographer filenames, and are not necessarily related to the

word sense information lies somewhere between the two extremes of granularity; however, locating the exact depth of the useful region is tricky, as the length of a hypernym path varies from sense to sense.

Rather than attempt to explicitly specify the useful depth, then, we simply produce features for all possible hypernyms, and leave it to the reranking model to sort out which hypernyms are useful. Returning to our oft-used example, consider the word "chocolate" and its three senses: "chocolate (beverage)", "chocolate (solid food)", and "chocolate (color)". For the solid food sense, we would produce features for "chocolate (solid food)", "food", "solid", "substance", and "entity"; and likewise for the other two senses. Note that we are not increasing the number of senses, only the number of features; the word "chocolate" would still have only three word senses, and each of these senses would carry features for all of its hypernyms.

The drawback of this hypernym approach is an explosion in the number of features. When we generate pairwise features, we must not only process all pairs of word senses, but for each pair of senses, we must create features for all possible pairs of the *hypernyms* of the two senses involved. More specifically, consider a pair of neighboring nodes u and v, whose possible word senses are given by S_u and S_v. If $H(s_u)$ gives the total number of hypernyms reachable from word sense s_u, then, the total number of features produced is

$$\sum_{s_u \in S_u} \sum_{s_v \in S_v} H(s_u)H(s_v) \;=\; O(S^2H^2)$$

where $S = \max_u |S_u|$ and $H = \max_s H(s)$.

4.3 Results and Discussion

This section describes our experimental results and discusses their implications. We report development test results for the hypernym and supersense feature sets.

hypernymy structure.

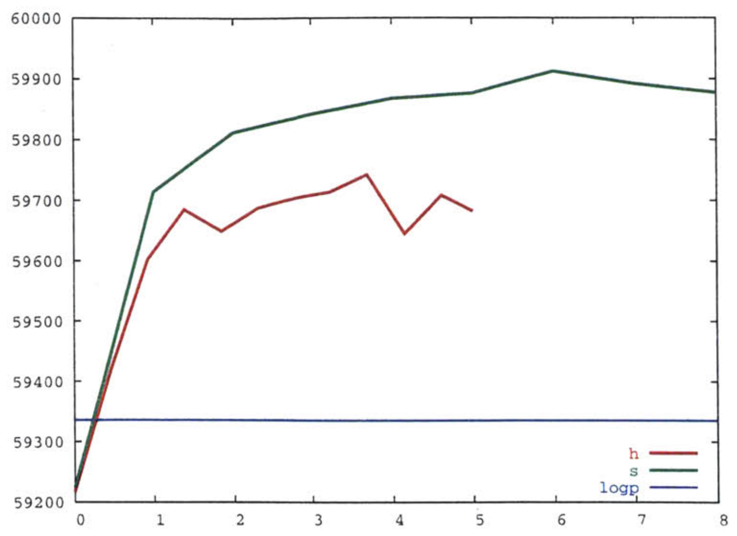


Figure 4-4: The development-set scores of the hypernym (h) and supersense (s) features are graphed versus the number of passes over the training set. The logp baseline score is also graphed.

As the supersense features outperformed the hypernym features in development testing, we chose to evaluate the supersense features in our final tests.

Our experiments fall into two categories: reranking tests, which measure the additive improvement of the new reranker over the base parser, and cascaded reranking tests, which measure the additive improvement of the new reranker over the (Michael Collins and Terry Koo 2004) boosting reranker. The remainder of this section presents our results for both kinds of experiments, and concludes with a discussion of some deficiencies of our reranking model that may have hampered its performance.

4.3.1 Reranking Tests

The point of comparison for the reranking tests is the `logp` baseline, which is simply the score of the base parser by itself. For a fair comparison, we integrated the base parser's ranking with our reranker as described in Section 3.4.1.

In tests on the development set, the hypernym feature set achieved an improvement of $\approx 0.68\%$ over the `logp` baseline. The supersense features, on the other hand, achieved an improvement of $\approx 0.97\%$ past baseline, a significant gain. Figure 4-4 shows the scores achieved by both hypernym and supersense features as

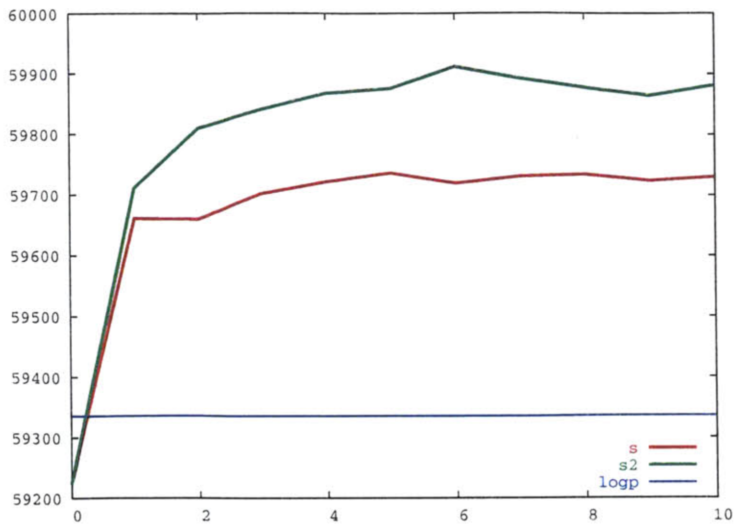


Figure 4-5: The development-set scores of the the backed-off supersense features (s2) and non-backed-off features (s) are graphed for every pass over the training set.

training progressed.

Given the coarseness of the supersense categories, we expected the supersense features to generalize well and were supported by our tests on the development set. The inclusion of backed-off features turned out to be crucial; without back-off, the supersense features achieve an improvement of only $\approx 0.77\%$ over baseline, dropping by a factor of about $1/5$. Figure 4-5 graphs the performance of the backed-off and non-backed-off features.

Nevertheless, the supersense feature set achieved an improvement of only $\approx 0.5\%$ over baseline on the section 23 test set (see Table 4.2). We attribute at least some of this drop to overspecialization of the model toward the development set; many parameters were chosen based on validation with the development set: the learning rate parameters a_0 and c, the number of iterations of training, the parameter θ_{logp} and so forth. Because the development test set was fairly large, we did not believed that we would encounter overspecialization problems. However, in the future, we plan to avoid such problems by using an N-way cross-validation and averaging scheme. We would divide our training data into N chunks, training in turn on $N-1$ chunks and using the held-out chunk as a development set. The N reranking models that are produced would be combined with a weighted

MODEL	$\leq$ 100 Words (2416 sentences)				
	LR	LP	CBs	0 CBs	2 CBs
CO99	88.1%	88.3%	1.06	64.0%	85.1%
CO04	89.6%	89.9%	0.86	68.7%	88.3%
SS	88.4%	88.9%	1.02	65.2%	85.5%

Table 4.2: Results on Section 23 of the Penn Treebank. **LR/LP** = labelled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs, 2 CBs** are the percentage of sentences with 0 or $\leq$ 2 crossing brackets respectively. CO99 is the baseline parser of (Collins 1999). CO04 is the boosting reranker of (Michael Collins and Terry Koo 2004).

average; the relative weightings could be chosen through validation on a separate development set that is withheld from the N-way training.

A somewhat unexpected result was the relative absence of overtraining in the hypernym features. The entire hypernym feature space contains over 30 million features, while the training set spans only about a million trees. Nevertheless, the hypernym features managed to achieve and maintain nontrivial gains. A possible explanation could be that the reranking algorithm has a tendency to assign more weight to more frequent features. Since the higher-level hypernyms appear most frequently, they take control of the hypernym feature set, and the hypernym feature set effectively migrates toward a supersense-like feature set.

4.3.2 Cascaded Reranking Tests

In our cascaded reranking tests, we compare our reranker to the `boost` baseline, which is the score of the base parser augmented by the (Michael Collins and Terry Koo 2004) boosting reranker. In order to measure the true additive improvement, we integrated the rankings of the base parser and boosting reranker into our reranker, as described in Sections 3.4.1 and 3.4.2.

By using a weighted combination of new and old rerankers, we were able to obtain small improvements: for the supersense features, the optimal weighting yielded a development set improvement of $\approx 0.153\%$ past the `boost` baseline. Unfortunately, this improvement did not carry over to the final test results.

4.3.3 Deficiencies of the Reranking Model

One of the biggest deficiencies of our current reranking model is that, as we mentioned in Section 4.2.1, our feature sets currently only assign noun word senses. This restriction was originally imposed in order to keep the model as simple as possible, at least until the various implementation issues could be resolved. Over the course of our experimentation, however, it has become increasingly apparent that word senses for other parts of speech are needed.

First of all, the power of our hidden variable model lies in its ability to model sense-to-sense interactions, yet when the model is restricted to noun senses only, there are few sense-to-sense interactions. Noun-noun dependencies occur in only a handful of situations: restrictive modifications (as in "satellite communications"), appositions, and conjunctions. These noun-noun interactions are typically quite short-range, near the leaves of the parse tree, and therefore they have only a small effect on the correctness of a parse.

If we included verb senses, we would greatly increase the amount of sense-to-sense interaction. Moreover, verb-noun interactions typically span a larger region of the parse tree, being closer to the core structure of a parse. Therefore, we can expect to make stronger gains by learning verb-noun interactions.

If we apply the tree transformations described in Section 5.2.5 on top of the verb senses, we could derive noun-noun and verb-noun dependencies from instances of prepositional phrase modification. These PP-attachment interactions have longer range and are usually quite ambiguous, so we might stand to gain much from using tree transformations.

Besides the restriction to noun senses only, another deficiency of our reranker arises from the predominance of proper nouns in Wall Street Journal text. Naturally, WordNet cannot be expected to provide coverage of these proper nouns. In fact, the use of WordNet on proper nouns can sometimes cause misleading sense interactions. For example, consider the name "Apple Computer, Inc.", in which the fruit sense of "Apple" would be assigned, or the name "John", which WordNet gives the senses of "slang for toilet", "king of England", "apostle", and "part of the bible" (i.e. the gospel according to John).

The example of "John" above also points out a quite different drawback of using WordNet: the frequent occurrence of rare or irrelevant "outlier" senses. For example, one of the senses of "investment" is "a covering of an organ or organism", and the senses of "man" include "Isle of Man" and "board game piece" (e.g. chess man). Although our reranking model can learn to avoid these outlier senses, doing so places an additional burden on the training data. If we can resolve this issue through outside means then the reranker will be better able to address other, more substantive matters.

Chapter 5

Conclusion

We have developed a WordNet-based parse reranking technique that avoids the onus of explicit word sense disambiguation by using a hidden variable model. Although our results were not as strong as we had hoped, there is still a good deal we can add to the reranking model and its feature sets; many possible changes have already been suggested in Section 4.3.3. The remainder of this chapter describes of our ideas for future research.

5.1 Different Gradient-Based Optimization Techniques

First of all, we may wish to use a different gradient-based optimization technique. We have attempted to verify that stochastic gradient descent is viable by using it to optimize the (Michael Collins and Terry Koo 2004) boosting reranker's features. Although stochastic gradient descent succeeded in this task, the boosting features are qualitatively different from our own feature sets in that they are insensitive to word senses. It might be that stochastic gradient descent suffers from some susceptibility to our hidden variable model or feature sets. Therefore, in order to determine whether stochastic gradient descent is at fault or not, we should experiment with other gradient descent methods, such as conjugate gradient descent (M. Johnson and Riezler 1999) or even plain line-search gradient descent.

5.2 New Feature Sets

It is possible that our WordNet-driven hidden variable model contains some innate deficiency that makes it unable to perform well at the parse reranking task. However, it would be rash to arrive at this conclusion without first making a more thorough investigation of alternative feature sets; we have currently explored only two different feature sets in depth. There are a number of alterations we can make to our feature model that offer hope of improved performance, and most of these fixes are quite simple. We describe each alteration in greater detail below.

5.2.1 Hypernyms with Depth Cutoff

A quite simple modification would be to add depth thresholding to the hypernym feature set. Rather than producing features for all hypernyms of a literal sense, we would produce features only for hypernyms that are within a certain distance from the top level. A proper threshold could keep the number of features at a manageable level while still capturing the "active region" of the hypernym structure. Our hope is that the reduction of the feature space would lead to more effective training, which would allow the depth-cutoff features to outperform the standard hypernym features.

5.2.2 Incorporating Verb Senses

As we mentioned in Section 4.3.3, the use of noun senses alone is insufficient. We could easily add verb senses using WordNet; WordNet supplies a verb sense index and a verb hypernymy relationship. In addition, as is the case with noun senses, verb senses can be organized according to their lexical filenames, opening the possibility of verb "supersenses".

Alternatively, we might attempt an unsupervised approach to including verb senses. For each verb, we could propose a small set of, for example, ten unnamed supersenses. Ideally, as training progressed, the unnamed verb supersenses would diverge from each other, each supersense coming to represent a different type of distributional affinity. The verbs in turn would become associated with a mixture of these archetypical supersenses. To encourage heavier learning on the unsupervised verb senses, we might train the reranker without using the rankings of the base parser or boosting reranker.

5.2.3 Overcoming Outlier Senses

In Section 4.3.3, we mentioned the problem of "outlier" senses, which are rare or irrelevant senses that are included in the WordNet sense index. One way to overcome this issue might be to establish a prior probability distribution on the word sense assignments with an independent word sense disambiguation system; these priors would draw the attention of the reranker away from the implausible outliers. Priors could also be inferred from the WordNet sense ordering[1], although this approach could potentially be more noisy.

We could also address the outlier senses using an unsupervised word-clustering method on a large corpus of parsed output from the same domain; words would be clustered based on the distribution of neighboring words in the dependency tree. The clusters formed with this technique would only reflect those word senses which appear in the domain, thereby eliminating the troublesome outlier senses.

5.2.4 Handling Proper Nouns Specially

We mentioned in Section 4.3.3 that proper nouns should not be assigned word senses in the same manner as common nouns. One way to handle proper nouns might be to use a named-entity tagger to fit proper nouns into broad categories. Alternatively, a simpler approach would be to abstain from assigning word senses to proper nouns at all. Finally, a more interesting unsupervised approach would be to give all proper nouns the senses of "person", "location", and "organization",

[1]WordNet orders its word sense index according to how frequently each sense is assigned to each word in various semantically-tagged corpora. However, these counts are not always available, so not all word senses will be ordered.

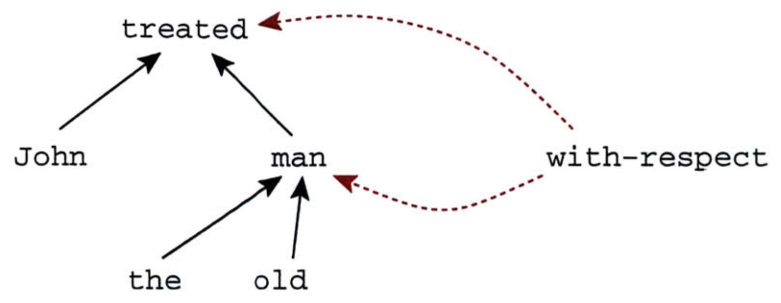


Figure 5-1: A PP-ambiguous dependency in which the preposition has been merged with its noun argument.

letting the reranking model resolve the uncertainty.

5.2.5 Applying Tree Transformations

As we described in Section 4.3.3, a major drawback of our basic model is that function words often interpose at critical junctions of the dependency tree. Therefore, we have begun work on a method for transforming the dependency trees so that function words are merged with their arguments. The senses of the merged node are conjunctions of the function word and the senses of its argument. For example, Figure 5-1 shows how this transformation technique would alter the dependency tree from Figure 4-2. The supersenses of "with-respect" would be given as:

"with-noun.cognition"	"with-noun.state"
"with-noun.act"	"with-noun.feeling"
"with-noun.attribute"	

In the transformed tree, the pairwise features that arise from the dependency between "with-respect" and the attachment points "treated" and "man" would capture interactions between the senses of "respect" and the senses of "treated" or "man", moderated by the preposition "with". Therefore, our reranking model could learn to disambiguate the PP-attachment ambiguity of Figures 4-2 and 5-1.

The general idea behind these tree transformations is simple, but there are some tricky details that need to be addressed. For instance, although the noun and preposition share the same node, we should still document the dependency that exists between the two. Therefore, we define the node feature set of a merged

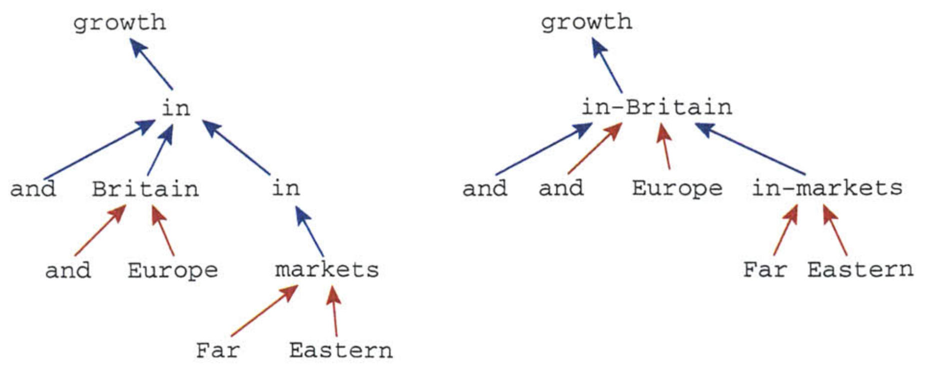


Figure 5-2: Dependency trees for the prepositional phrase "[growth] in Britain and Europe, and in Far Eastern markets" before and after transformations. The dependency arcs are colored according to whether the dependency originally modified a preposition or noun; note that after the transformation, some of the children of "in-Britain" originally modified "Britain" while others originally modified "in".

node to contain all of the features that would normally arise from the two nodes when separate. That is, the node features of "with-respect" would include all the normal node features of "with" and "respect", as well as all of the normal *pairwise* features arising from the dependency between "with" and "respect".

By the same token, we should also preserve the features that would normally arise between the preposition and its other neighbors, as well as the features that would normally arise between the noun argument and its other neighbors. For example, consider the dependency trees shown in Figure 5-2, which depict a complex prepositional phrase before and after transformations are applied. From the dependency between "growth" and "in-Britain", we should produce the features that would normally arise from the dependency between "growth" and "in" as well as the features arising from "growth" and "in-Britain". Similarly, from the dependency between "in-Britain" and "Europe", we should produce the pairwise features that would normally arise from the dependency between "Britain" and "Europe", as well as those arising from "in-Britain" and "Europe". However, note that we should *not* produce features derived from "growth" and "Britain" or "in" and "Europe", as these dependencies did not exist in the untransformed tree. The coloration of the dependency arcs in Figure 5-2 reflects whether the noun or preposition should be used in the production of these preserved features.

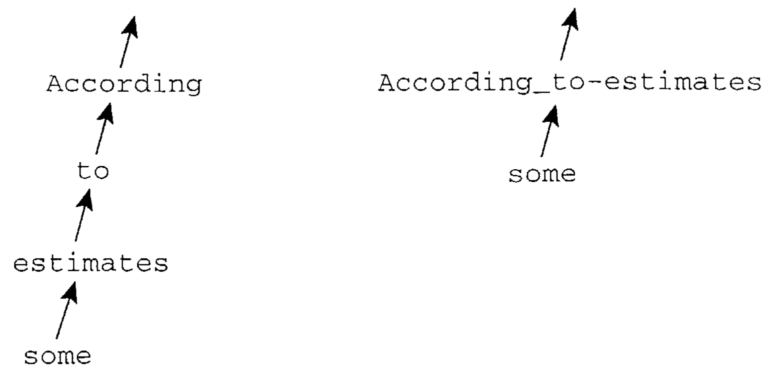


Figure 5-3: Dependency trees for the prepositional phrase "According to some estimates" both before and after transformations.

A last issue is how we deal with cascaded prepositions, such as the example in Figure 5-3. Our current approach is to merge all chains of preposition nodes, concatenating the text of the prepositions and treating them as a single preposition. Note that this approach would discard the preposition-to-preposition dependencies in the original tree and modify the preposition-to-noun features. In the example in Figure 5-3, we would discard the pairwise features arising from the dependency between "According" and "to", and instead of producing pairwise features for "to" and "estimates", we would produce features for "According_to" and "estimates".

Although we have consistently used prepositional phrases as examples, note that these tree transformations can be applied to other interposing function words. Figure 5-4 shows some other transformations that could prove useful.

5.3 Experimenting with a Sense-Tagged Corpus

One possible problem with our reranking approach may be that, despite our intuitions, word sense assignments do not bear strongly on the problem of parse disambiguation. To determine if this is indeed the case, we could experiment with a sense-tagged corpus; our reranker would therefore have access to "perfect" word sense assignments. If the reranking performance remains low, even when using a variety of feature sets, then we must conclude that our initial assumption about

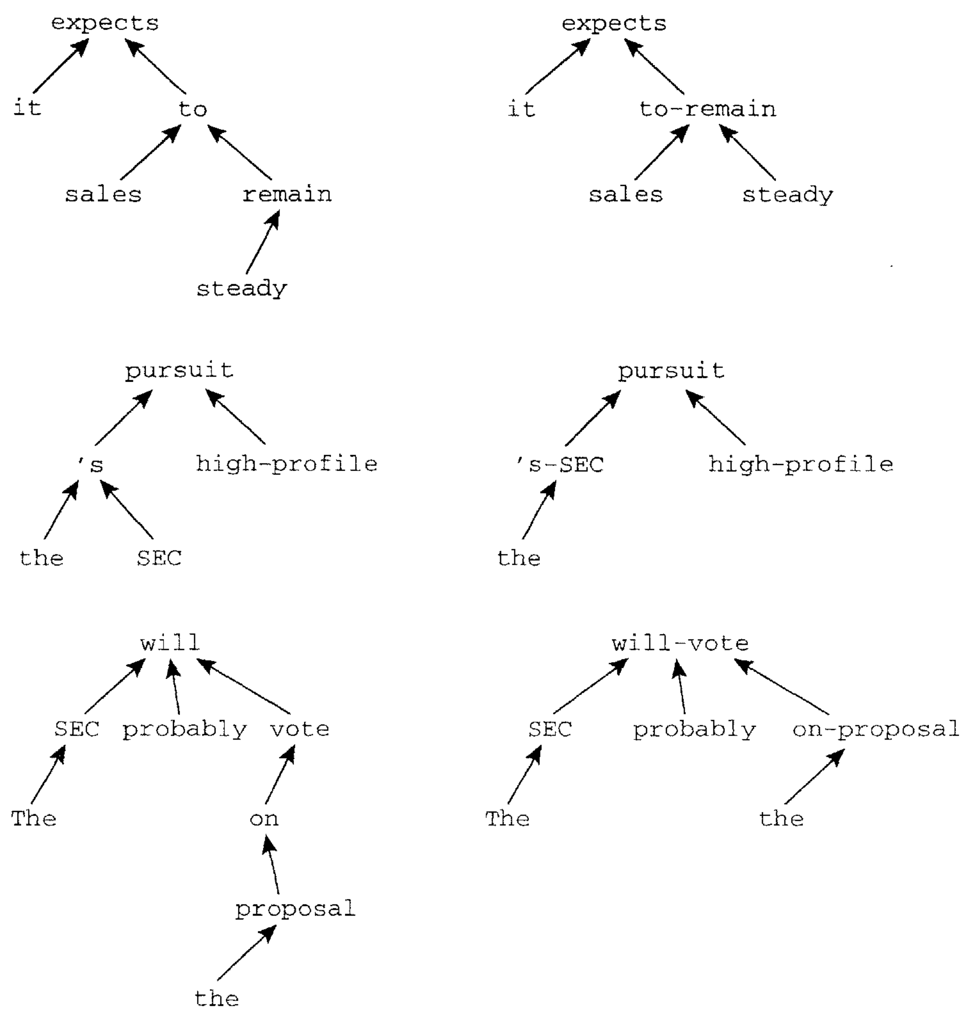


Figure 5-4: Dependency trees depicting transformations that operate on possessive markers and verbal auxiliaries, such as the infinitival "to" and modal verbs. Each transformation removes an intervening functional word, allowing the content words to interact directly. The text of the phrases are "it expects sales to remain steady", "the SEC's high-profile pursuit", and "The SEC will probably vote on the proposal".

the importance of word senses was incorrect. On the other hand, if reranking performance is high, then we can infer that our problems must lie elsewhere.

5.4 Exploring PP-Attachment Ambiguity

Finally, our experiments indicate that a large proportion of reranking decisions hinge on prepositional phrase attachment ambiguities. Therefore, it may be helpful to mount a study that applies our hidden variable model to the PP-attachment ambiguity problem in isolation. Insights we gain from this study would no doubt aid us in the larger reranking task; moreover, the reduced scope of the problem would allow much simpler error analysis and a faster design cycle.

Appendix A

Proofs

This part of the appendix provides proofs that were elided in earlier sections of the document.

A.1 Equivalence of Beliefs and Marginals in Trees

For the purposes of our proof, we define the following notation. Let $\mathcal{T}$ denote the entire tree $t_{i,j}$, and for every edge (u, v), let $\mathcal{T}_{u/v}$ denote the subtree of $\mathcal{T}$ that is rooted at u and that lies "behind" v (i.e. if we were to divide $\mathcal{T}$ into two connected components by cutting edge (u, v), then $\mathcal{T}_{u/v}$ is the component that contains u; see Figure A-1 for some visual examples). Note that the property

$$\forall v \in \mathcal{T}\ \forall s, t \in N(v) \qquad \mathcal{T}_{s/v} \cap \mathcal{T}_{t/v} = \{\}$$

holds, or else we could show a cycle in $\mathcal{T}$. We define the function $\mathbf{a}$ such that for any set of nodes S, $\mathbf{a}(S)$ enumerates through all word sense assignments to the nodes in S. We define $\mathbf{a}$ over trees as well; $\mathbf{a}(\mathcal{T})$ and $\mathbf{a}(\mathcal{T}_{u/v})$ enumerate through sense assignments to the nodes of these subtrees. Under this new notation, the marginal probabilities for each node and pair of nodes can now be given by:

$$\begin{aligned} p_u(x_u) &= \frac{1}{Z_{i,j}} \sum_{\mathbf{a}(\mathcal{T}) \,|\, s_u = x_u} \prod_{w \in \mathcal{T}} \alpha_w(s_w) \prod_{(u',v') \in \mathcal{T}} \beta_{u',v'}(s_{u'}, s_{v'}) \\ p_{u,v}(x_u, x_v) &= \frac{1}{Z_{i,j}} \sum_{\mathbf{a}(\mathcal{T}) \,|\, s_u = x_u, s_v = x_v} \prod_{w \in \mathcal{T}} \alpha_w(s_w) \prod_{(u',v') \in \mathcal{T}} \beta_{u',v'}(s_{u'}, s_{v'}) \end{aligned}$$

where the notation $\mathbf{a}(\mathcal{T}) \mid s_u = x_u$ indicates enumeration over all sense assignments

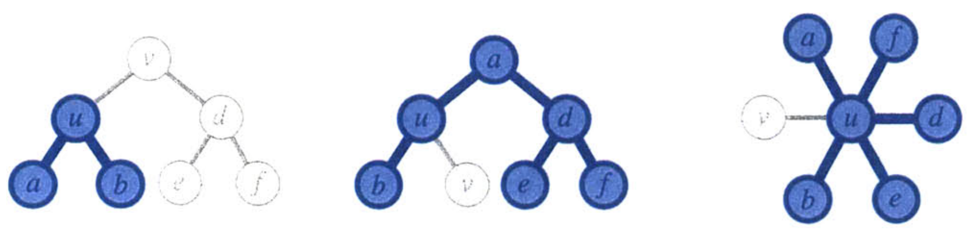


Figure A-1: Depictions of the shape of $\mathcal{T}_{u/v}$ in some sample graphs.

of $\mathcal{T}$ where s_u is fixed to the value x_u. We define partial normalization constants:

$$\begin{aligned}
\mathbf{Z}(\mathcal{T}_{u/v}) &= \sum_{\mathbf{a}(\mathcal{T}_{u/v})} \prod_{w \in \mathcal{T}_{u/v}} \alpha_w(s_w) \prod_{(u',v') \in \mathcal{T}_{u/v}} \beta_{u',v'}(s_{u'}, s_{v'}) \\
\mathbf{Z}(\mathcal{T}_{u/v} \,|\, s_u = x_u) &= \sum_{\mathbf{a}(\mathcal{T}_{u/v}) \,|\, s_u = x_u} \prod_{w \in \mathcal{T}_{u/v}} \alpha_w(s_w) \prod_{(u',v') \in \mathcal{T}_{u/v}} \beta_{u',v'}(s_{u'}, s_{v'})
\end{aligned}$$

where $\mathbf{Z}(\mathcal{T}_{u/v})$ is the normalization constant for subtree $\mathcal{T}_{u/v}$, and $\mathbf{Z}(\mathcal{T}_{u/v} \,|\, s_u = x_u)$ gives the normalization constant for subtree $\mathcal{T}_{u/v}$ when sense s_u is fixed to the particular value x_u.

It is worthwhile to explore some of the properties of these partial normalization constants before we continue. First, note that the normalization constant $\mathbf{Z}(\mathcal{T}_{u/v})$ can be built up out of partially-fixed normalization constants $\mathbf{Z}(\mathcal{T}_{u/v} \,|\, s_u = x_u)$ as follows:

$$\mathbf{Z}(\mathcal{T}_{u/v}) = \sum_{x_u \in S_u} \mathbf{Z}(\mathcal{T}_{u/v} \,|\, s_u = x_u)$$

This property also holds when more than one word sense is being fixed

$$\mathbf{Z}(\mathcal{T}_{u/v}) = \sum_{x_u \in S_u,\, x_w \in S_w} \mathbf{Z}(\mathcal{T}_{u/v} \,|\, s_u = x_u, s_w = x_w)$$

and in general, when a subset S of the nodes has their word senses fixed, then

$$\mathbf{Z}(\mathcal{T}_{u/v}) = \sum_{\mathbf{a}(S)} \mathbf{Z}(\mathcal{T}_{u/v} \,|\, \mathbf{a}(S))$$

In addition, the product between the normalization constants for any two disjoint subtrees $\mathcal{T}_{u/v}$ and $\mathcal{T}_{s/t}$ yields a normalization constant for the union of the two subtrees:

$$\begin{aligned}
\mathbf{Z}(\mathcal{T}_{u/v})\mathbf{Z}(\mathcal{T}_{s/t}) &= \left(\sum_{\mathbf{a}(\mathcal{T}_{u/v})} \prod_{w\in\mathcal{T}_{u/v}} \alpha_w(s_w) \prod_{(u',v')\in\mathcal{T}_{u/v}} \beta_{u',v'}(s_{u'}, s_{v'})\right) \\
&\quad \left(\sum_{\mathbf{a}(\mathcal{T}_{s/t})} \prod_{r\in\mathcal{T}_{s/t}} \alpha_r(s_r) \prod_{(s',t')\in\mathcal{T}_{s/t}} \beta_{s',t'}(s_{s'}, s_{t'})\right) \\
&= \sum_{\mathbf{a}(\mathcal{T}_{u/v})} \sum_{\mathbf{a}(\mathcal{T}_{s/t})} \left(\begin{array}{c} \prod_{w\in\mathcal{T}_{u/v}} \alpha_w(s_w) \prod_{(u',v')\in\mathcal{T}_{u/v}} \beta_{u',v'}(s_{u'}, s_{v'}) \\ \prod_{r\in\mathcal{T}_{s/t}} \alpha_r(s_r) \prod_{(s',t')\in\mathcal{T}_{s/t}} \beta_{s',t'}(s_{s'}, s_{t'}) \end{array} \right) \\
&= \sum_{\mathbf{a}(\mathcal{T}_{u/v}\cup\mathcal{T}_{s/t})} \prod_{w\in(\mathcal{T}_{u/v}\cup\mathcal{T}_{s/t})} \alpha_w(s_w) \prod_{(u',v')\in(\mathcal{T}_{u/v}\cup\mathcal{T}_{s/t})} \beta_{u',v'}(s_{u'}, s_{v'}) \\
&= \mathbf{Z}\left(\mathcal{T}_{u/v}\cup\mathcal{T}_{s/t}\right)
\end{aligned}$$

We can draw an interesting conclusion from the above. Recall that for any node v, $\forall s,t \in N(v)$, $\mathcal{T}_{s/v} \cap \mathcal{T}_{t/v} = \{\}$; that is, all neighboring subtrees are disjoint. Therefore, we can construct the tree normalization constant $\mathbf{Z}(\mathcal{T}_{u/v})$ by piecing together smaller normalization constants with node and edge weights.

$$\mathbf{Z}(\mathcal{T}_{u/v}) = \sum_{x_u\in S_u} \alpha_u(x_u) \prod_{w\in N(u)\setminus v} \sum_{x_w\in S_w} \beta_{u,w}(x_u, x_w)\mathbf{Z}(\mathcal{T}_{w/u} \mid s_w = x_w)$$

and a natural extension is that for any node u, we can compute the normalization constant for the entire tree by combining the tree normalization constants for all of the subtrees neighboring u:

$$\forall u, \qquad \mathbf{Z}(\mathcal{T}) = \sum_{x_u\in S_u} \alpha_u(x_u) \prod_{w\in N(u)} \sum_{x_w\in S_w} \beta_{u,w}(x_u, x_w)\mathbf{Z}(\mathcal{T}_{w/u} \mid s_w = x_w)$$

We now continue with our proof that tree beliefs equal marginal probabilities. For convenience, we reproduce the recursive formula defining the messages:

$$m_{u\to v}(s_v) = \sum_{s_u\in S_u} \alpha_u(s_u)\beta_{u,v}(s_u, s_v) \prod_{w\in N(u)\setminus v} m_{w\to u}(s_u)$$

The key intuition in our proof is to think of the messages $m_{u\to v}$ as dynamic-programming subproblems, where each message $m_{u\to v}$ is related to the normalization factor $\mathbf{Z}(\mathcal{T}_{u/v})$. The exact relation is given by predicate P below:

$$P(m_{u\to v}) \equiv \left\{ m_{u\to v}(s_v) = \sum_{x_u \in S_u} \beta_{u,v}(x_u, s_v)\mathbf{Z}(\mathcal{T}_{u/v} \mid s_u = x_u) \right\}$$

and the following proof shows that P holds for messages produced by the message-passing algorithm as described in Section 3.2.2.

Proof by Induction

Base Case The messages emanating from the leaves of $\mathcal{T}$ are the base case of the induction. For a leaf ℓ and its parent p, $P(m_{\ell\to p})$ holds trivially. Since the only neighbor of a leaf node is its parent, $N(\ell) \setminus p = \{\}$ and $\mathbf{Z}(\mathcal{T}_{\ell/p} \mid s_\ell = x_\ell) = \alpha_\ell(x_\ell)$. Therefore,

$$\begin{aligned} m_{\ell\to p}(s_p) &= \sum_{s_\ell \in S_\ell} \alpha_\ell(s_\ell)\beta_{\ell,p}(s_\ell, s_p) \prod_{v\in N(\ell)\setminus p} m_{v\to\ell}(s_\ell) \\ &= \sum_{s_\ell \in S_\ell} \alpha_\ell(s_\ell)\beta_{\ell,p}(s_\ell, s_p) \\ &= \sum_{x_\ell \in S_\ell} \beta_{\ell,p}(x_\ell, s_p)\mathbf{Z}(\mathcal{T}_{\ell/p} \mid s_\ell = x_\ell) \end{aligned}$$

so that $P(m_{\ell\to p})$ holds.

Inductive Case We prove the property $P(m_{u\to v})$, and our inductive assumption is that $\forall w \neq v$, $P(m_{w\to u})$ holds. We argue that this is a fair inductive assumption, since the messages $m_{w\to u}$, $w \neq v$ are exactly those messages which would be required to compute $m_{u\to v}$ in the message passing algorithm (as laid out in Section 3.2.2). We begin by writing out the formula for $m_{u\to v}(s_v)$ and substituting in our inductive assumptions:

$$\begin{aligned} m_{u\to v}(s_v) &= \sum_{x_u \in S_u} \alpha_u(x_u)\beta_{u,v}(x_u, s_v) \\ &\qquad \prod_{w\in N(u)\setminus v} \sum_{x_w \in S_w} \beta_{w,u}(x_w, x_u)\mathbf{Z}(\mathcal{T}_{w/u} \mid s_w = x_w) \end{aligned}$$

We define $F(w, x_w) = \beta_{w,u}(x_w, x_u)\mathbf{Z}(\mathcal{T}_{w/u} \mid s_w = x_w)$ and rewrite the above as follows:

$$m_{u\to v}(s_v) \quad = \quad \sum_{x_u \in S_u} \alpha_u(x_u)\beta_{u,v}(x_u, s_v) \prod_{w\in N(u)\backslash v} \sum_{x_w \in S_w} F(w, x_w)$$

Next, we rearrange the product over sums $\prod_{w\in N(u)\backslash v} \sum_{x_w\in S_w} F(w, x_w)$ into a sum over products:

$$\begin{aligned}
&\prod_{w\in N(u)\backslash v} \sum_{x_w\in S_w} F(w, x_w) \\
&= \begin{pmatrix}
\left(F(w_1, x^1_{w_1}) + F(w_1, x^2_{w_1}) + \ldots + F(w_1, x^{|S_{w_1}|}_{w_1})\right) \\
\left(F(w_2, x^1_{w_2}) + F(w_2, x^2_{w_2}) + \ldots + F(w_2, x^{|S_{w_2}|}_{w_2})\right) \\
\vdots \\
\left(F(w_M, x^1_{w_1}) + F(w_M, x^2_{w_M}) + \ldots + F(w_M, x^{|S_{w_M}|}_{w_M})\right)
\end{pmatrix} \\
&= \begin{pmatrix}
\left(F(w_1, x^1_{w_1}) \quad F(w_2, x^1_{w_2}) \quad \ldots \quad F(w_M, x^1_{w_M})\right) + \\
\left(F(w_1, x^1_{w_1}) \quad F(w_2, x^1_{w_2}) \quad \ldots \quad F(w_M, x^2_{w_M})\right) + \\
\vdots \\
\left(F(w_1, x^2_{w_1}) \quad F(w_2, x^1_{w_2}) \quad \ldots \quad F(w_M, x^1_{w_M})\right) + \\
\vdots \\
\left(F(w_1, x^{|S_{w_1}|}_{w_1}) \quad F(w_2, x^{|S_{w_2}|}_{w_2}) \quad \ldots \quad F(w_M, x^{|S_{w_M}|}_{w_M})\right) +
\end{pmatrix} \\
&= \sum_{\mathbf{a}(N(u)\backslash v)} \prod_{w\in N(u)\backslash v} F(w, s_w)
\end{aligned}$$

Applying this rearrangement to our original expression allows the following simplifications:

$$\begin{aligned}
m_{u\to v}(s_v) &= \sum_{x_u\in S_u} \alpha_u(x_u)\beta_{u,v}(x_u, s_v) \sum_{\mathbf{a}(N(u)\backslash v)} \prod_{w\in N(u)\backslash v} \beta_{w,u}(s_w, x_u)\mathbf{Z}(\mathcal{T}_{w/u} \mid s_w) \\
&= \sum_{x_u\in S_u} \beta_{u,v}(x_u, s_v) \sum_{\mathbf{a}(N(u)\backslash v)} \mathbf{Z}\left(\bigcup_{w'\in N(u)\backslash v} \mathcal{T}_{w'/u} \,\middle|\, \mathbf{a}(N(u) \setminus v)\right) \\
&\qquad\qquad \alpha_u(x_u) \prod_{w\in N(u)\backslash v} \beta_{w,u}(s_w, x_u)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{x_u \in S_u} \beta_{u,v}(x_u, s_v) \sum_{\mathbf{a}(N(u)\setminus v)} \mathbf{Z}(\mathcal{T}_{u/v} \mid \mathbf{a}(N(u) \setminus v), s_u = x_u) \\
&= \sum_{x_u \in S_u} \beta_{u,v}(x_u, s_v) \mathbf{Z}(\mathcal{T}_{u/v} \mid s_u = x_u)
\end{aligned}
$$

Therefore, $P(m_{u \to v})$ holds. ■

Now, using the definition in proposition P, we can prove that the node beliefs are equal to the marginal probabilities:

$$
\begin{aligned}
b_u(x_u) &= \frac{1}{z_u} \alpha_u(x_u) \prod_{v \in N(u)} m_{v \to u}(s_u) \\
&= \frac{1}{z_u} \alpha_u(x_u) \prod_{v \in N(u)} \sum_{x_v \in S_v} \beta_{v,u}(x_v, x_u) \mathbf{Z}(\mathcal{T}_{v/u} \mid s_v = x_v) \\
&= \frac{1}{z_u} \alpha_u(x_u) \sum_{\mathbf{a}(N(u))} \prod_{v \in N(u)} \beta_{v,u}(x_v, x_u) \mathbf{Z}(\mathcal{T}_{v/u} \mid s_v = x_v) \\
&= \frac{1}{z_u} \sum_{\mathbf{a}(N(u))} \mathbf{Z}\left(\bigcup_{w \in N(u)} \mathcal{T}_{w/u} \,\middle|\, \mathbf{a}(N(u)), s_u = x_u \right) \alpha_u(x_u) \prod_{v \in N(u)} \beta_{v,u}(x_v, x_u) \\
&= \frac{1}{z_u} \sum_{\mathbf{a}(N(u))} \mathbf{Z}\left(\mathcal{T} \mid \mathbf{a}(N(u)), s_u = x_u\right) \\
&= \frac{1}{z_u} \mathbf{Z}\left(\mathcal{T} \mid s_u = x_u\right) \qquad\qquad (*) \\
&= \frac{1}{z_u} \sum_{\mathbf{a}(\mathcal{T}) \mid s_u = x_u} \prod_{w \in \mathcal{T}} \alpha_w(s_w) \prod_{(u',v') \in \mathcal{T}} \beta_{u',v'}(s_{u'}, s_{v'}) = p_u(x_u)
\end{aligned}
$$

and similar reasoning can be applied to the pairwise beliefs:

$$
\begin{aligned}
b_{u,v}(x_u, x_v) &= \frac{1}{z_{u,v}} \alpha_u(x_u) \alpha_v(x_v) \beta_{u,v}(x_u, x_v) \prod_{s \in N(u) \setminus v} m_{s \to u}(s_u) \prod_{t \in N(v) \setminus u} m_{t \to v}(s_v) \\
&= \frac{1}{z_{u,v}} \beta_{u,v}(x_u, x_v) \left(\alpha_u(x_u) \prod_{s \in N(u)\setminus v} \sum_{x_s \in S_s} \beta_{s,u}(x_s, x_u) \mathbf{Z}(\mathcal{T}_{s/u} \mid s_s = x_s) \right) \\
&\quad \left(\alpha_v(x_v) \prod_{t \in N(v)\setminus u} \sum_{x_t \in S_t} \beta_{t,v}(x_t, x_v) \mathbf{Z}(\mathcal{T}_{t/v} \mid s_t = x_t) \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{z_{u,v}}\beta_{u,v}(x_u,x_v)\left(\alpha_u(x_u)\sum_{\mathbf{a}(N(u)\backslash v)}\prod_{s\in N(u)\backslash v}\beta_{s,u}(x_s,x_u)\mathbf{Z}(\mathcal{T}_{s/u}\mid s_s=x_s)\right)\\
&\qquad\left(\alpha_v(x_v)\sum_{\mathbf{a}(N(v)\backslash u)}\prod_{t\in N(v)\backslash u}\beta_{t,v}(x_t,x_v)\mathbf{Z}(\mathcal{T}_{t/v}\mid s_t=x_t)\right)\\
&= \frac{1}{z_{u,v}}\beta_{u,v}(x_u,x_v)\mathbf{Z}(\mathcal{T}_{u/v}\mid s_u=x_u)\mathbf{Z}(\mathcal{T}_{v/u}\mid s_v=x_v)\\
&= \frac{1}{z_{u,v}}\mathbf{Z}(\mathcal{T}\mid s_u=x_u,s_v=x_v) \qquad\qquad (*)\\
&= \frac{1}{z_{u,v}}\sum_{\mathbf{a}(\mathcal{T})\mid s_u=x_u,s_v=x_v}\prod_{w\in\mathcal{T}}\alpha_w(s_w)\prod_{(u',v')\in\mathcal{T}}\beta_{u',v'}(s_{u'},s_{v'})=p_{u,v}(x_u,x_v)
\end{aligned}
$$

Incidentally, the above also proves that the node and pairwise normalization constants are equal to the tree-wide normalization constant $Z_{i,j}$. Consider the lines marked $(*)$ above, and note that

$$
\begin{aligned}
&\sum_{x_u\in S_u} b_u(x_u)=1 \quad\Rightarrow\\
&\qquad z_u=\sum_{x_u\in S_u}\mathbf{Z}(\mathcal{T}\mid s_u=x_u)=\mathbf{Z}(\mathcal{T})=Z_{i,j}\\
&\sum_{x_u\in S_u,\,x_v\in S_v} b_{u,v}(x_u,x_v)=1 \quad\Rightarrow\\
&\qquad z_{u,v}=\sum_{x_u\in S_u,\,x_v\in S_v}\mathbf{Z}(\mathcal{T}\mid s_u=x_u,s_v=x_v)=\mathbf{Z}(\mathcal{T})=Z_{i,j}
\end{aligned}
$$

A.2 Computing Messages and Beliefs in Linear Time

We show that given node and pairwise potentials α_u and $\beta_{u,v}$, belief propagation can create messages and beliefs in time linear in the size of the tree. Recall that n is the number of nodes in the tree, $E_{i,j}$ gives the set of edges in the tree, $N(u)$ gives the set of neighbors of u, and the constant $S=\max|S_u|$ bounds the size of the word sense domains. We reproduce below the recursive formula defining the messages:

$$
m_{u\to v}(s_v)=\sum_{s_u\in S_u}\alpha_u(s_u)\beta_{u,v}(s_u,s_v)\prod_{w\in N(u)\backslash v}m_{w\to u}(s_u)
$$

We create the messages in each tree by first computing messages from the leaves inward to the root, and then from the root outward to the leaves (see Figure

3-5 on page 34). We call the first set of messages (leaves to root) *upstream* messages and the second set (root to leaves) *downstream* messages.

Now, suppose we wish to compute the upstream message from node u to its parent p. This message $m_{u \to p}(s_p)$ will have $|S_p| \leq S$ dimensions, and for each dimension we must take a summation over $|S_u| \leq S$ products of the $|N(u)| - 1$ messages that arrive from u's children. After multiplying in the weights α_u and $\beta_{u,p}$ and completing the summations, the entire message will require $O(|N(u)|S^2)$ time. As we compute this upstream message, however, we save each product of child messages in a "message product" array $mp_u(s_u)$ that is attached to the node. That is, we set

$$mp_u(s_u) = \prod_{c \in N(u) \setminus p} m_{c \to u}(s_u)$$

We will save a lot of time by reusing these computations later on. Of course, these message product arrays use up an additional $O(nS)$ space. However, note that the messages, node beliefs, and pairwise beliefs already require $O(nS)$, $O(nS)$, and $O(nS^2)$ space respectively, so incurring an additional $O(nS)$ space cost is acceptable.

Now, we compute the downstream messages. We begin at the root node r and compute the downstream message to a child node u. Again, there are $|S_u| \leq S$ dimensions and a summation over $|S_r| \leq S$ terms. However, instead of recomputing the product of messages, we reuse the stored values in the message product array, dividing them by the single held-out message:

$$\prod_{c \in N(r) \setminus u} m_{c \to r}(s_r) = \frac{mp_r(s_r)}{m_{u \to r}(s_r)}$$

When we compute the downstream messages from a non-root node u to its child c, we augment u's stored message product with the downstream message received from u's parent p, and then we reuse the message products as before; that is:

$$\begin{aligned} mp_u(s_u) &\leftarrow mp_u(s_u) m_{p \to u}(s_u) \\ \prod_{v \in N(u) \setminus c} m_{v \to u}(s_u) &= \frac{mp_u(s_u)}{m_{c \to u}(s_u)} \end{aligned}$$

Therefore, the message product required by the recursive message definition can be computed in $O(1)$ time for downstream messages; an entire downstream message can be computed in $O(S^2)$ time. However, each node u has $O(|N(u)|)$ downstream messages to create, and thus taking care of node u's entire quota of downstream messages requires $O(|N(u)|S^2)$ time.

Thus, every node u (except the root) must compute a single upstream message at a cost of $O(|N(u)|S^2)$, and every node u (except the leaves) must compute $O(|N(u)|)$ downstream messages at a cost of $O(S^2)$ each. The total cost of computing all messages, therefore, is bounded by

$$2\sum_u O(|N(u)|S^2) = O\left(S^2 \sum_u |N(u)|\right)$$

The summation $\sum_u |N(u)|$ counts each edge twice, once for each endpoint; therefore, $\sum_u |N(u)| = O(n)$, and the total time required to compute all messages is $O(nS^2)$.

Note that at the finish of the message passing algorithm, for every node u, the message product array mp_u contains the product of all incoming messages: when we computed the upstream messages, we initialized mp_u as the product of all child messages, and when we computed the downstream messages, we updated mp_u to include the parent message. We will exploit these saved products again below.

We compute the beliefs b_u and $b_{u,v}$ and their normalizing constants z_u and $z_{u,v}$ by defining intermediate terms

$$\begin{aligned} B_u(s_u) &= \alpha_u(s_u)mp_u(s_u) \\ B_{u,v}(s_u, s_v) &= \alpha_u(s_u)\alpha_v(s_v)\beta_{u,v}(s_u, s_v)\left(\frac{mp_u(s_u)}{m_{v\to u}(s_u)}\right)\left(\frac{mp_v(s_v)}{m_{u\to v}(s_v)}\right) \end{aligned}$$

and using them to create the normalization constants and beliefs:

$$\begin{aligned} z_u &= \sum_{s_u \in S_u} B_u(s_u) \\ z_{u,v} &= \sum_{s_u \in S_u,\, s_v \in S_v} B_{u,v}(s_u, s_v) \\ b_u(s_u) &= \frac{1}{z_u} B_u(s_u) \\ b_{u,v}(s_u, s_v) &= \frac{1}{z_{u,v}} B_{u,v}(s_u, s_v) \end{aligned}$$

Computing each B_u requires $O(S)$ time and computing each $B_{u,v}$ requires $O(S^2)$ time; preparing all of the intermediate terms requires $O(nS + nS^2) = O(nS^2)$ time. Next, each z_u requires $O(S)$ time and each $z_{u,v}$ requires $O(S^2)$ time; all of them together require another $O(nS^2)$ time. Finally, the node and pairwise beliefs require only $O(1)$ additional computation per belief, which incurs a third $O(nS^2)$ cost.

In conclusion, the messages require $O(nS^2)$ time to create, and when the messages are complete, the normalization constants and beliefs can be computed with an additional $O(nS^2)$ time. Therefore, given node and pairwise weights α_u and $\beta_{u,v}$, the belief propagation algorithm can create node and pairwise beliefs b_u and $b_{u,v}$, together with the associated normalization constants z_u and $z_{u,v}$, in $O(nS^2)$ time per tree.

Bibliography

Daniel Bikel. A Statistical Model for Parsing and Word-Sense Disambiguation. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 2000.

Brian Roark, Murat Saraclar, and Michael Collins. Corrective Language Modeling for Large Vocabulary ASR with the Perceptron Algorithm. To appear in *Proceedings of the ICASSP*, 2004a.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative Language Modeling with Conditional Random Fields and the Perceptron Algorithm. To appear in *Proceedings of the ACL*, 2004b.

Eugene Charniak. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

Eugene Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of the NAACL*, 2000.

Michael Collins. Head-Driven Statistical Models for Natural Language Parsing. Doctoral Dissertation, Dept. of Computer Science, Brown University, Providence, RI, 1999.

Michael Collins and James Brooks. Prepositional Phrase Attachment through a Backed-off Model. In *Proceedings of the Third Workshop on Very Large Corpora*, 1995.

Mark Johnson and Massimiliano Ciaramita. Supersense Tagging of Unknown Nouns in WordNet. In *EMNLP 2003*, 2003.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

S. Canon Z. Chi M. Johnson, S. Geman and S. Riezler. Estimators for Stochastic "Unification-Based" Grammars. In *Proceedings of the ACL*, 1999.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 1994.

Michael Collins and Terry Koo. Discriminative Reranking for Natural Language Parsing. To appear in *Computational Linguistics*, 2004.

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Five Papers on WordNet. Technical report, Stanford University, 1993.

A. Ratnaparkhi and S. Roukos. A Maximum Entropy Model for Prepositional Phrase Attachment. In *Proceedings of the ARPA Workshop on Human Language Technology*, 1994.

Adwait Ratnaparkhi. Learning to Parse Natural Language with Maximum Entropy Models. *Machine Learning*, 34:151–178, 1999.

Jira Stetina and Makato Nagao. Corpus Based PP Attachment Ambiguity Resolution with a Semantic Dictionary. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding Belief Propagation and its Generalizations. Technical Report TR2001-22, Mitsubishi Electric Research Labs, 2002.