University of Alberta

LEARNING STRUCTURED CLASSIFIERS FOR STATISTICAL DEPENDENCY PARSING

by

Qin Iris Wang ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Fall 2008

Your file Votre référence
ISBN: 978-0-494-46446-5
Our file Notre référence
ISBN: 978-0-494-46446-5

To my parents, who made me who I am today.

Abstract

In this thesis, I present three supervised and one semi-supervised machine learning approach for improving statistical natural language dependency parsing. I first introduce a generative approach that uses a strictly lexicalised parsing model where all the parameters are based on words, without using any part-of-speech (POS) tags or grammatical categories. Then I present an improved large margin approach for learning dependency parsers from treebank data that allows a more general set of linguistic features to be used. Specifically, I incorporate local constraints that enforce the correctness of each individual link, rather than just scoring the whole parse tree. For dealing with sparse data, I smooth the lexical parameters according to their underlying word similarities using Laplacian regularization. Third, I present a simpler and more efficient approach to training dependency parsers by applying a boosting-like procedure to standard supervised training methods. By using logistic regression as an efficient base classifier (for predicting dependency links between word pairs), I am able to efficiently train a dependency parsing model, via structured boosting, that achieves state-of-the-art results in English, and surpasses state-of-the-art in Chinese. Finally, I propose a novel semi-supervised training algorithm for learning dependency parsers. By combining a supervised large margin loss with an unsupervised least squares loss, I obtain a discriminative, convex, semi-supervised training algorithm for dependency parsing.

Acknowledgements

I was very fortunate to have two fantastic supervisors, Dr. Dekang Lin and Dr. Dale Schuurmans. I am profoundly grateful to Dekang for his continuous help and encouragement throughout my graduate career. (I just wish he could've stayed longer at the University of Alberta!) I am also eternally grateful to Dale, who is always insightful, energetic, encouraging and helpful. His vigorous pursuit of excellence in research, teaching, advising and every other aspect of his academic work is truly inspiring.

I would like to thank my thesis committee, Dr. Greg Kondrak, Dr. Randy Goebel, Dr. Chris Westbury and my external examiner Dr. Martha Palmer, for their excellent and helpful suggestions and thought-provoking questions. In particular, many thanks to Martha and Chris for their valuable comments on my thesis.

I am grateful to all the staff at the Alberta Ingenuity Center for Machine Learning lab, especially Leslie Acker, who is a wonderful lady and cheered me up every day.

I would also like to thank my brilliant colleagues, Colin Cherry, Chris Pinchak, Shane Bergsma, and my fabulous friends I met at conferences, especially Liang Huang from the University of Pennsylvania. Their persistence, diligence and perseverance have always inspired me to better myself.

Special thanks to my great friends, Nathan, Yuntong, Yang and Fang, who are always there whenever I need a hand. I am also grateful to Baochun, Yuxi, Guohua, Linli, Feng, John, Dan, Jason, Shanny, Oliver, Hongjing, Sufeng, Oneway, Xiaorui, Joyce, and many others, for making Edmonton an enjoyable and memorable place to spend my time doing graduate studies.

Finally, I would like to thank my parents. They provided an enormous amount of support and made me who I am today. I dedicate this thesis to them.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Main Contributions . . . 3
- 1.3 Thesis Outline . . . 5

2 Background on Parsing **7**
- 2.1 What Is a Parser? . . . 7
- 2.2 Syntactic Representations of a Parse Tree . . . 7
  - 2.2.1 Dependency Structures vs. Constituency Structures . . . 8
  - 2.2.2 Other Syntactic Representations . . . 10
- 2.3 Parsing Algorithms . . . 11
  - 2.3.1 Dynamic Programming . . . 11
  - 2.3.2 Dependency Parsing Algorithms . . . 12
- 2.4 Scoring Parses . . . 13
  - 2.4.1 Local Scoring Functions . . . 13
  - 2.4.2 Static Features . . . 14
  - 2.4.3 Dynamic Features . . . 16
- 2.5 Treebanks . . . 17

3 Background on Learning to Parse **18**
- 3.1 Introduction . . . 19
- 3.2 Early Statistical Approaches . . . 20
- 3.3 Reranking . . . 23
- 3.4 Local Training Methods . . . 24
  - 3.4.1 Logistic Regression Models . . . 24
  - 3.4.2 Support Vector Machines . . . 26
- 3.5 Global Training Methods . . . 27
  - 3.5.1 Conditional Random Fields . . . 28
  - 3.5.2 Structured Large Margin Training . . . 29
- 3.6 Other Relevant Machine Learning Techniques . . . 30
  - 3.6.1 Boosting . . . 30
  - 3.6.2 Similarity Smoothing . . . 32
  - 3.6.3 Unsupervised/Semi-supervised Learning . . . 34

4 Experimental Setup **37**
- 4.1 Data Sets . . . 37
  - 4.1.1 English . . . 37
  - 4.1.2 Chinese . . . 38
- 4.2 Evaluation Measures . . . 38
- 4.3 Complexity Issues . . . 40

5 Strictly Lexicalised Dependency Parsing **41**
- 5.1 Lexicalised Parsing . . . 41
- 5.2 A Probabilistic Dependency Parsing Model . . . 42
- 5.3 Similarity-based Smoothing . . . 46
- 5.4 Dependency Parsing Algorithms . . . 47

5.5 Experimental Results 49
5.6 Related Work 50
5.7 Contributions 52
5.8 Conclusion 52

6 Extensions to Large Margin Dependency Parsing **53**
6.1 Large Margin Training with Local Constraints 54
6.1.1 Large Margin Training 54
6.1.2 Training with Local Constraints 55
6.2 Laplacian Regularization 57
6.3 Experimental Results 58
6.4 Related Work 61
6.5 Contributions 62
6.6 Conclusion 62

7 Training Dependency Parsers via Structured Boosting **63**
7.1 Introduction 63
7.2 Structured Boosting 64
7.3 Implementation Details 65
7.4 Experimental Results 67
7.5 Contributions 70
7.6 Conclusion 71

8 Semi-supervised Convex Training for Dependency Parsing **72**
8.1 Introduction 73
8.2 Supervised Structured Large Margin Training 75
8.3 Semi-supervised Structured Large Margin Objective 76
8.4 Semi-supervised Convex Training for Structured SVMs 77
8.4.1 Least Squares Convex Objective 77
8.4.2 Semi-supervised Convex Objective 78
8.5 Efficient Optimization Strategy 79
8.6 Implementation Details 79
8.7 Experimental Results 80
8.7.1 Experimental Design 80
8.7.2 Results 81
8.8 Contributions 83
8.9 Conclusion 83

9 Conclusions and Future Directions **84**
9.1 Summary of Contributions 84
9.2 Future Directions 84
9.2.1 Variations of Structured Boosting 84
9.2.2 Multilingual Dependency Parsing 85
9.2.3 Domain Adaptation 85
9.2.4 A Unified View of My Current Work 86

Bibliography **87**

List of Tables

2.1 Static features 15

4.1 PARSEVAL measures 38
4.2 Dependency tree accuracy measures 39

5.1 Evaluation results on CTB4 49
5.2 Performance of alternative models 49

6.1 Accuracy on CTB4-10 dev set (%) 59
6.2 Accuracy on CTB4-10 test set (%) 59
6.3 Accuracy on CTB4-10 training set (%) 59
6.4 Comparison with the approach in Chapter 5 on CTB4-10 (%) 60

7.1 Boosting with static features (%) 68
7.2 Boosting with dynamic features (%) 69
7.3 Comparison with state-of-the-art (%) 70

8.1 Size of experimental data (# of sentences) 81
8.2 Supervised and semi-supervised dependency parsing accuracy on Chinese (CTB4) (%) 82
8.3 Supervised and semi-supervised dependency parsing accuracy on English (PTB) (%) 82

List of Figures

2.1 A dependency tree 8
2.2 A constituency tree 8

3.1 The AdaBoost boosting algorithm 31
3.2 Similar words of *keystone* 33

5.1 A dependency parsing algorithm 48

Chapter 1

Introduction

Learning syntactic parsers has been an active research area for more than a few decades. My research is focused on developing effective and efficient machine learning algorithms for inferring dependency parsers from natural language data, e.g., discovering dependency structure from raw text. I present several machine learning approaches to dependency parsing, which can be implemented efficiently, are extensible through the feature set and provide state-of-the-art accuracy.

1.1 Motivation

Syntactic parsing is a fundamental problem in natural language processing (NLP), where one maps a sentence from a non-structured linear form into a structured form. The structured form simplifies the process of extracting meaning from text. (Henceforth, I will use the term "parsing" to refer to "syntactic parsing".) Parsing has been an active area of research since the 1960s, not only because parsing output is very useful for many other NLP applications, such as machine translation (Ding and Palmer, 2005), information extraction (Culotta and Sorensen, 2004), question answering (Pinchak and Lin, 2006), coreference resolution (Bergsma and Lin, 2006) and query segmentation (Bergsma and Wang, 2007), but also because parsing itself is a very interesting research area.

Ambiguities, which abound in natural language, are a major problem in parsing. For example, for the sentence *"I saw her duck"*, there are three interpretations:

- I saw her pet duck.
- I saw her bend over.
- I cut her pet duck with a saw.

Before the 1990s, the predominant approaches to parsing were *symbolic*, where a hand-coded grammar was a necessary component (Allen, 1987). Symbolic parsers may perform well in a specific domain. However, the performance of hand-coded parsers drops off dramatically when they are applied to broader domains. The reason is that symbolic parsers suffer two main problems. One is that they do not deal with ambiguities well. They rely on selectional preferences and heuristic grammar rules to resolve ambiguities, but there are often many different parses that satisfy the same number of selectional preferences and meet the same heuristic criteria. The second problem is that they cannot parse sentences that are not covered by the specified grammar rules. Both of these problems, however, can be solved simultaneously by using statistical/machine learning approaches. First, by using a scoring function to select the most preferred parse tree, a statistical parser deals with ambiguities in a more principled way (see more discussion below in Section 2.4). Second, any parse tree is potentially allowed in a statistical parser.

Statistical approaches have been playing an increasingly prominent role in parsing since the 1990s. Parsing accuracy has increased significantly due to the introduction of machine learning methods in recent years. Parsing is an especially challenging and important task for machine learning, since it involves complex structured outputs (parse trees) and limited training data (manually constructed *treebanks*), and yet machine learning methods have proved to provide the best approach to obtaining robust parsers for real data. From the machine learning perspective, parsing can be considered as a structured classification problem. In a parsing model, during the learning phase, parameters defining the model are estimated using some machine learning criterion. During the classification phase, a parser takes a test sentence and the learned parameters as input and outputs the most preferred parse tree for that sentence.

Although machine learning approaches have shown significant improvement in increasing parsing accuracy, there are still some drawbacks in existing methods. First, in almost all previous statistical parsers, part-of-speech tags, either obtained from an external tagger or generated within the parser, are always considered to be a necessary component, since POS tags play an important role in dealing with the data sparseness problem. Unfortunately, for some languages, such as Chinese or Japanese, POS tags are harder to assign because there is more ambiguity and fewer morphological clues. For example, in Chinese, a single word is often combined with many other words and the combined items often have different POS tags. Alternative smoothing approaches are needed to handle the data sparseness problem in these languages. Second, there are weaknesses in current large margin training approaches to parsing, including: the introduction of an exponential number of constraints, corresponding to each possible parse tree of each sentence; evaluating the standard large margin training loss only at the global tree level, without penalizing any specific component in an incorrect parse tree; and creating serious over-fitting problems due to the large number of bi-lexical (word-pair) features. Although smoothing techniques have been widely used in probability models, they have not been used in large margin frameworks. Third, recent global training algorithms for learning structured predictors have been applied to parsing. However, the drawbacks of these global training algorithms are that they are specialized, complex to implement and expensive to run. For a complex task like parsing, these drawbacks are significant. Finally, although a great deal of progress has been made on semi-supervised learning for parsing, available semi-supervised training algorithms for parsing are either too expensive or the results are not as good as expected.

My thesis research is based upon the observation that there are still some limitations in existing statistical approaches to parsing and this has motivated me to employ and develop advanced supervised and semi-supervised machine learning algorithms to train accurate dependency parsers from natural language data.

1.2 Main Contributions

This thesis addresses the problem of learning dependency parsers from language data. The main contributions of the thesis are summarized as follows.

- First, I present a conditional probability parsing model that uses a maximum likelihood Markov network training approach to dependency parsing (Wang et al., 2005). The resulting model is similar to a maximum entropy Markov model. In this approach, whenever a dependency link between two words is generated, it considers the link labels of the neighbors of the two words. Thus the search for a parse tree with the highest probability is equivalent to finding a tree with the highest score of sum over all the dependency links in the logarithmic space. Unlike previous generative parsing models that compute the joint probability of a parse tree and a sentence, I find a tree which has the highest conditional probability given the sentence. Without relying on part-of-speech tags or grammatical categories to deal with the data sparseness issue in parsing, I apply similarity-based smoothing to the conditional parsing model instead. To the best of my knowledge, the similarity-based smoothing technique has not been applied to parsing before.

- Next, I propose an improved large margin training approach to learning dependency parsers, which is an extension to the standard large margin training (Wang et al., 2006). Specifically, I incorporate local errors of a parse tree that enforce the correctness of each individual link, rather than just scoring the whole parse tree. This approach only introduces a polynomial number of constraints, instead of an exponential number, where these constraints capture local errors with more detailed parse tree information. To deal with sparse data, I smooth the lexical parameters according to their underlying word similarities, introducing similarity-based smoothing technique into the large margin framework. The refined new large margin training objective— with fewer constraints and a better regularizer— provides better parsing results and is computationally less expensive than standard training formulations.

- Then, I present a simpler and much more efficient approach for training dependency parsers by applying a boosting-like procedure to standard supervised training methods (Wang et al., 2007). I call this approach *structured boosting*. Unlike previous structured training techniques, which are based on specialized training algorithms that are complex to implement and expensive to run, structured boosting provides global parsing accuracy with only local training cost. It does not require the underlying training algorithm be modified, while still ensuring the training outcome is directly influenced by the resulting accuracy of the parser. By using logistic regression as an efficient base link classifier (for predicting dependency links between word pairs), I am able to efficiently train a dependency parsing model that achieves state-of-the-art results in English, and surpasses state-of-the-art in Chinese.

- Finally, I present a novel, convex, semi-supervised large margin training algorithm for learning dependency parsers, where I can make use of both labeled and unlabeled data resource (Wang et al., 2008). A large number of distinct approaches to semi-supervised training algorithms have been investigated, such as self-training (Yarowsky, 1995; Charniak, 1997; Steedman et al., 2003; McClosky et al., 2006a), generative models (Klein and Manning, 2002; Klein and Manning, 2004; Smith and Eisner, 2005a), semi-supervised support vector machines (S3VM) (Bennett and Demiriz, 1998; Altun et al., 2005; Xu and Schuurmans, 2005), graph-based algorithms and multi-view algorithms (Zhu, 2005). In recent years, many researchers have put effort into developing algorithms for S3VMs in particular. However, the standard objective of an S3VM is non-convex on unlabeled data, thus requiring sophisticated global optimization heuristics to obtain reasonable solutions. Instead of devising various techniques for coping with non-convex loss functions as done by other researchers, I approach the problem from a different perspective. In particular, I investigate a semi-supervised approach for structured large margin training, where the objective is a combination of two convex functions, a structured large margin loss on labeled data and a least squares loss on unlabeled data. I apply the resulting semi-supervised convex objective to dependency parsing, and obtain significant improvement over the corresponding supervised structured SVM.

In summary, I present three supervised and one semi-supervised approach to learning dependency parsers for both English and Chinese. These approaches can be implemented efficiently, are extensible through the feature set and provide state-of-the-art accuracy.

1.3 Thesis Outline

Below is a brief summary of the remainder of this thesis.

Chapter 2. Background on Parsing: First, I briefly introduce the problem of natural language parsing, parse tree representations, parsing algorithms and the role of scoring functions in parsing.

Chapter 3. Background on Learning to Parse: Then, I survey the dominant machine learning approaches that have been applied in the area of dependency parsing, from early statistical approaches, local training methods to global training methods and other relevant machine learning techniques, such as boosting, similarity-based smoothing and unsupervised/semi-supervised learning.

Chapter 4. Experimental Setup: As a final preliminary step, I briefly discuss the data sets and the evaluation metrics I will use for evaluating parsers' performance to be reported in later chapters.

Chapter 5. Lexicalised Dependency Parsing: In this and the following three chapters, I describe a few advanced statistical approaches I have investigated for natural language dependency parsing. In particular, I introduce a generative approach to purely lexicalised dependency parsing that is augmented with similarity based smoothing, but no POS tags or grammatical categories are used.

Chapter 6. Extensions to Large Margin Dependency Parsing: Next, I present an improved large margin approach for learning dependency parsers from treebank data, which is an extension to standard large margin training. I introduce both the idea of capturing the local parse tree errors and using word similarities in the large margin framework.

Chapter 7. Training Dependency Parsers via Structured Boosting: I then present a simpler and more efficient approach to training dependency parsers by applying a boosting-like procedure to standard supervised training methods. By using logistic regression as an efficient base classifier (for predicting dependency links between word pairs), one is able to efficiently train a dependency parsing model, via structured boosting, that achieves state-of-the-art results in English, and surpasses state-of-the-art in Chinese.

Chapter 8. Semi-supervised Convex Training for Dependency Parsing: Finally, I propose a novel semi-supervised training algorithm for learning dependency parsers. By combining a convex large margin training loss on labeled data with a convex least squares loss on unlabeled data, I obtain a discriminative, convex, semi-supervised large margin training algorithm for dependency parsing.

Chapter 9. Conclusions and Future Directions: To conclude, I give a brief review of the main contributions of the thesis and discuss future research directions.

Chapter 2

Background on Parsing

This thesis focuses on automatically inferring dependency structure from natural language data. In this chapter I will briefly introduce the syntactic representations of a parse tree, parsing algorithms, the scoring functions which specify the most preferred parse tree, the features which are used in a scoring function, and finally treebanks, which are used to evaluate parsers.

2.1 What Is a Parser?

Generally speaking, parsing is the process of analyzing an input sentence in order to determine its syntactic structure, which in most cases is represented as a tree structure. The two dominant syntactic tree representations are *constituency trees* and *dependency trees* (Manning and Schutze, 1999). Thus, the goal of a parser is to analyze an input sentence and output the corresponding (most preferred) parse tree.

After more than a decade of research, many statistical approaches have been successfully applied to natural language parsing. These approaches mainly differ in three aspects: the output representations (i.e., what linguistic structure is used to represent a parse tree?), the parsing algorithms (i.e., which parsing algorithms can be used and how efficient are they?), and the scoring functions (i.e., how can alternative parses be scored so that the best scoring parses correspond to humans' preferred interpretations?). All of these three elements are necessary components of a good parser.

2.2 Syntactic Representations of a Parse Tree

The first question one would ask about a parser is, what does the output of the parser look like? For many years, researchers have been using a tree structure to represent the output

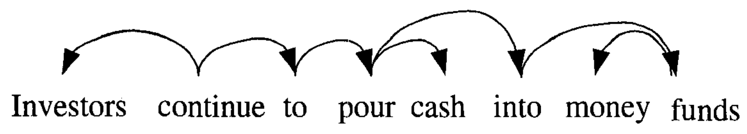

Figure 2.1: A dependency tree

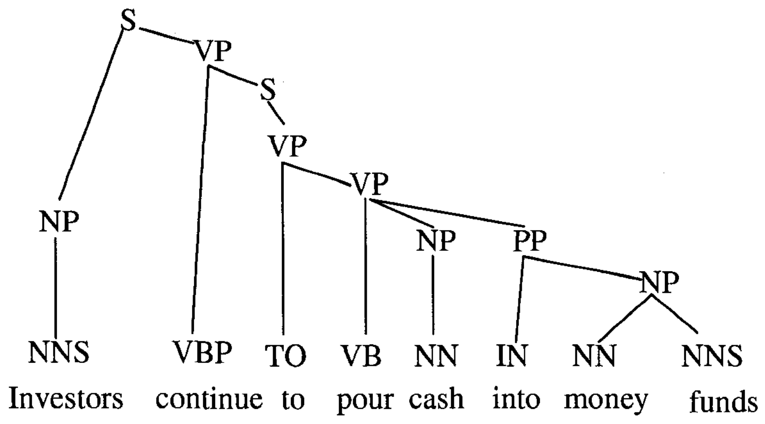

Figure 2.2: A constituency tree

of a parsed sentence. Many different formalisms have been proposed to represent parsing output, but most machine learning work has focused on just two: constituency trees (also called phrase structure trees) and dependency trees. My own work has focused on dependency trees for reasons I outline below.

2.2.1 Dependency Structures vs. Constituency Structures

The dominant tradition within modern NLP has been to use a constituency structure to represent the syntactic structure of a sentence. However, using a dependency tree to describe linguistic structures in terms of dependencies between words is a viable alternative tradition. Dependency structures have a remarkably longer history in comparison with constituency structures. They can be traced back to two thousand years to the notion of dependency rooted in ancient Greek and Indian linguistic traditions.

As shown in Figure 2.1, in a dependency structure, the basic units of a sentence are the syntactic relationships (aka. head-modifier or governor-dependent or regent-subordinate relations) between two individual words, where the relationships are expressed by drawing links connecting individual words (Manning and Schutze, 1999). The direction of each link points from a head word to a modifier word, and each word has one and only one head. In a dependency tree, one word is the head of the sentence (e.g., in Figure 2.1, the headword of the sentence is *continue*), and all other words are either a dependent of that word, or a dependent of some other word that connects to the headword through a sequence of dependencies. In this sense, a dependency structure can be considered as a rooted, directed graph. The dependency links in the tree (shown in Figure 2.1) are not labeled. Some researchers have also considered *labeled* dependency trees, where each link is assigned a corresponding *functional category* (e.g., subj—subject, obj—object). Although the planarity constraint (*projectivity* or *no crossing arcs*)—if a word u depends on a word v, then all words between u and v are also subordinate to v— is enforced in most previous work, non-projective dependency structures are needed to account for long-distance dependencies (e.g., in English), or free word order (e.g., in Czech).

By contrast, in a constituency structure, constituents are the basic units of a sentence. A constituent is defined as a word or a group of words that functions as a single unit within a hierarchical structure. The syntactic relationships are expressed by breaking up the sentence into constituents (phrases). For example, Figure 2.2 shows a constituency structure for the same sentence in Figure 2.1. The non-terminal symbols (labels of internal nodes) correspond to syntactic categories such as noun phrase (NP), verbal phrase (VP) or prepositional phrase (PP) and part-of-speech tags like nouns (NN), verbs (VB) and prepositions (IN). The terminal symbols (leaves) are the words of the sentence.

Although the basic units in dependency trees and constituency trees are different, the two types of trees are closely related. Indeed, their structures imply each other. In a dependency tree, a word plus its dependents constitutes a phrase. In a constituency tree, the dependencies between words are implied by phrases. A constituency tree can be automatically converted into a dependency tree using a set of simple predefined head-rules (Magerman, 1995; Collins, 1999; Yamada and Matsumoto, 2003; Bikel, 2004). On the other hand, a dependency tree can also be converted into a constituency tree by using simple heuristic rules and treebank-specific information (Xia and Palmer, 2001). It has been shown that dependency structures and constituency structures are strongly equivalent (Gaifman, 1965). (Note that although a constituency tree corresponds to a dependency tree, a dependency tree usually maps to multiple constituency trees.)

In recent years, there has been an increasing interest in dependency-based as opposed to constituency-based approaches to syntactic parsing, with application to a wide range of research areas and different languages. For example, Fox (2002) found that the dependency structures of a pair of translated sentences have a greater degree of cohesion than their constituency structures. Cherry and Lin (2003) exploited such cohesion between the dependency structures to improve the quality of word alignment of parallel sentences. Dependency relations have also been found to be useful in information extraction (Culotta and Sorensen, 2004; Yangarber et al., 2000), machine translation (Ding and Palmer, 2005) and coreference resolution (Bergsma and Lin, 2006).

There are many reasons why dependency structures have advantages over constituency and other syntactic structures. First, the original, fundamental notion in traditional grammar of "parsing a sentence into subject and predicate" is based on lexical relations between word-pairs rather than constituent relations (Manning and Schutze, 1999). Second, the indirect representation of constituency structures makes unsupervised language acquisition very difficult, whereas dependency structures are much clearer and hence easier to understand. Consequently, many researchers who are working on (unsupervised) grammar induction focus on dependency structures instead of constituency structures (Klein and Manning, 2004; Smith and Eisner, 2005b; Smith and Eisner, 2006). In addition, although constituency structures have dominated linguistics theory for the last few decades, dependency-based theories can often be found embedded in various aspects of constituency-based approaches to natural language parsing. For example, it has been found that the performance of a statistical parser can be greatly improved by adding information about lexical dependencies between headwords in a sentence (Collins, 1996). Lexical dependency information has also been used in more recent statistical parsers as well (Collins, 1999; Charniak, 2000). Furthermore, it has been argued in (Lin, 1995) that dependency-based evaluation is much more meaningful for the applications that use parse trees, since semantic relationships are generally embedded in dependency relationships. Finally, syntactic relations between word-pairs in dependency structures are closer to one's ultimate goal of extracting meaning from sentences (Yuret, 1998).

2.2.2 Other Syntactic Representations

Beyond the constituency structures and dependency structures discussed above, there are many other syntactic formalisms that have been proposed in the computational linguistic literature. Prominent examples include Link Grammars (Lafferty et al., 1992), Head-driven Phrase Structure Grammars (HPSGs) (Pollard and Sag, 1994), Tree-Adjoining Grammars (TAGs) (Joshi et al., 1975), and Combinatory Categorial Grammars (CCGs) (Steedman, 2000). These representations are more complex, and consequently have not been well investigated as potential targets for machine learning based approaches, although some preliminary attempts have been made (Copestake and Flickinger, 2000; Vijay-Shanker, 1993; Clark and Curran, 2004). Much more work has been devoted to learning to predict constituency or dependency structures, and, as mentioned, I have focused on dependency structures in my current work.

2.3 Parsing Algorithms

Parsing is a complicated task; with increasing sentence length, the number of possible parse trees increases exponentially. A parse tree can be constructed either top-down or bottom-up. A parser's goal is to search through the space of all the possible parse trees to find the most appropriate tree for the sentence. Almost all current parsing algorithms that can effectively deal with the problem of ambiguities—i.e., when a sentence has more than one parse—are based on dynamic programming approaches.

2.3.1 Dynamic Programming

Dynamic Programming (DP) is a class of algorithms that apply a table-driven method to solve problems by combining solutions to sub-problems (Bellman, 1957). The intuition of using a dynamic programming strategy is that a large problem can be solved by appropriately combining the solutions to various sub-problems. In the case of constituency parsing, a table is used to store subtrees for each of the various constituents in the input as they are discovered. Once all the subtrees have been systematically filled into the table, the whole parse tree can be obtained by combining the different subtrees. The advantage of this approach is that these subtrees are only discovered once, stored, and then used in all the parse trees calling for that constituent. This solves the re-parsing problem (subtrees are looked up, not re-parsed) and allows for an approach to handling the ambiguity problem (the parsing table implicitly stores all possible parses by storing all the constituents with links that enable the parses to be reconstructed). There are three well-known parsing algorithms using dynamic programming (Jurafsky and Martin, 2000): the Cocke-Kasami-Younger (CKY) algorithm, the Graham-Harrison-Ruzzo (GHR) algorithm and the Earley algorithm. Although originally developed for constituency parsing, these algorithms can be applied to dependency parsing with minor modifications.

2.3.2 Dependency Parsing Algorithms

A dependency parsing algorithm is, in effect, a dynamic programming algorithm that has the goal of producing a maximum weight spanning tree subject to the constraints (Eisner, 1996). The weight of a tree is computed by a scoring function—which will be introduced in Section 2.4 below—whose role is to specify the most preferred parse from among the set of all possible, legal parse trees for a sentence. There are a wide variety of dependency parsing algorithms available with different computational cost. They range from Eisner's $O(n^3)$ projective dependency parsing algorithm (Eisner, 1996), where n is the length of the sentence, all the way up to an $O(n^5)$ chart parsing algorithm (Jurafsky and Martin, 2000).

With minor modifications, traditional constituency parsing algorithms (e.g., a CKY parser) can be used for dependency parsing, with $O(n^5)$ complexity. The basic idea is simply to treat dependencies as constituents. In a dependency chart parsing algorithm, each chart entry consists of the head, the modifier and all their left and right dependents. Specifically, the dependency parser constructs a set of chart items, each of which has a head word. Each chart item is a 4-tuple: (*low, head, high, score*) where *low*, *head* and *high* (*low* $\leq$ *head* $\leq$ *high*) are positions of words in a sentence and *score* is a non-negative number. This means that there exists a dependency tree that spans the words from *low* to *high* with the given *score*, and rooted at the position *head*. Initially, the parsing algorithm creates a chart item for each individual word in the input sentence. The items are then combined with the existing items that are adjacent items to their left. The combined item has the span of the union of the two components and may take either item's head as its head. Thus, a dependency tree for the whole sentence can then be built up in a bottom-up manner, by successively combining adjacent chart items into bigger ones. A dependency parsing algorithm implemented in this way has $O(n^5)$ complexity, in the worst case.

The novel probabilistic dependency parsing algorithm described in (Eisner, 1996) is a modified chart-parsing algorithm, with $O(n^3)$ complexity. The modification is that instead of storing spans of subtrees, it stores spans of half subtrees. A span is defined as a substring such that no interior word links to any word outside the span. The underlying idea is that in a span, only the end-words are active, i.e., those that still need a head. Either one or both of the end-words can be active.

The difference between these algorithms has to do with the linguistic constraints they can enforce and the types of features they can use during dynamic programming. Faster algorithms enforce fewer linguistic constraints and need to use a more restricted class of features. For example, when attaching a preposition, Eisner's $O(n^3)$ parser cannot access the noun after the preposition, whereas an $O(n^5)$ chart parser is able to do so and can therefore correctly disambiguate some prepositional phrase attachments that the cheaper $O(n^3)$ parser cannot handle appropriately.

The above algorithms I have discussed are projective dependency parsing algorithms. Several non-projective dependency parsing algorithms have been proposed. For example, McDonald et al. (2005b) propose an $O(n^2)$ non-projective dependency parsing algorithm using the Chu-Liu-Edmonds (1965; 1967) maximum spanning tree algorithm. I only use projective dependency parsing algorithms in my thesis work.

2.4 Scoring Parses

In principle, dependency parsing algorithms can enumerate all legal parses. For a given sentence however, there are usually exponentially many possible parse trees. Usually, a scoring function is introduced to specify the most preferred parse tree among legal parse trees. That is, the scoring function plays the important role of specifying how potential parsing ambiguities are to be resolved. The main computational question is, can a scoring function be integrated into current dependency parsing algorithms while maintaining an efficient dynamic programming approach to computing the most preferred (i.e., highest scoring) parse? The answer is yes. If the scoring function decomposes into a combination of local link scores, then one can use dependency parsing algorithms to efficiently compute the legal parse that attains a maximum score.

2.4.1 Local Scoring Functions

To discuss scoring functions in more detail, it will be advantageous to introduce some notation. Given a sentence $X = (x_1, ..., x_n)$ (x_i denotes each word in the sentence), I am interested in computing a directed dependency tree, Y, over X. In particular, I assume that a directed dependency tree Y consists of ordered pairs $(x_i \rightarrow x_j)$ of words in X such that each word appears in at least one pair and each word has in-degree at most one. As discussed in Section 2.2.1, dependency trees are usually assumed to be projective, which means that if there is an arc $(x_i \rightarrow x_j)$, then x_i is an ancestor of all the words between x_i and x_j in the sentence X. Let $\Phi(X)$ denote the set of all the directed, projective trees that span on X. The parser's goal is then to find the most preferred parse; that is, a projective tree, $Y \in \Phi(X)$, that obtains the highest "score". In particular, one assumes that the score of a complete spanning tree Y for a given sentence, whether probabilistically motivated or not, can be decomposed as a sum of local scores for each link (a word pair) (Eisner, 1996; Eisner and Satta, 1999; McDonald et al., 2005a). Given this assumption, the parsing problem reduces to

$$\begin{aligned} Y^* &= \arg\max_{Y \in \Phi(X)} score(Y|X) \\ &= \arg\max_{Y \in \Phi(X)} \sum_{(x_i \to x_j) \in Y} score(x_i \to x_j) \end{aligned} \tag{2.1}$$

where the $score(x_i \to x_j)$ can depend on any measurable property of x_i and x_j within the sentence X. This formulation is sufficiently general to capture most dependency parsing models, including probabilistic dependency models of (Eisner, 1996), non-probabilistic models (McDonald et al., 2005a), and my own work in this thesis.

For standard scoring functions, particularly those used in non-generative models, one further assumes that each link score in (2.1) can be decomposed into a weighted linear combination of features

$$score(x_i \to x_j) = \boldsymbol{\theta} \cdot \mathbf{f}(x_i \to x_j) \tag{2.2}$$

where $\mathbf{f}(x_i \to x_j)$ is a feature vector for the link $(x_i \to x_j)$, and $\boldsymbol{\theta}$ are the weight parameters to be estimated during training. (Parameter estimation will be discussed in Chapter 3 below. The larger set of "dynamic" features often used in generative models will be discussed in Section 2.4.3 and revisited in greater detail in Chapter 5 below.)

2.4.2 Static Features

Of course, the specific features used in any real situation are critical for obtaining a reasonable dependency parser. The natural sets of features to consider in this setting are very large, consisting at the very least of features indexed by all possible lexical items (words). For example, natural features to use for dependency parsing are indicators of each possible word pair

$$f_{uv}(x_i \to x_j) = 1_{(x_i = u)} 1_{(x_j = v)}$$

which allows one to represent the tendency of two words, u and v, to be directly linked in a parse. Here $1_{(x=u)}$ denotes the indicator function such that

$$1_{(x=u)} = \begin{cases} 1 & \text{if } x = u; \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, there are a large number of lexical features, which causes sparse data problems below when I begin to consider how to learn parsers from data (in Chapter 3). A

Basic Uni-gram Features	p-word, p-pos p-word p-pos c-word, c-pos c-word c-pos
Basic Bi-gram Features	p-word, p-pos, c-word, c-pos p-pos, c-word, c-pos p-word, c-word, c-pos p-word, p-pos, c-pos p-word, p-pos, c-word p-word, c-word p-pos, c-pos
In Between POS Features	p-pos, b-pos, c-pos
Surrounding Word POS Features	p-pos, p-pos+1, c-pos-1, c-pos p-pos-1, p-pos, c-pos-1, c-pos p-pos, p-pos+1, c-pos, c-pos+1 p-pos-1, p-pos, c-pos, c-pos+1

Table 2.1: Static features

p-word/c-word: word of parent/child node in a dependency tree.
p-pos/c-pos: POS of parent/child node in a dependency tree.
p-pos-1/p-pos+1: POS to the left/right of parent in a sentence.
c-pos-1/c-pos+1: POS to the left/right of child in a sentence.
b-pos: POS of a word in between parent and child nodes.

standard way to handle sparseness is to combine features via *abstraction*. A common strategy for abstraction is to use parts-of-speech (POS) to compress the feature set, for example by only considering the tag (instead of the word) of the parent

$$f_{pv}(x_i \rightarrow x_j) = 1_{(POS(x_i)=p)} 1_{(x_j=v)}$$

In general, the most important aspect of a link feature is simply that it measures something about a candidate word pair that is predictive of whether the words will actually be linked in a given sentence. Thus, many other natural features, beyond parts-of-speech and abstract grammatical categories, immediately suggest themselves as being predictive of link existence. For example, one very useful feature is simply the degree of association between the two words as measured by their pointwise mutual information (PMI) computed from natural word co-occurrence statistics (more on this in Chapter 3 below)

$$f_{PMI}(x_i \rightarrow x_j) = PMI(x_i, x_j)$$

Another useful link feature is simply the distance between the two words in the sentence; that is, how many words they have between them

$$f_{dist}(x_i \rightarrow x_j) \;=\; |position(x_i) - position(x_j)|$$

In fact, the likelihood of a direct link between two words diminishes quickly with distance, which motivates me to use more rapidly increasing functions of distance, such as the square

$$f_{dist2}(x_i \rightarrow x_j) \;=\; (position(x_i) - position(x_j))^2$$

All the above simple features can be easily used by a scoring function to compute the weight of a link. In practice, I can actually use a much larger set of static features to score a dependency link. For example, the work I will propose in Chapter 7 uses almost all the features described in (McDonald et al., 2005a). These static features are given in Table 2.1.

2.4.3 Dynamic Features

"Dynamic" features (also called non-local features) are features that take into account the *labels* (i.e., the links) of (some) of the surrounding components when predicting the label of a target component. In particular, when predicting the label of a target component $x_{i,j} \in X$ from a composite object X (that is, whether or not there is a link between two words x_i and x_j in the sentence X, and what the orientation of this link might be) one can assume that the labels for *some* other components, say $x_{i,j-1} \in X$, have already been computed. The only constraint is that the required neighboring labels must always be available before attempting to label $x_{i,j}$.

The easiest way to illustrate the concept is in a sequential labeling task, like part-of-speech tagging: Given a sentence $X = x_1, ..., x_n$, the goal is to predict the corresponding tag sequence $Y = y_1, ..., y_n$. Here the *preceding* tags can be used as features for the current word under consideration—e.g. in a maximum entropy Markov model (MEMM) (McCallum et al., 2000)—while still permitting an efficient dynamic programming algorithm (Viterbi decoding) to be used for structured prediction. Alternatively, one could use the *following* tags as features or use both the preceding and following tags (Toutanova et al., 2003). The idea of dynamic features, however, is more general than sequence labeling and maximum conditional likelihood training.

For dependency parsing, dynamic features can also be easily employed. For example, when considering a possible link label that connects a head word to a subordinate word, one will always have access (in any standard parsing algorithm) to the existing children of the head that occur between the two words under consideration. In this case, the number and types of pre-existing subordinate children are valid features that can be used to predict whether the new head-subordinate link should occur, which turns out to be a very informative feature for link prediction in parsing. These dynamic features are used in the work of (Collins, 1997; McDonald and Pereira, 2006), and in my own work to be presented in later chapters. In particular, I will make extensive use of dynamic features of this kind in Chapter 5 below, which employs a generative model of dependency tree construction.

Although the idea of using dynamic features is not new for parsing (Collins, 1997; Magerman, 1995), it is still not as widely appreciated as perhaps it should be. In fact, even though the possibility is not always used in non-generative approaches to parsing (McDonald et al., 2005a), it yields immediate improvements when subsequently reintroduced (McDonald and Pereira, 2006).

2.5 Treebanks

A *treebank* is a collection of sentences manually annotated with the "correct" parse tree. Treebanks have been widely used by researchers for training and evaluating their parsers for more than a decade.

There are treebanks available for different languages, such as the Penn Treebank (Marcus et al., 1993) for English, the Penn Chinese Treebank 4.0 (Palmer *et al.*, 2004) and Penn Chinese Treebank 5.0 (Palmer *et al.*, 2005) for Chinese, and the Czech Prague Dependency Treebank (Hajic, 1998) for Czech. The Penn Treebank and Penn Chinese Treebank contain constituency trees, which can be automatically converted into dependency trees using predefined head rules (Collins, 1999; Yamada and Matsumoto, 2003; Bikel, 2004), as briefly discussed in Section 2.2.1. There are also dependency treebanks available for many other languages from CoNLL-X Shared Task (Buchholz and Marsi, 2006).

With treebanks, it is trivial to extract the grammatical knowledge that is implicit in the example parses. More importantly, by making the evaluation of parsers more standardized, treebanks have been playing an important role for the growing interest in parsing since the last decade when the first treebank was constructed. However, treebanks are an extremely precious resource. For example, the average cost of producing an English treebank parse can run as high as 30 person-minutes per sentence (more than 20 words on average). Therefore, there is a need for machine learning methods that can either learn accurate parsers with limited treebank data, or exploit auxiliary unlabeled data (raw text), which is plentiful.

Chapter 3

Background on Learning to Parse

Over the past decade, there has been tremendous progress on learning parsing models from treebank data (Magerman, 1995; Collins, 1996; Collins, 1997; Charniak, 1997; Ratnaparkhi, 1999; Charniak, 2000; McDonald et al., 2005a). Most of the early work in this area was based on postulating *generative* probability models of language that included parse structures (Magerman, 1995; Collins, 1997; Charniak, 1997). Learning in this context consisted of estimating the parameters of the model with simple likelihood based techniques, but incorporating various smoothing and back-off estimation tricks to cope with the sparse data problems (Collins, 1997; Bikel, 2004). Subsequent research began to focus more on *conditional* models of parse structure given the input sentence, which allowed discriminative training techniques such as maximum conditional likelihood (i.e. "maximum entropy") to be applied (Ratnaparkhi, 1999; Charniak, 2000). In fact, I will show in my own work in Chapter 5 that effective conditional parsing models can be learned using relatively straightforward "plug-in" estimates, augmented with similarity based smoothing. Currently, the work on conditional parsing models appears to have culminated in large margin training approaches (Taskar et al., 2003; Taskar et al., 2004b; Tsochantaridis et al., 2004; McDonald et al., 2005a), which demonstrate state-of-the-art performance in English dependency parsing (McDonald et al., 2005a).

Despite the realization that maximum margin training is closely related to maximum conditional likelihood for conditional models (McDonald et al., 2005a), a sufficiently unified view has not yet been achieved that permits the easy exchange of improvements between the probabilistic and non-probabilistic approaches. For example, smoothing methods have played a central role in probabilistic approaches, as shown in the previous work of (Collins, 1997) and in my own work which I will present in Chapter 5, and yet they are not being used in current large margin training algorithms. I address this issue in my current work in Chapter 6 below. Other unexploited connections also exist between large margin and probabilistic approaches. For example, it also turns out that probabilistic approaches pay closer attention to the individual errors made by each component of a parse, whereas the training error minimized in the large margin approach—the "structured margin loss" (Taskar et al., 2003; Tsochantaridis et al., 2004; McDonald et al., 2005a)—is a coarse measure that only assesses the total error of an entire parse rather than focusing on the error of any particular component. I will address this in Chapter 6.

In this chapter, I first discuss some most commonly used learning methods for parsing: early statistical approaches, reranking, local training methods and global training methods. Here I begin to show the connections between these alternative approaches (and exploit these connections in more detail in my work in Chapter 5, Chapter 6, Chapter 7 and Chapter 8). Then I will introduce other relevant machine learning techniques for parsing: boosting, distributional word similarity and unsupervised/semi-supervised learning methods.

3.1 Introduction

The problem of learning a dependency parser can be formulated as follows. One is given a set of annotated sentences $(X_1, Y_1), ..., (X_N, Y_N)$ from a treebank, where each sentence X_i consists of a word string and each annotation Y_i is a complete labeling of a link label (left link, right link, no link) for every pair of words in the sentence. The goal of learning is to estimate the parameters $\boldsymbol{\theta}$ of the scoring function used in the parsing model. In particular, one seeks values for the parameters that can accurately reconstruct the training parses, and more importantly, are also able to accurately predict the dependency parse structure on future test sentences.

A sentence $X_i = x_{i,1}, ..., x_{i,n}$ and a dependency tree labeling $Y_i = y_{i,1,2}, ..., y_{i,n-1,n}$ can be decomposed into local examples

$$(x_{i,1}, x_{i,2}; y_{i,1,2})|_{(X_i,Y_i)}, \; ..., \; (x_{i,n-1}, x_{i,n}; y_{i,n-1,n})|_{(X_i,Y_i)}$$

consisting of arbitrary (not necessarily adjacent) word pairs and their link label (none, left, right) in context (X_i, Y_i). The context is important because accurately predicting a component label, even locally, requires the consideration of more than just the word pair itself, but also the surrounding words, and possibly even the labels of some of the surrounding words (McDonald and Pereira, 2006).

This decomposition facilitates a purely local approach to the dependency parsing problem that amounts to learning a scoring function (of the kind described in Section 2.4) from local training examples consisting of local contexts and their corresponding link labels. That is, given the original training data $(X_1, Y_1), ..., (X_N, Y_N)$, one can first break the data up into local examples

$$(x_{1,1}, x_{1,2}; y_{1,1,2})|_{(X_1,Y_1)},\ ...,\ (x_{i,j}, x_{i,k}; y_{i,j,k})|_{(X_i,Y_i)},\ ...$$

ignore the relationships between examples, and use a standard supervised learning algorithm to learn a local predictor $x_{i,j}, x_{i,k} \mapsto y_{i,j,k}$ in context (X_i, Y_i). For example, if one restricts attention to linear predictors (support vector machines or logistic regression models), one only needs to learn a weight vector $\boldsymbol{\theta}$ over a set of features defined on the local examples $\mathbf{f}(x_{i,j}, x_{i,k}, y_{i,j,k}; X_i, Y_i)$. Here, each feature f_m computes its value based on the component $(x_{i,j}, x_{i,k})$, the label $y_{i,j,k}$, in their context (X_i, Y_i), as described in Section 2.4. In this case, a multi-class support vector machine (Crammer and Singer, 2001) or logistic regression model (Hastie et al., 2001) could be trained in a conventional manner to achieve an accurate local prediction model. In fact, exactly this approach has been considered by many researchers—see Section 3.4 below. Then a dependency parser can be learned simply by combining a local link classifier with a dependency parsing algorithm to achieve global consistency. That is, the link classifier can play the role of the scoring functions as discussed in Section 2.4.

Thus, the goal of learning is to learn a link predictor (specified by a good set of weights $\boldsymbol{\theta}$) that yields an accurate parser. The different training algorithms investigated to date differ in whether they use static vs. dynamic features (as discussed in Sections 2.4.2 and 2.4.3), the training objective used, and whether training takes into account global parsing accuracy after dynamic programming, or merely trains a local link predictor from local examples only.

3.2 Early Statistical Approaches

Most statistical parsers in the late 1990s represented sentences as constituency structures (Collins, 1996; Collins, 1997; Charniak, 2000). More specifically, a set of rewrite rules, each of which has the form *category* $\longrightarrow$ *category**, are used to capture the regularities of word order in a sentence. The first rule starts with the *start symbol* S (for sentence). A rule rewrites the non-terminal symbol on the left-hand side as a sequence of syntactic categories or part-of-speech tags (e.g., NP, VP, PP or NN, VB, IN). In general, a non-terminal can be rewritten as one or more other syntactic categories or words. The possibilities for rewriting depend only on the category, but not on any surrounding context. So such constituency grammars are commonly referred to as *context-free grammars* (CFG). A *Probabilistic Context Free Grammar* (PCFG) is simply a CFG with probabilities added to the rules, indicating how likely different rewriting rules are (Manning and Schutze, 1999).

These earlier PCFG-based parsers differ in:

- how the statistics needed by the system are gathered;
- how those statistics are smoothed;
- the degree of supervision;
- how heavily the lexical information is used;
- and whether a hand-written grammar is needed.

But all these systems make use of a scoring function of the kind discussed in Section 2.4, based on dynamic features. The corresponding link scores are log probabilities of parse trees. Choosing the parse tree with the highest probability amounts to choosing the tree with the highest score as discussed in Section 2.4. These parsers vary greatly on how head word information, i.e., the one word that best represents the meaning of the constituent, is used to disambiguate possible parses for an input sentence. Black et al. (1993) introduces history-based parsing, in which learned decision-tree probability models are used to score the different derivations of sentences produced by a hand-crafted grammar. Jelinek et al. (1994) and Magerman (1995) also train history-based decision tree models from a treebank, but without using a hand-written grammar. Other early learned parsers use statistics of lexical information between pairs of head words combined with chart parsing techniques to achieve better performance (Collins, 1996; Collins, 1997; Charniak, 1997). I discuss some of these previous statistical parsers in more detail.

Magerman (1995) described a statistical parser based on decision-tree learning techniques. First, a decision tree model is defined as an interpolated n-gram model. Then the parsing problem is viewed as making a sequence of disambiguation decisions. The probability of a complete parse tree Y of a sentence X is the product of each decision (d_i) conditioned on all previous decisions:

$$P(Y|X) = \prod_{d_i \in Y} P(d_i|d_{i-1}, d_{i-2}, ..., d_1, X) \tag{3.1}$$

All decisions are pursued non-deterministically according to the probability of each choice, which is estimated using statistical decision tree models. In particular, the model parameters are estimated by counting the frequencies of each n-gram from a training corpus with a deleted interpolation smoothing technique (Manning and Schutze, 1999). Thus the parser is trained completely automatically from the treebank, without using any hand-crafted grammar. With a large amount of lexical information incorporated into the decision tree model, the resulting parser represents a substantial improvement over previous PCFG-based parsers.

Another statistical parser based on bigram lexical dependencies is presented by Collins (1996). The model is defined as:

$$P(Y|X) = P(B,D|X) = P(B|X)P(D|X,B) \tag{3.2}$$

where B is the set of baseNPs and D is the set of dependencies between pair of words. The method uses lexical information directly by modeling head-modifier relations between word-pair, which is similar to dependency grammars. The parameters of the baseNP model and the dependency model are estimated separately. The parsing algorithm is a simple bottom-up chart parser which uses dynamic programming algorithms, as discussed in Section 2.3.2. Here the parser finds the tree that maximizes (3.2) subject to the constraint that no crossing links are allowed in a dependency tree.

Charniak (1997) proposes another statistical parser which uses a generative parsing model, where it computes the joint probability of a sentence X and its parse tree Y, $P(X,Y)$, instead of computing the conditional probability $P(Y|X)$ as used in previous models (Magerman, 1995; Collins, 1996). Although this difference has no effect for parsing, it would be useful when these parsing systems are attached to another system and the parser is used as a language model. Another difference is that Charniak (1997) uses a formal treebank grammar (Charniak, 1996), while no such grammar is explicitly used in both (Magerman, 1995) and (Collins, 1996). Except for those differences, the three systems have much in common. In all the three cases the statistics are gathered from the training corpus and the lexical information predefined in a phrase is heavily used in its statistics. Furthermore, all three systems find the parse with the highest probability score according to the smoothed probability distribution they define. The differing performance between these systems is due to the different statistics they gather and the different smoothing techniques used for those statistics.

Collins (1997) proposed three generative, lexicalised, probabilistic parsing models that gave better performance than the previous conditional counterpart (Collins, 1996). In (Collins, 1997), he first introduces a model that is similar to (Collins, 1996) in structure, but with additional probabilities for generating the head and STOP events for each constituent.

The model is then extended to include a probabilistic treatment of both subcategorization and wh-movement. The advantage of this model over the previous one proposed by Charniak (1997) is that the latter may suffer the sparse data problem more seriously.

The fact that lexical information is very useful for constituency parsing (Magerman, 1995; Collins, 1996), which has caused increasing interest in studying the lexical affinities between words directly. Eisner (1996) proposes three distinct, lexicalised, probabilistic parsing models to investigate dependency relationships between word pairs: Model A—a bigram lexical affinity model where words attempt to modify each other; Model B—a sense tagging model where each word chooses a subcategorization/supercategorization frame and the model selects an analysis that satisfies all frames if possible; and Model C—a recursive generative model where each word generates its left and right dependents separately. The results show that the generative model (Model C), which is similar to (Collins, 1997), performs the best of the three. Subsequently, a novel $O(n^3)$ dependency parsing algorithm was presented by Eisner (1996), as discussed already in Section 2.3.2. No hand-written grammar is required.

3.3 Reranking

Ratnaparkhi (1999) observed that the correct parse almost always appeared in the top 20 completed parses produced by his algorithm. This observation led to great subsequent interest in parse reranking schemes. This work noticed that some features that are not easily incorporated in a generative model like those first considered in (Charniak, 1997; Collins, 1997), but are nevertheless still very useful for discriminating the correct tree for a sentence. Initially, it was not understood how to use such features in the original training algorithms so instead the problem was solved by using reranking as a postprocessing procedure. In reranking methods (Johnson et al., 1999; Collins, 2000; Shen et al., 2003; Charniak and Johnson, 2005), an initial parser is used to generate a number of candidate parses. A discriminative model is then used to choose the best parse among these candidates. In a reranking model, a tree is represented as an arbitrary set of features, without concerns about how these features interact or overlap, and without the need to define a derivation which takes these features into account. Thus features like those discussed in Section 2.4 could be used.

An obvious drawback of reranking is that an initial probabilistic parser is needed. In addition, the best parse is not necessarily in the list of the top k best parses. This approach has since been more or less subsumed by the more recent logistic regression and large margin approaches discussed in Section 3.4 and Section 3.5 below.

3.4 Local Training Methods

The decomposition of training data (sentences and their parses) into local examples (as described in Section 3.1) facilitates a purely local approach to the learning problem. That is, one can first break the data up into local examples, ignore the relationship between examples, and use a standard supervised learning algorithm to learn a local predictor. For example, if one restricts attention to linear predictors (e.g., support vector machines or logistic regression models), one only needs to learn a weight vector $\boldsymbol{\theta}$ over a set of features defined on the local examples, as discussed in Section 2.4. In this case, a multi-class support vector machine (Crammer and Singer, 2001) or logistic regression model (Hastie et al., 2001) could be trained in a conventional manner to achieve an accurate local prediction model. Then the local prediction model can be used to perform dependency parsing on sentences by combining with a parsing algorithm. Exactly this approach to combining local link predictors with dependency parsing algorithms has been tried, with some success, by many researchers—using support vector machines (Yamada and Matsumoto, 2003), logistic regression (aka. maximum entropy models) (Ratnaparkhi, 1999; Charniak, 2000), and generative probability models, such as the work described in (Collins, 1997) and my own work which will be presented in Chapter 5 later—to learn local scoring functions.

In principle, any machine learning approach can be used as a local link predictor. However, most researchers have focused their work on two linear predictors—logistic regression models and support vector machines—for dependency parsing. The main reason that these two models have been most commonly used is due to their ability to cope with a large set of features. I briefly introduce these two local training models in the remainder of this section.

3.4.1 Logistic Regression Models

Logistic regression models (aka. log-linear models, maximum likelihood exponential models or maximum conditional likelihood) have been widely used in many areas. In the NLP community, logistic regression models are also often called *maximum entropy* (maxent) models.

Logistic regression models are of the form

$$P(y|X) = \frac{\exp(\boldsymbol{\theta}^\top \mathbf{f}(X,y))}{\sum_{y'} \exp(\boldsymbol{\theta}^\top \mathbf{f}(X,y'))} \tag{3.3}$$

where X is an input context and y is a single output. The elements of f are the feature indicator functions that are true if a particular property of X, y is true. The parameters are θ, i.e., one adjustable weight per feature, as discussed in Section 2.4.

Logistic regression models are a common modeling technique for a variety of NLP tasks. There are at least three obvious advantages of logistic regression models over generative ones. First, logistic regression models can be easily updated by changing the set of features used, whereas arbitrary features cannot be easily incorporated in generative models. Second, logistic regression models are more capable of dealing with (inter-dependent) features of the inputs. By contrast, inter-dependent observations are difficult to represent in generative models, since these create intractable inference problems (i.e., intractable parsing problems) (Lafferty et al., 2001). Finally, logistic regression models are trained to minimize a loss function related to labeling error, which leads to smaller error in practice if enough training data is available. Generative models, on the other hand, are trained to maximize the joint probability of the training data, which is not necessarily closely related to the classification accuracy if the data is not generated by the model, which is usually the case in practice. Due to these advantages, logistic regression models are now much more commonly applied to parsing problems across different grammar formalisms (Ratnaparkhi, 1999; Charniak, 2000; Miyao and Tsujii, 2002; Riezler et al., 2002; Toutanova et al., 2002; Clark and Curran, 2003; Clark and Curran, 2004; Charniak and Johnson, 2005).

The first work that used a maximum entropy approach to learn a parser was conducted by Ratnaparkhi (1999). His parser constructs parse trees with actions similar to those of a standard shift-reduce parser. The derivation of a parse tree is a sequence of actions $a_1, a_2, ..., a_n$ used for constructing a completed parse tree. Therefore all actions that lead to a well-formed parse tree are allowable, and maximum entropy models are trained by examining the derivations of the parse trees in a treebank. The actions of the procedures are scored with maximum entropy probability models that use information in the local context to compute their probabilities. The drawback of this approach is that the model maximizes the probability of the action during each step of the parsing process, instead of overall quality of the parse tree, and thus may suffer from the label bias problem (Lafferty et al., 2001).

Logistic regression models are also prone to overfitting. A common strategy to regularize logistic regression models is to use a Gaussian prior on the parameters, i.e., to maximize

$$\prod_{j=1}^{n} P(y|X) \prod_{i=1}^{F} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp(-\frac{\theta_i^2}{2\sigma_i^2}) \tag{3.4}$$

Goodman (2004) has shown that using a exponential prior is better motivated by the data than previous techniques and often produces lower error rates. With an exponential prior, one maximizes

$$\prod_{j=1}^{n} P(y|X) \prod_{i=1}^{F} \alpha_i \exp(-\alpha_i \theta_i) \tag{3.5}$$

where α_i is a discounting constant and the parameters θ_i are non-negative. This maximization has the advantage of performing implicit feature selection, since many of the parameters θ_i will be set to zero in the final solution.

Charniak (2000) presents a maximum-entropy-inspired parser, which is based on a probabilistic generative model. The generative model is very similar to the model proposed by Collins (1997). Although this maximum-entropy-inspired parser is not quite as flexible as true maximum entropy based models, it is much simpler. The use of the model for conditioning and smoothing, which leads the author successfully to test and combine many different conditioning events, is the major technical innovation of this work.

3.4.2 Support Vector Machines

Support Vector Machines (SVMs) are based on maximum margin strategy introduced by Vapnik (1995) but otherwise can be built on top of exactly the same set of features as a logistic regression (maxent) model. A binary classification problem can be defined in a SVM as follows. Given a set of training data $X = \mathbf{x_1}, ..., \mathbf{x_N}$ and corresponding labels $Y = y_1, ..., y_N$, one would like to find a hyperplane $\boldsymbol{\theta}^\top \mathbf{x} + b = 0$ that correctly separates training examples and has a maximum margin, i.e., maximizes the distance between two hyperplanes. This is achieved by imposing constraints $\boldsymbol{\theta}^\top \mathbf{x} + b \geq 1$ and $\boldsymbol{\theta}^\top \mathbf{x} + b \leq -1$. In fact, the optimal hyperplane with a maximum margin can be obtained by solving the following quadratic program

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top \boldsymbol{\theta} + \mathrm{e}^\top \boldsymbol{\xi} \quad \text{subject to} \\ & y_i \left(\boldsymbol{\theta}^\top \mathbf{x_i} + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{3.6}$$

where β is the regularization parameter and ξ_i are the *slack variables* for the non-separable case.

For parsing, one typically needs a multi-class SVM (Crammer and Singer, 2001) that can handle three or more different class labels (e.g., left link, right link, no link). A multi-class SVM uses features $\mathbf{f}(X, l)$ that depend on both the input example X and a candidate label l. The optimal multi-class classifier is learned by solving the quadratic program

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \mathrm{e}^\top\boldsymbol{\xi} \quad \text{subject to} \\ & \xi_i \geq 1_{(l \neq y_i)} + \boldsymbol{\theta}^\top \mathbf{f}(X_i, l) - \boldsymbol{\theta}^\top \mathbf{f}(X_i, y_i) \\ & \xi_i \geq 0 \\ & \text{for all } i, l \end{aligned} \tag{3.7}$$

Given $\boldsymbol{\theta}$, a test input X is classified to a label according to

$$\hat{y} = \arg\max_{l} \quad \boldsymbol{\theta}^\top \mathbf{f}(X, l) \tag{3.8}$$

As efficient classifiers, SVMs have been widely used in NLP in recent years. For example, as mentioned before, Yamada and Matsumoto (2003) use a multi-class SVM to learn a local scoring function for a dependency parser. Their model uses a discriminative method that maximizes the differences between scores of the link labels in the correct parse versus the scores of the top competing link labels. They obtain satisfactory dependency parsing results on English, although they only train the model parameters locally, not considering optimizing those parameters based on minimizing the global loss of the entire parse tree.

3.5 Global Training Methods

Unfortunately, the simple local learning methods mentioned in Section 3.4 have an obvious shortcoming. The problem is that the training loss being minimized during local parameter optimization has nothing directly to do with the parser. Parsing is a large-scale structured prediction problem where multiple predictions must be coordinated (via dynamic programming) to achieve an accurate parse for a given input sentence. The obvious drawback of using local scoring functions mentioned above is that each link is scored separately, instead of being computed in coordination with other links in a sentence. Although it is true that an accurate local predictor is a prerequisite for an accurate parse prediction, the parameters of the local model are not being trained to directly optimize the global accuracy of the parser. That is, a far better choice of parameters might exist within the given space defined by the features that leads to better global parsing accuracy. This is where the advent of recent training algorithms for learning *structured* predictors has been helpful. The main idea behind these training algorithms has been to explicitly incorporate the effects of the structured predictor (i.e., the parser) *directly* into the training algorithm. That is, parameter optimization of a local predictor is performed by directly considering the implied effects on the structured (global rather than local) prediction error. The challenge is that each component $\hat{y}_i$ of $\hat{Y}$ should not depend only on the input X, but instead should take into account correlations between $\hat{y}_i$ and its neighboring components $\hat{y}_j \in \hat{Y}$.

A significant amount of progress has recently been made on developing training algorithms for learning *structured* predictors from data. The extension to structured training loss has been developed for both the large margin training principle of support vector machines (Tsochantaridis et al., 2004; Altun et al., 2003; Taskar et al., 2003) and the maximum conditional likelihood principle of logistic regression (Lafferty et al., 2001). It has been shown in many application areas that structured prediction models that directly capture the relationships between output components perform better than models that do not directly enforce these relationships (Lafferty et al., 2001; Tsochantaridis et al., 2004; Altun et al., 2003; Taskar et al., 2003; Taskar et al., 2004b). In particular, training algorithms based on these principles have been applied to parsing (Taskar et al., 2004b; Tsochantaridis et al., 2004), and have recently resulted in state-of-the-art accuracy for English dependency parsing (McDonald et al., 2005a; McDonald and Pereira, 2006; Corston-Oliver et al., 2006). I employ this technique in Chapter 6.

Below I give a brief review of the most successful global training methods: Conditional Random Fields and Structured Large Margin Training.

3.5.1 Conditional Random Fields

Conditional Random Fields (CRFs) are closely related to log-linear (logistic regression) models described in Section 3.4.1. Standard log-linear models are trained to make the best local decision, while CRFs are trained to use global normalization for parameter estimation, thus can make the best global decision instead. For example, in a sequence labeling task, a log-linear model (e.g., a maximum entropy Markov model (MEMM), as described in (McCallum et al., 2000)) normalizes the conditional probabilities of next states given the current states with per-state exponential models. A CRF, however, has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence. Therefore, CRFs compute the weights of different features at different states globally and the *label bias problem* shared by locally normalized log-linear models can be avoided (Lafferty et al., 2001; Sha and Pereira, 2003).

A CRF's *global feature vector* for input sequence X and label sequence Y is given by

$$\mathbf{F}(X,Y) = \sum_j \mathbf{f}(X,Y,j) \tag{3.9}$$

where j ranges over input positions. The conditional probability distribution defined by the CRF is then

$$P_\theta(Y|X) = \frac{\exp\left(\boldsymbol{\theta}^\top \mathbf{F}(X,Y)\right)}{\sum_L \exp\left(\boldsymbol{\theta}^\top \mathbf{F}(X,L)\right)} \tag{3.10}$$

where L ranges over all possible Y.

CRFs were first introduced by Lafferty et al. (2001), who applied them to part-of-speech tagging. Since then, CRFs have been widely applied to sequence labeling tasks, such as information extraction (McCallum and Li, 2003) and shallow parsing (Sha and Pereira, 2003). More recently, Jiao et al. (2006) have presented a CRF-based semi-supervised training algorithm for sequence segmentation and labeling tasks, which uses a combination of labeled and unlabeled training data. Their experimental results show that incorporating unlabeled data improves the performance of the supervised CRF. However, CRFs have never previously been applied to dependency parsing, due to the extremely expensive computational cost.

3.5.2 Structured Large Margin Training

Many margin-based discriminative approaches exist and have been applied to parsing, including perceptron (Collins, 2002), support vector machines (Altun et al., 2003), and structured maximum margin methods (Taskar et al., 2004b; McDonald et al., 2005a).

Structured large margin training can be expressed as minimizing a regularized loss (Hastie et al., 2004)

$$\min_{\boldsymbol{\theta}} \quad \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \sum_i \max_{L_{i,k}} \Delta(L_{i,k}, Y_i) - (score(\boldsymbol{\theta}, Y_i) - score(\boldsymbol{\theta}, L_{i,k})) \tag{3.11}$$

where Y_i is the target tree for sentence X_i; $L_{i,k}$ ranges over all possible alternative trees in $\Phi(X_i)$; $score(\boldsymbol{\theta}, Y_i) = \sum_{(x_m \to x_n) \in Y_i} \boldsymbol{\theta}^\top \mathbf{f}(x_m \to x_n)$, as discussed in Section 2.4.1; and $\Delta(L_{i,k}, Y_i)$ is a measure of distance between the two trees $L_{i,k}$ and Y_i. This is an application of the structured large margin training approach first proposed in (Taskar et al., 2003) and (Tsochantaridis et al., 2004).

Using the techniques of (Hastie et al., 2004) one can show that minimizing the objective

(3.11) is equivalent to solving the quadratic program

$$\begin{aligned}\min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \mathbf{e}^\top\boldsymbol{\xi} \quad \text{subject to} \\ & \xi_{i,k} \geq \Delta(L_{i,k}, Y_i) + score(\boldsymbol{\theta}, L_{i,k}) - score(\boldsymbol{\theta}, Y_i) \\ & \xi_{i,k} \geq 0 \\ & \text{for all } i, L_{i,k} \in \Phi(X_i) \end{aligned} \tag{3.12}$$

which corresponds to the training problem posed in (McDonald et al., 2005a). This approach has yielded the best published results for English dependency parsing (McDonald et al., 2005a). The main drawback of this approach are the exponential number of constraints in (3.12). McDonald et al. bypass this problem with an on-line training algorithm. I will improve the standard supervised large margin training technique and invent a novel semi-supervised convex training algorithm for dependency parsing in later chapters.

3.6 Other Relevant Machine Learning Techniques

In this section, I discuss a few other machine learning techniques that I have exploited in my research.

3.6.1 Boosting

Boosting is a general machine learning method for improving the accuracy of a learning algorithm. The idea of boosting is to combine many moderately accurate "rules of thumb" (or weak classifiers) in a principled manner to produce a single highly accurate classifier (Schapire, 2001). Due to their sound theoretical foundations, a lot of work related to boosting algorithms has been conducted (Freund and Schapire, 1996; Freund and Schapire, 1997; Schapire and Singer, 1999; Collins et al., 2002).

The AdaBoost algorithm (Freund and Schapire, 1997), which solved many of the practical difficulties of the earlier boosting algorithms, is the most popular. A simplified description of AdaBoost is shown in Figure 3.6.1.

Given an input training set $(x_1, y_1), ..., (x_n, y_n)$, AdaBoost calls a given *weak learner* (also called *weak hypothesis* or *base learning algorithm*) in a series of rounds $t = 1, 2, ..., T$. The algorithm updates the set of weights over the training examples at each boosting round. The idea behind AdaBoost is to start with a uniform weighting over the training examples, and progressively adjust the weights to emphasize the examples that have been frequently misclassified by the weak hypotheses. Thus the weak learner is forced to concentrate on

```
Given: $(x_1, y_1), ..., (x_n, y_n)$ where $x_i \in X$, $y_i \in Y$ = {-1,+1}
Initialize $D_1(i) = 1/n$.
For $t = 1, ..., T$ :
    • Train weak learner using distribution $D_t$.
    • Get weak hypothesis $h_t : X \longrightarrow \{-1, +1\}$ with error
          $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
    • Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
    • Update:
          $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
      where $Z_t$ is a normalization factor (choose so that $D_{t+1}$ will be a distribution).
Output the final hypothesis:
    $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

Figure 3.1: The AdaBoost boosting algorithm

the hard examples in the training set. The final hypothesis is a weighted majority vote of the T weak hypotheses. The most essential theoretical property of AdaBoost has to do with its ability to reduce the training error. It has been proved that if each weak hypothesis is slightly better than random guess, then the training error drops exponentially fast.

The advantages of AdaBoost are its simplicity and generality. First, it is simple and easy to implement. Second, it has no parameters to tune, while most other machine learning methods possess multiple parameters. Furthermore, it can be applied to any standard training method without requiring any prior knowledge about the weak learner. More importantly, it provides a set of theoretical guarantees given sufficient training data and a weak learner which can produce only moderately accurate weak hypotheses. Due to its many practical advantages, boosting has been applied to many tasks, such as text categorization (Schapire and Singer, 2000), tagging and prepositional phrase attachment (Abney et al., 1999) and parsing (Haruno et al., 1999). I will consider this technique in my own work on parsing in Chapter 7 below.

3.6.2 Similarity Smoothing

The sparse data problem is very common in NLP. For learning a parser from data, the feature set introduced in Section 2.4.2 is far too large to yield uniformly reliable estimates. Abstraction (e.g. using parts-of-speech) and smoothing are two standard techniques for mitigating the sparse data problem. Note that most features discussed in Section 2.4.2 correspond to words or pairs of words. The weights on these features can be smoothed based on *similarity* determined on an auxiliary, unannotated corpus.

Words that tend to appear in the same contexts tend to have similar meanings. This is known as the Distributional Hypothesis in linguistics (Harris, 1968). For example, the words *test* and *exam* are similar because both of them follow verbs such as *administer, cancel, cheat on, conduct,* etc. and both of them can be preceded by adjectives such as *academic, comprehensive, diagnostic, difficult,* etc.

Many methods have been proposed to compute distributional similarity between words (Hindle, 1990; Pereira et al., 1993; Grefenstette, 1994; Lin, 1998). Almost all of those methods represent a word by a feature vector where each feature corresponds to a type of context in which the word appears. They differ in how the feature vectors are constructed and how the similarity between two feature vectors is computed.

centrepiece 0.28, figment 0.27, fulcrum 0.21, culmination 0.20, albatross 0.19, bane 0.19, pariahs 0.18, lifeblood 0.18, crux 0.18, redoubling 0.17, apotheosis 0.17, cornerstones 0.17, perpetuation 0.16, forerunners 0.16, shirking 0.16, cornerstone 0.16, birthright 0.15, hallmark 0.15, centerpiece 0.15, evidenced 0.15, germane 0.15, gist 0.14, reassessing 0.14, engrossed 0.14, Thorn 0.14, biding 0.14, narrowness 0.14, linchpin 0.14, enamored 0.14, formalised 0.14, tenths 0.13, testament 0.13, certainties 0.13, forerunner 0.13, re-evaluating 0.13, antithetical 0.12, extinct 0.12, rarest 0.12, imperiled 0.12, remiss 0.12, hindrance 0.12, detriment 0.12, prouder 0.12, upshot 0.12, cosponsor 0.12, hiccups 0.12, premised 0.12, perversion 0.12, destabilisation 0.12, prefaced 0.11, ...

Figure 3.2: Similar words of *keystone*

Similarity Measures

The most popular method is to use *point-wise mutual information* (PMI) to compute word similarity (Manning and Schutze, 1999). In this method, each word is presented as a feature vector f of contexts. The contexts of a word w are defined to be the set of words that occur within a small context window of w in a large corpus. The contexts of an instance of w consist of the closest *non-stop-words* on each side of w and the *stop-words* in between. Usually, the set of stop-words are defined as the top k most frequent words in the corpus. The value of a feature c is then defined as the point-wise mutual information between c and w:

$$f_w(c) = PMI\,(w, c) = \log\left(\frac{P\,(w, c)}{P\,(w)\,P\,(c)}\right) \tag{3.13}$$

where $P(w,\, c)$ is the probability of w and c co-occur in a context window.

Once the feature vectors have been determined, the *similarity* between two words w_1 and w_2 is then computed as the cosine of the corresponding feature vectors:

$$Sim(w_1, w_2) \quad = \quad \frac{\mathbf{f}_{w_1} \cdot \mathbf{f}_{w_2}}{||\mathbf{f}_{w_1}||\,||\mathbf{f}_{w_2}||} \tag{3.14}$$

For example, Figure 3.6.2 shows the top similar words and corresponding similarities for the word *keystone*. They are computed from the English Gigaword corpus (Graff, 2003), which is a raw, unannotated newswire text data containing about one giga English words.

Similarity-based Smoothing

Similarity-based smoothing is used in (Dagan et al., 1999) to estimate word co-occurrence probabilities. Their method performs almost 40% better than the more commonly used back-off method. Unfortunately, similarity-based smoothing has not been successfully applied to statistical parsing up to now. However, in my own work in Chapter 5 and Chapter 6 below, I show how similarity-based smoothing can be used to improve the accuracy of generative and large margin based learning approaches for parsing respectively.

In the original application of similarity-based smoothing (Dagan et al., 1999), bigram probabilities P($w_2|w_1$) were computed as the weighted average of the conditional probability of w_2 given similar words of w_1:

$$P_{SIM}\left(w_2|w_1\right) = \sum_{w_1' \in S(w_1)} \frac{Sim\left(w_1, w_1'\right)}{norm\left(w_1\right)} P_{MLE}\left(w_2|w_1'\right) \tag{3.15}$$

where $Sim(w_1, w_1')$ denotes the similarity (or an increasing function of the similarity) between w_1 and w_1', and $S(w_1)$ denotes the set of words that are most similar to w_1. The normalization factor *norm*(w_1) is computed as

$$norm\left(w_1\right) = \sum_{w_1' \in S(w_1)} Sim\left(w_1, w_1'\right)$$

The underlying assumption of this smoothing scheme is that a word is more likely to occur after w_1 if it tends to occur after similar words of w_1.

Similarity-based smoothing has turned out to be an important smoothing approach in many areas of natural language processing, which allows one to tap into unlimited auxiliary sources of raw unannotated text. By using similarity-based smoothing, one can easily estimate parameters for words that have never appeared in the training corpus. One of the goals of my work has been to obtain similar advantages on parsing.

3.6.3 Unsupervised/Semi-supervised Learning

Unsupervised learning is clearly important in statistical natural language research as it eliminates the need for extensive manual annotation. However, in general, and parsing in particular, unsupervised learning is significantly harder than supervised learning, especially on language data where so many linguistic variables remain latent. The dangers of poor estimation are exaggerated and even self-reinforcing in this case, since unsupervised learners typically attempt to bootstrap from inferred values of hidden variables, which themselves are based on earlier, weaker estimates.

Learning a parser from treebanks has its advantages—it is an example of supervised learning, which is well studied—but it has the limitation of domain dependence. A parser trained using state-of-the-art techniques and models from a given treebank tends to perform very well within that treebank's domain, but it usually tends to perform very poorly on the data from other domains. Another limitation is that there are only a few languages available in treebanks. Except for the English treebank, most other treebanks are significantly smaller. Building a treebank is very expensive, usually taking a few human-years to construct, at the minimum.

Perceiving these limitations of supervised parsing, many researchers have investigated *unsupervised parsing* techniques. To date, however, the performance of unsupervised parsers are far inferior to supervised parsers. It has been more than a decade since researchers began working on unsupervised parsing (Carroll and Charniak, 1992; Pereira and Schabes, 1992). Unsupervised parsing requires only raw, unannotated text, which is unlimited and practically free (although using a small set of annotated examples is advantageous in most unsupervised cases). Hence the domain and language dependency issues vanish. The ideal goal of unsupervised parsing research would be the ability to automatically synthesize useful parsing models for any given application, with a minimum of human effort.

Earlier work on unsupervised grammar induction is mainly based on information-theoretic criteria (Magerman and Marcus, 1990), such as linguistic knowledge and lexical relationships between words. For example, Yuret (1998) introduces probabilistic *lexical attraction* models which can represent long distance relations between words. Within the framework of lexical attraction, he developed an unsupervised language acquisition program that learns to identify linguistic relations in a given sentence.

Recent successful work in grammar induction has used generative probabilistic models and unsupervised parameter estimation techniques, specifically, EM-like algorithms (Klein and Manning, 2002; Klein and Manning, 2004). For example, Klein and Manning (2004) have presented a dependency-based model for the unsupervised induction of syntactic structure. They then combine this model with another constituent-induction model. The combined model substantially outperforms either individual model. A crucial reason that their models are capable of recovering syntactic structure more accurately than previous ones is that they minimize the amount of hidden structure that must be induced.

The drawbacks of using EM algorithm for unsupervised learning are the issues of local optima and the disconnect between likelihood and accuracy. EM tries to fit the parameters of a statistical model with hidden structure to the training data, which is not enough to recover useful syntactic structure. Subsequently, alternative estimation strategies for unsupervised learning have been proposed, such as my own work on POS tagging by using better smoothing techniques or added constraints (Wang and Schuurmans, 2005), or *Contrastive Estimation* (CE) by Smith and Eisner (2005a). Contrastive Estimation is a generalization of

EM, by defining a notion of learner guidance. It makes use of a set of examples (its *neighborhood*) that are similar in some way to an observed example, requiring the learner to move probability mass to a given example, taking only from the example's neighborhood. The neighborhood selection is crucial, and Smith and Eisner (2005b) choose a neighborhood that explicitly represents potential mistakes of the model. The model is then trained to avoid these mistakes.

Recently, another new unsupervised learning algorithm of training structured predictors, which is discriminative, convex, and avoids the use of EM has been presented by Xu et al. (2006). Their experimental results for training hidden Markov models (HMMs) without supervision show that the convex discriminative procedure can produce better conditional models than conventional EM training. The current technique is computationally very expensive, however, requiring one to solve very large semi-definite programs. More recently, as discussed in Section 3.5.1, Jiao et al. (2006) have presented a CRFs-based semi-supervised training algorithm for sequence segmentation and labeling tasks. Their approach is based on extending the minimum entropy regularization framework to the structured prediction case, yielding a training objective that combines unlabeled conditional entropy with labeled conditional likelihood. Their experimental results show that incorporating unlabeled data improves the performance of the supervised CRF. None of these alternative unsupervised training procedures, however, have yet been applied to parsing. This is a direction for future research that I would like to pursue, as discussed in Chapter 9.2 below.

Chapter 4

Experimental Setup

In previous chapters, I have introduced some background knowledge on syntactic parsing, and how one can use machine learning methods to tackle this problem. I have discussed some most commonly used learning methods for parsing and shown the connection between these alternative approaches. I have also introduced other relevant machine learning techniques for parsing. In this chapter, I describe the data sets and the evaluation metrics I will use for the experiments to be reported in later chapters.

4.1 Data Sets

The treebanks discussed in Section 2.5 are the primary data source for researchers to evaluate their parsers. There are treebanks for Arabic, Chinese, Czech, English and Korean available from the Linguistic Data Consortium (LDC). These treebanks differ in data size and the tree annotations (dependency trees or constituency trees).

4.1.1 English

For experiments on English, the treebank used is the Wall Street Journal (WSJ) section of the English Penn Treebank (PTB) (Marcus et al., 1993). PTB contains a collection of 43,600 manually parsed English sentences (over one million words) with constituency trees. For the PTB, researchers have used a standard training, development and test split, i.e., section 02-21 for training, section 22 for development and section 23 for test. The constituency structures were converted to dependency trees based on the same rules as (Yamada and Matsumoto, 2003) mentioned in Section 2.2.1. When POS tags are needed, I use the gold standard tags for the training set. The development and test sets are tagged with the part-of-speech tagger of (Ratnaparkhi, 1996).

Precision	$\frac{A}{B}$
Recall	$\frac{A}{C}$
Crossing Brackets	$\frac{D}{C}$

Table 4.1: PARSEVAL measures

A = number of correct constituents in a proposed parse
B = number of constituents in a proposed parse
C = number of constituents in the treebank parse
D = number of constituents violating constituent boundaries in a proposed parse

4.1.2 Chinese

For experiments on Chinese, I used both the Penn Chinese treebank 4.0 (CTB4) (Palmer *et al.*, 2004) and the Penn Chinese treebank 5.0 (CTB5) (Palmer *et al.*, 2005). Chinese Treebank 4.0 contains 15,162 sentences (404,156 words). Chinese Treebank 5.0 contains Chinese Treebank 4.0 as a subset, but adds approximately 3000 sentences (100,000 words) of Taiwanese Chinese text. CTB4 and CTB5 also contain constituency trees for each training sentence. The conversion from constituency structures to dependency trees is based on the rules described in (Bikel, 2004). For CTB4, researchers have been using the data split of (Bikel, 2004): Sections 1-270 and 400-931 for training, Sections 301-325 for development and Sections 271-300 for test. However, for the new released Chinese treebank 5.0, researchers have not yet agreed on a data split. One option is to use the split of (Corston-Oliver et al., 2006), i.e., a 70% / 15% / 15% split for training / development / test by sampling the whole treebank. Following other researchers, I use the gold standard tags for all the data sets.

4.2 Evaluation Measures

An important question is how to evaluate the success of a parser. One evaluation method is called *task-based evaluation*, where researchers use parsers to improve the performance of other systems, such as information extraction, question answering and machine translation systems. Thus, parsers can be evaluated by embedding them into such a system and to analyze the differences that the various parsers make.

However, for simplicity, modularization, and reasons of convenience, an evaluation method that directly measures the performance of a parser is more desirable. In this case, one would first think of comparing a parse tree with a gold standard tree taken from a

Dependency Accuracy	percentage of words that have the correct head
Root Accuracy	percentage of sentences that have the correct root
Complete Match	percentage of sentences where the entire dependency tree is correct

Table 4.2: Dependency tree accuracy measures

treebank—a collection of manually parsed sentences. The most stringent measure is to investigate if the two trees are exactly the same. This measure is called *tree accuracy* or *complete match*, which completely ignores those parses which have any error. But for some purposes, partially correct parses would still be very useful, and the complete match score is not detailed enough to truly assess parsing accuracy. Therefore, less stringent evaluation measures are usually used for both constituency trees and dependency trees.

The PARSEVAL measures shown in Table 4.1 are used to evaluate the component pieces of constituency trees (but not dependency trees). These measures have been commonly used for parser evaluation for more than a decade (Black et al., 1991), originally used to compare the performance of non-statistical parsers.

For dependency trees, due to the constraint that each word has only one head word, the number of total links in a dependency tree is the same as the number of words in the sentence. Thus, the three accuracy measures shown in Table 4.2 are usually used instead to assess the accuracy of a dependency parser: dependency accuracy (DA), root accuracy (RA) and complete match (CM). Of the three, dependency accuracy is often considered to be the most important measure, since lexical relations between word-pairs are embedded in dependency links, while the other two measures, root accuracy and complete match, are not detailed enough for the evaluation of a parser's performance.

I can also evaluate a dependency parser with either directed dependency accuracy (where dependency links are directed) or undirected dependency accuracy (where the direction of links are ignored). The downside of the undirected accuracy measure is that, it is not as informative as the directed accuracy measure, since a syntactic relationship between two words is usually asymmetric. However, the undirected measure is still desirable when one is only interested in the likelihood of the linkage between two words. Moreover, the number of features can be largely reduced by using undirected features. In effect, an undirected dependency tree can be easily converted into a directed one by specifying the head of the sentence, since the head induces a unique head-outward ordering over all other dependencies.

In general, a parser's performance is evaluated based on these pre-defined data splits discussed in Section 4.1. Typically, the procedure is as follows:

- Train the parser on training set;
- Use development set to set parameters;
- Use accuracy measures discussed above to evaluate the parser on test set to get unbiased estimate of parsing accuracy.

For the four pieces of my own work I will present in Chapter 5, Chapter 6, Chapter 7, and Chapter 8 below, I adopt these data splits as discussed in Section 4.1 to evaluate my parsers and to compare my parsing results on both Chinese and English to the state-of-the-art results.

4.3 Complexity Issues

Dependency parsing is a complicated task that involves dealing with a huge set of features and a large search space with exponential number of possible parse trees. In particular, for discriminative training, one needs to parse training corpora repeatedly, which is usually expensive.

In practice, many strategies can be used to handle these issues. First, one can easily solve the huge feature set problem by introducing a cut-off number, i.e., those features that occur less than a cut-off number are ignored. Second, using a fast dependency parsing algorithm is an effective way to speed up the training phase. Finally, one can partition the training data according to some criteria and train the separate partitions in parallel.

Chapter 5

Strictly Lexicalised Dependency Parsing

In this chapter, I present a strictly lexical parsing model where all the parameters are based on words. This model does not rely on part-of-speech tags or grammatical categories. It maximizes the conditional probability of a parse tree given a sentence, as discussed in Section 3.4.1. This is in contrast with most previous generative models that compute the joint probability of the parse tree and the sentence. As I discussed in Section 3.4.1, this conditional model is trained to minimize a loss function related to labeling error and is able to deal with inter-dependent features of input sentences. It also allows one to use distributional word similarity to generalize the observed frequency counts in the training corpus (as discussed in Section 3.6.2). The experimental results on the Chinese Treebank 4.0 show that the accuracy of the conditional model is 13.6% higher than the joint model and that the strictly lexicalised conditional model outperforms the corresponding unlexicalized model based on part-of-speech tags. This work was published in Wang et al. (2005).

5.1 Lexicalised Parsing

A common characteristic of previous generative parsers (Collins, 1996; Collins, 1997; Charniak, 2000) is their use of lexicalised statistics. However, it was subsequently discovered that bi-lexical statistics (parameters that involve two words) actually play a much smaller role than previously believed. It has been found by Gildea (2001) that the removal of bi-lexical statistics from a state-of-the-art PCFG parser resulted in little change in the output. Bikel (2004) observes that only 1.49% of the bi-lexical statistics needed in parsing were found in the training corpus. When considering only bigram statistics involved in the highest probability parse, this percentage becomes 28.8%. However, even when bi-lexical statistics do get used, they are remarkably similar to their back-off values using part-of-speech tags. Therefore, the utility of bi-lexical statistics becomes rather questionable. Klein and Manning (2003) present an unlexicalized parser that eliminates all lexicalised parameters, with a performance score close to the state-of-the-art lexicalised parsers.

I present a statistical dependency parser that represents the other end of spectrum where all statistical parameters are lexical and the parser does not require part-of-speech tags or grammatical categories. This is called *strictly lexicalised parsing*. A part-of-speech lexicon has always been considered to be a necessary component in any natural language parser, as mentioned in Section 2.4.2. This is true in early rule-based as well as modern statistical parsers and in dependency parsers as well as constituency parsers. The need for part-of-speech tags arises from the sparseness of natural language data. They provide generalizations of words that are critical for parsers to deal with the sparseness. Words belonging to the same part-of-speech are expected to have the same syntactic behavior.

Instead of relying on part-of-speech tags, I use *distributional word similarities* computed automatically from a large unannotated corpus as described in Section 3.6.2. One of the benefits of strictly lexicalised parsing is that the parser can be trained with a treebank that only contains dependency relationships between words. The annotators do not need to annotate parts-of-speech or non-terminal symbols (they do not even have to know about them), making the construction of treebanks easier. Strictly lexicalised parsing is especially beneficial for languages such as Chinese, where parts-of-speech are not as clearly defined as English. In Chinese, clear indicators of a word's part-of-speech, such as suffixes *-ment, -ous* or function words, such as *the*, are largely absent. In fact, monolingual Chinese dictionaries intended for native speakers almost never contain part-of-speech information.

In the remainder of this chapter, I first present a method for modeling the probabilities of dependency trees. Next, in Section 5.3, I apply a similarity-based smoothing technique to the probability model to deal with data sparseness. Then I describe a dependency parsing algorithm I use for experimental evaluation in Section 5.4. Finally, I present dependency parsing results on the Chinese Treebank 4.0 in Section 5.5 and discuss related work in Section 5.6.

5.2 A Probabilistic Dependency Parsing Model

Let X be a sentence and Y be its dependency tree (shown in Figure 2.1). As discussed in Section 2.2.1, Y is a directed tree connecting all the words in X.

A triple (u, v, d) specifies a dependency link l, where u and v are the indices $(u < v)$ of the words connected by l, and d specifies the direction of the link l. The value of d is either L or R. If $d = L$, v is the index of the head word; otherwise, u is the index of the head word.

As discussed in Section 2.2.1, dependency trees are typically assumed to be projective (without crossing arcs), which means that if there is an arc from h to m, h is then an ancestor of all the words between h and m. Let $\Phi(X)$ be the set of possible directed, projective trees spanning on X. The parsing problem defined in Equation (2.1) in Section 2.4.1 is to maximize the sum of all the link scores in a candidate tree. Here, the score will be log conditional probabilities. Thus finding the tree with highest probability would be equivalent to finding the tree with a maximum score in Equation (2.1).

Generative parsing models are usually defined recursively from top down, even though the decoders (parsers) for such models almost always take a bottom-up approach. The model proposed here is a bottom-up one. Like previous approaches, the generation of a parse tree can be decomposed into a sequence of steps. The probability of the tree is simply the product of the probabilities of the steps involved in the generation process. This scheme requires that different sequences of the steps must not lead to the same tree. This can be achieved by defining a canonical ordering of the links in a dependency tree. Each generation step corresponds to the construction of a dependency link in the canonical order.

Given two dependency links l and l' with the heads being h and h' and the modifiers being m and m', respectively, the order between l and l' is determined as follows:

- If $h \neq h'$ and there is a directed path from one (say h) to the other (say h'), then l' precedes l.
- If $h \neq h'$ and there does not exist a directed path between h and h', the order between l and l' is determined by the order of h and h' in the sentence (h precedes h' => l precedes l').
- If $h = h'$ and the modifiers m and m' are on different sides of h, the link with modifier on the right precedes the other.
- If $h = h'$ and the modifiers m and m' are on the same side of the head h, the link with its modifier closer to h precedes the other one.

For example, if we add indices 1, 2, 3... to the words in Figure 2.1 (index 0 is for the dummy node at the beginning of the sentence), the canonical order of the links in the dependency tree is: (4, 5, R), (7, 8, L), (6, 8, R), (4, 6, R), (3, 4, R), (2, 3, R), (1, 2, L), (0, 2, R).

The generation process according to the canonical order is similar to the head outward generation process in (Collins, 1999), except that it is bottom-up whereas Collins' models are top-down. Suppose a dependency tree Y is constructed in steps $G_1, \ldots, G_N$ in the canonical order of the dependency links, where N is the number of words in the sentence. The conditional probability of Y given X can be computed as

$$P(Y|X) = P(G_1, G_2, ..., G_N|X) = \prod_{i=1}^{N} P(G_i|X, G_1, ..., G_{i-1}) \tag{5.1}$$

To search for a parse tree with the highest probability is equivalent to find out a tree with the highest score of sum over all the dependency links in the logarithmic space.

$$\begin{aligned} Y^* &= \arg \max_{Y \in \Phi(X)} P(Y|X) \\ &= \arg \max_{Y \in \Phi(X)} log(P(Y|X)) \\ &= \arg \max_{Y \in \Phi(X)} \sum_{i=1}^{N} log(P(G_i|X, G_1, ..., G_{i-1})) \end{aligned} \tag{5.2}$$

Following (Klein and Manning, 2004), I require that the creation of a dependency link from head h to modifier m be preceded by placing a left STOP and a right STOP around the modifier m and ¬STOP between h and m. The STOP events are crucial for modeling the number of dependents. Without them, a parse tree often contains some 'obvious' errors, such as determiners taking arguments, or prepositions having arguments on their left (instead of right).

Let E_w^L (and E_w^R) denote the event that there are no more modifiers on the left (and right) of a word w. Suppose the dependency link created in the step i is (u, v, d). If $d = L$, G$_i$ is the conjunction of the four events: E_u^R, E_u^L, $\neg E_v^L$and *link*$_L(u, v)$. If $d = R$, G$_i$ consists of four events: E_v^R, E_v^L, $\neg E_u^R$ and *link*$_R(u, v)$. The event G_i is conditioned on X, $G_1, \ldots, G_{i-1}$, which are the words in the sentence and a forest of trees constructed up to step i-1. Let C_w^L (and C_w^R) be the number of modifiers of w on its left (and right). I make the following independence assumptions:

- Whether there are any more modifiers of w on side d depends only on the number of modifiers already found on side d of w. That is, E_w^d depends only on w and C_w^d.
- Whether there is a dependency link from a word h to another word m depends only on the words h and m and the number of modifiers of h between m and h. That is,
  - *link*$_R(u,v)$ depends only on u, v, and C_u^R.

  - *link*$_L(u,v)$ depends only on u, v, and C_v^L.

Suppose G_i corresponds to a dependency link (u, v, L). The probability can be computed as:

$$\begin{aligned} &P\left(G_i|S,G_1,...,G_{i-1}\right) \\ &= P\left(E_u^L, E_u^R, \neg E_v^L, link_L\left(u,v\right)|S,G_1,...,G_{i-1}\right) \\ &= P\left(E_u^L|u,C_u^L\right) \times P\left(E_u^R|u,C_u^R\right) \times \\ &\quad \left(1 - P\left(E_v^L|v,C_v^L\right)\right) \times P\left(link_L\left(u,v\right)|u,v,C_v^L\right) \end{aligned}$$

The events E_w^R and E_w^L correspond to the STOP events in (Collins, 1999; Klein and Manning, 2004). This model requires three types of parameters:

- $P\left(E_w^d|w,C_w^d\right)$, where w is a word, d is a direction (left or right). This is the probability of a STOP after taking C_w^d modifiers on the d side.
- $P\left(link_R\left(u,v\right)|u,v,C_u^R\right)$ is the probability of v being the $(C_u^R + 1)$'th modifier of u on the right.
- $P\left(link_L\left(u,v\right)|u,v,C_v^L\right)$ is the probability of u being the $(C_v^L + 1)$'th modifier of v on the left.

The maximum likelihood estimations of these parameters can be obtained from the frequency counts in the training corpus:

- $C(w,c,d)$: the frequency count of w with c modifiers on the d side.
- $C(u,v,c,d)$: If $d = L$, this is the frequency count of words u and v co-occurring in a sentence and v has c modifiers between itself and u. If $d = R$, this is the frequency count words u and v co-occurring in a sentence and u has c modifiers between itself and v.
- $K(u,v,c,d)$: similar to $C(u,v,c,d)$ with an additional constraint that *link*$_d(u,v)$ is true.

$$P\left(E_w^d|w,C_w^d\right) = \frac{C\left(w,c,d\right)}{\sum_{c' \geq c} C\left(w,c',d\right)}, \quad where \quad c = C_w^d;$$

$$P\left(link_R\left(u,v\right)|u,v,C_u^R\right) = \frac{K\left(u,v,c,R\right)}{C\left(u,v,c,R\right)}, \quad where \quad c = C_u^R;$$

$$P\left(link_L\left(u,v\right)|u,v,C_v^L\right) = \frac{K\left(u,v,c,L\right)}{C\left(u,v,c,L\right)}, \quad where \quad c = C_v^L.$$

All the parameters in the model are conditional probabilities of the tree given the sentence, where the variables on the left side of the conditioning bar are binary. Taking logs of these probabilities one can then obtain a local scoring function that uses dynamic features as discussed in Section 2.4.3. This scoring function still decomposes in a way that allows one to use a dynamic programming parsing algorithm as described in Section 2.3 to parse sentences. The algorithm builds a packed parse forest from bottom up according to the canonical order introduced above.

5.3 Similarity-based Smoothing

I now introduce similarity-based smoothing into the dependency parsing framework outlined above, which to the best of my knowledge, is novel.

The parameters in the model consist of conditional probabilities P($E|C$) where E is the binary variable *link*$_d(u, v)$ or E^d_w and the context C is either $[w, C^d_w]$ or $[u, v, C^d_w]$, which involves one or two words in the input sentence. Due to the sparseness of natural language data, the contexts observed in the training data only cover a tiny fraction of the contexts whose probability distributions are needed during parsing. The standard approach is to back off the probability to word classes (such as part-of-speech tags). In this chapter, I take a different approach: the training data is searched to find a set of similar contexts to C, and the probability of E is estimated based on its probabilities in the similar contexts observed in the training corpus.

Section 3.6.2 introduced the smoothing method of (Dagan et al., 1999). The underlying assumption of their smoothing scheme is that a word is more likely to occur after w if it tends to occur after similar words of w. Here I make a similar assumption: the probability P($E|C$) of event E given the context C is computed as the weighted average of P($E|C'$) where C' is a similar context of C and is attested in the training corpus:

$$P_{SIM}\left(E|C\right) = \sum_{C' \in S(C) \cap O} \frac{Sim\left(C, C'\right)}{norm\left(C\right)} P_{MLE}\left(E|C'\right)$$

where S(C) is the set of top k most similar contexts of C (in the experiments reported in this chapter, k = 50); O is the set of contexts observed in the training corpus, *Sim*(C, C') is the similarity between two contexts and *norm*(C) is the normalization factor.

Here, a context is either $[w, C^d_w]$ or $[u, v, C^d_w]$ and their similar contexts are defined as:

$$S\left(\left[w, C^d_w\right]\right) = \left\{ \left[w', C^d_{w'}\right] \middle| w' \in S\left(w\right)\right\}$$
$$S\left(\left[u, v, C^d_w\right]\right) = \left\{ \left[u', v', C^d_w\right] \middle| u' \in S(u), v' \in S(v)\right\}$$

where $S(w)$ is the set of top k similar words of w (k = 50).

Since all contexts used in the model contain at least one word, the similarity between two contexts, Sim(C, C'), is computed as the geometric average of the similarities between corresponding words:

$$Sim\left(\left[w, C_w^d\right], \left[w', C_{w'}^d\right]\right) = Sim\left(w, w'\right)$$
$$Sim\left(\left[u, v, C_w^d\right], \left[u', v', C_{w'}^d\right]\right) = \sqrt{Sim\left(u, u'\right) Sim\left(v, v'\right)}$$

Note that using a similarity-smoothed probability estimate is only necessary when the frequency count of the context C in the training corpus is low. Therefore the final probability is computed as the linear interpolation of the MLE probability and the similarity-based probability.

$$P\left(E|C\right) = \alpha P_{MLE}(E|C) + (1-\alpha)P_{SIM}(E|C) \tag{5.3}$$

where the smoothing factor $\alpha = \frac{|C|+1}{|C|+5}$ and $|C|$ is the frequency count of the context C in the training data. The purpose of α is to dynamically scale the smoothing, based on the frequency of the pair.

A difference between the similarity-based smoothing approach of (Dagan et al., 1999) and the approach proposed here is that this model only computes probability distributions of binary variables. Words only appear as parts of contexts on the right side of the conditioning bar. This has two important implications. First, when a context contains two words, one can use the cross product of similar words, whereas Dagan et al. (1999) can only use the similar words of one of the words. This turns out to have significant impact on accuracy (see Section 5.5). Second, in (Dagan et al., 1999), the distribution P($.|w_1'$) may itself be sparsely observed. When $P_{MLE}(w_2|w_1')$ is 0, it is often due to data sparseness. Their smoothing scheme therefore tends to under-estimate such probability values. This problem is avoided with the approach presented here. If a context does not occur in the training data, it is not included in Equation 5.3. If it does occur, the maximum likelihood estimation is reasonably accurate even if the context only occurs a few times, since the entropy of the probability distribution is upper-bounded by log 2.

5.4 Dependency Parsing Algorithms

Before presenting experimental results, I first describe the dependency parsing algorithm used in the experimental evaluation, which is adapted from a standard CKY parsing algorithm (Jurafsky and Martin, 2000). This has been discussed in Section 2.3.2.

Although the output of the parser is a dependency tree, internally, it works as similarly as a chart parsing algorithm for Context Free Grammars. Specifically, in a dependency

```
Parse() {
  for (h = 0; h < N; ++h) {
    AddItem(new Item(h, h, h, 0));
    for l from h down to 0 do {
      foreach item t in items(l, h) {
        MergeAsModifier(t);
        MergeAsHead(t);
      }
    }
  }
}

MergeAsHead(item) {
  h = item.high;  mid = item.low - 1;
  for l from mid down to 0 do {
    m = argmax_{t in items(l, mid)} combined_score(h, t)
    AddItem(new Item(m.low, item.head, item.high, Combined_score(h, m)));
  }
}

MergeAsModifier(item) {
  h = item.high;  mid = item.low - 1;
  for l from mid down to 0 do {
    foreach item m in items(l, mid) without a pre-head modifier
      AddItem(new Item(m.low, m.head, item.high, Combined_score(m, item);
  }
}

AddItem(item) {
  if not exist t in item(l, h) s.t. t.head==item.head and
     t.score > item.score
  then add item to items(item.low, item.high);
}
```

Figure 5.1: A dependency parsing algorithm

parsing algorithm, the parser constructs a set of chart items, each of which has a head word. Each chart item is a 4-tuple: (*low, head, high, score*) where *low*, *head* and *high* (*low* $\leq$ *head* $\leq$ *high*) are positions of words in a sentence and *score* is non-negative. This means that there exists a dependency tree that spans the words from *low* to *high* with the given *score*, and rooted at the position *head*. Initially, the parsing algorithm creates a chart item for each individual word in the input sentence. The items are then combined with the existing items that are adjacent items to their left. The combined item has the span of the union of the two components and may take either item's head as its head. Thus, a dependency tree for the whole sentence can then be built up in a bottom-up manner, by successively combining adjacent chart items into bigger ones. A dependency parsing algorithm implemented in this way has $O(n^5)$ complexity, in the worst case, as I have mentioned in Section 2.3.2. An algorithm outline is given in Figure 5.1.

This dependency parsing algorithm is essentially a modified CKY parsing algorithm. I use this algorithm in the experimental evaluation in this chapter and in all the following chapters.

Test data	CTB4-10	CTB4-15	CTB4-20	CTB4-40
Undirected Accuracy (%)	90.8	85.6	84.0	79.9

Table 5.1: Evaluation results on CTB4

Models	**Accuracy (%)**
(a) Strictly lexicalised conditional model	**79.9**
(b) At most one word is different in a similar context	77.7
(c) Strictly lexicalised joint model	66.3
(d) Unlexicalized conditional models	71.1
(e) Unlexicalized joint models	71.1

Table 5.2: Performance of alternative models

5.5 Experimental Results

I evaluated the proposed learning technique developed in this chapter on the Penn Chinese Treebank 4.0 (CTB4) as described in Section 4.1.2. I used the same experimental settings as discussed in Section 4.1.2 for CTB4. I tested on the sets of data with different sentence length: CTB4-10, CTB4-15, CTB4-20 and CTB4-40, which contain test sentences with up to 10, 15, 20 and 40 words respectively. Parsing Chinese generally involves segmentation as a pre-processing step. I used the gold standard segmentation in the CTB4. The distributional similarities between words are computed using the Chinese Gigaword corpus (Graff and Chen, 2003). I did not segment the corpus when computing the word similarities.

I measured the quality of the parser by undirected accuracy as discussed in Section 4.2, which is defined as the number of correct undirected dependency links divided by the total number of dependency links in the corpus (the treebank parse and the parser output always have the same number of links). The results are summarized in Table 5.1. These results show the performance of the parser is highly correlated with the length of sentences, due to the fact that the number of possible parse trees increases exponentially with sentence length.

I also experimented with several alternative models. Table 5.2 summarizes the results of these models on the test corpus with sentences with less than or equal to 40 words.

One of the characteristics of the parser developed here is that it uses words similar to both the head and the modifier for smoothing. The similarity-based smoothing method in

(Dagan et al., 1999) uses the words similar to only one of the words in a bigram. The definition of similar context can be changed as follows so that only one word in a similar context of C may be different from a word in C (see Model (b) in Table 5.2):

$$\begin{aligned} &S\left(\ [u,v,C^d_w]\ \right) \\ &= \left\{\ [u',v,C^d_w]\ |\ u' \in S(u)\right\} \cup \left\{\ [u,v',C^d_w]\ |\ v' \in S(v)\right\} \end{aligned}$$

where w is either v or u depending on whether d is L or R. This change leads to a 2.2% drop in accuracy (compared with Model (a) in Table 5.2), which is probably due to the fact that many contexts do not have similar contexts in the training corpus.

Since most previous parsing models maximize the joint probability of the sentence and the parse tree P(X, Y) instead of the conditional probability of P($Y|X$), I also implemented a joint model (see Model (c) in Table 5.2):

$$P(X,Y) = \prod_{i=1}^{N} \begin{matrix} P\left(E^L_{m_i}|m_i, C^L_{m_i}\right) \times P\left(E^R_{m_i}|m_i, C^R_{m_i}\right) \times \\ \left(1 - P\left(E^d_{h_i}|h_i, C^d_{h_i}\right)\right) \times P\left(m_i|h_i, C^{d_i}_{h_i}\right) \end{matrix}$$

where h_i and m_i are the head and the modifier of the i'th dependency link. The probability $P\left(m_i|h_i, C^{d_i}_{h_i}\right)$ is smoothed by averaging the probabilities$P\left(m_i|h_i, C^{d_i}_{h'_i}\right)$, where h'_i is a similar word of h_i, as in (Dagan et al., 1999). This change of using a joint model causes a dramatic decrease in accuracy, from 79.9% for the conditional model to 66.3% for the joint model.

In the model proposed here, the use of distributional word similarity can be viewed as assigning soft clusters to words. In contrast, parts-of-speech can be viewed as a hard clusters of words. Both the conditional and joint models can be modified to use part-of-speech tags instead of words. Since there are only a small number of tags, the modified models use maximum likelihood estimation without any smoothing except for a small probability constant for unseen events. Without smoothing, maximizing the conditional model is equivalent to maximizing the joint model. The accuracy of the unlexicalized models (see Model (d) and Model (e) in Table 5.2) is 71.1% which is considerably lower than the strictly lexicalised conditional model I have proposed, but higher than the strictly lexicalised joint model. This demonstrates that soft clusters obtained through distributional word similarity perform better than the part-of-speech tags, when used appropriately.

5.6 Related Work

Previous probabilistic parsing models (Collins, 1997; Charniak, 2000) maximize the joint probability $P(X,Y)$ of a sentence X and its parse tree Y. This chapter considers an approach that maximizes the conditional probability $P(Y|X)$. The use of conditional model allows one to take advantage of similarity-based smoothing.

Clark et al. (2002) also compute a conditional probability of dependency structures. While the probability space considered in this chapter consists of all possible projective dependency trees, their probability space is constrained to be all dependency structures that are allowed by a Combinatorial Category Grammar (CCG) and a category dictionary (lexicon). They therefore do not need the STOP markers in their model. Another major difference between the model presented here and (Clark et al., 2002) is that the parameters used here consist exclusively of conditional probabilities of binary variables.

Ratnaparkhi's maximum entropy model (Ratnaparkhi, 1999) is also a conditional model. However, his model maximizes the probability of the action during each step of the parsing process, instead of overall quality of the parse tree.

In many dependency parsing models such as (Eisner, 1996; McDonald et al., 2005a), the score of a dependency tree is the sum of the scores of the dependency links, which are computed independently of other links. An undesirable consequence of this is that the parser often creates multiple dependency links that are separately likely but jointly improbable (or even impossible). For example, there is nothing in such models to prevent the parser from assigning two subjects to a verb. In the DMV model (Klein and Manning, 2004), the probability of a dependency link is partly conditioned on whether or not there is a head word of the link that already has a modifier. The model proposed in this chapter is quite similar to the DMV model, except that it computes the conditional probability of the parse tree given the sentence, instead of the joint probability of the parse tree and the sentence.

There have been several previous approaches to parsing Chinese with the Penn Chinese Treebank (Bikel and Chiang, 2000; Levy and Manning, 2003). Both of these approaches employ phrase-structure joint models and use part-of-speech tags in back-off smoothing. Their results were evaluated with the precision and recall of the brackets implied in the phrase structure parse trees. In contrast, the accuracy of the proposed model is measured in terms of the dependency relationships. A dependency tree may correspond to more than one constituency trees. My results are therefore not directly comparable with the precision and recall values in previous research. Moreover, it was argued in (Lin, 1995) that dependency based evaluation is much more meaningful for the applications that use parse trees, since the semantic relationships are generally embedded in the dependency relationships.

Non-probabilistic approaches to dependency parsing, including large margin based techniques in particular will be considered in the next chapter.

5.7 Contributions

In this chapter, I presented a generative approach that employs Maximum Likelihood Markov Network training for dependency parsing. This model is similar to the maximum entropy Markov models (MEMMs). In both MEMMs and the model presented here, the goal is to maximize the conditional probability given the observations and previous state. The probability parsing model I presented is also very closely related to the score-based parsing model introduced in Section 2.4.1, since the product of each link probability can be converted to sums of score in the log space. Similarity-based smoothing was also applied to deal with data sparseness, instead of relying on part-of-speech tags or grammatical categories, which has not been applied to parsing before.

5.8 Conclusion

To the best of my knowledge, all previous natural language parsers have to rely on part-of-speech tags. In this chapter I presented a strictly lexicalised model for dependency parsing that only relies on word statistics. I compared the resulting parser with an unlexicalized parser that employs the same probabilistic model except that the parameters are estimated using gold standard tags in the Chinese Treebank. My experiments show that the strictly lexicalised parser significantly outperformed its unlexicalized counterpart.

An important distinction between the proposed statistical model and previous parsing models is that all the parameters in the model presented here are conditional probability of binary variables. This allows one to take advantage of similarity-based smoothing, which has not been successfully applied to parsing before.

Chapter 6

Extensions to Large Margin Dependency Parsing

The approach presented in Chapter 5 has a limitation; it uses a local scoring function instead of a global scoring function to compute the score for a candidate tree. The structured large margin approach (discussed in Section 3.5.2), on the other hand, uses a global scoring function by minimizing a training loss—the "structured margin loss" (Taskar et al., 2003; Tsochantaridis et al., 2004; McDonald et al., 2005a)—which is directly coordinated with the global tree. However, the training error minimized in the large margin approach is a coarse measure that only assesses the total error of an entire parse rather than focusing on the error of any particular component. Also smoothing methods, which have been widely used in probabilistic approaches, are not currently being used in large margin training algorithms. In this chapter I improve structured large margin training for parsing in two ways. First, I incorporate local constraints that enforce the correctness of each individual link, rather than just scoring the global parse tree. Second, to cope with sparse data and generalize to unseen words, I smooth the lexical parameters according to their underlying word similarities. To smooth parameters in the large margin framework, one needs to introduce the technique of Laplacian regularization in large margin parsing. Finally, to demonstrate the benefits of the proposed approach, I reconsider the problem of parsing Chinese treebank data using only lexical features, as in Chapter 5. My results show that improved accuracy can be obtained over current large margin approaches, and furthermore show that similarity smoothing combined with local constraint enforcement leads to state-of-the-art performance. Once again, these results only require word-based features, and do not rely on part-of-speech tags nor grammatical categories in any way.

In this chapter, I present my two modifications to large margin training for parsing, i.e., enforcing local parsing constraints and incorporating word similarity smoothing via

Laplacian regularization, in Section 6.1 and Section 6.2 respectively. Then I present my experimental results on fully lexical dependency parsing for Chinese. This work was published in Wang et al. (2006).

6.1 Large Margin Training with Local Constraints

In this chapter, I only use those simple, lexically determined features which are defined in Section 2.4.2, $\{f_{uv}\}$, f_{PMI}, f_{dist} and f_{dist2} (*without* the parts-of-speech $\{f_{pv}\}$). The corresponding parameters $\boldsymbol{\theta}$ for these features are, θ_{uv}, θ_{PMI}, θ_{dist}, θ_{dist2}. Moreover, I only use undirected forms of these features, where, for example, $f_{uv} = f_{vu}$ for all pairs (or, put another way, I tie the parameters $\theta_{uv} = \theta_{vu}$ together for all u, v). Ideally, I would like to use directed features, but I have already found that these simple undirected features permit state-of-the-art accuracy in predicting (undirected) dependencies. Nevertheless, extending my approach to directed features and contextual features (as in Chapter 5), remains an important direction for future research. Note that this is a much simpler feature set than that used in Chapter 5, here consisting only of static features, whereas Chapter 5 used dynamic features extensively. Unifying these two projects remains a direction for future work (Section 9.2.4 below).

6.1.1 Large Margin Training

To train the parameters $\boldsymbol{\theta}$, I follow the structured large margin training approach discussed in Section 3.5.2 (Taskar et al., 2003; Tsochantaridis et al., 2004), which has been applied with great success to dependency parsing (Taskar et al., 2004b; McDonald et al., 2005a). Even through I have already discussed the structured large margin training framework in Section 3.5.2, I will repeat some of the details here that I need to explain and derive my new approach.

Large margin training can be expressed as minimizing a regularized loss (Hastie et al., 2004) as shown in (3.11), which is equivalent to solving the following quadratic program

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \mathbf{e}^\top\boldsymbol{\xi} \quad \text{subject to} \\ & \xi_{i,k} \geq \Delta(L_{i,k}, Y_i) + score(\boldsymbol{\theta}, L_{i,k}) - score(\boldsymbol{\theta}, Y_i) \\ & \text{for all } i, L_{i,k} \in \Phi(X_i) \end{aligned} \tag{6.1}$$

where Y_i is the target tree for sentence X_i; $L_{i,k}$ ranges over all possible alternative trees in $\Phi(X_i)$; $score(\boldsymbol{\theta}, Y) = \sum_{(x_m \to x_n) \in Y} \boldsymbol{\theta}^\top \mathbf{f}(x_m \to x_n)$; and $\Delta(L_{i,k}, Y_i)$ is a measure of distance between the two trees $L_{i,k}$ and Y_i.

Unfortunately, the quadratic program (6.1) has three problems one must address. First, there are exponentially many constraints—corresponding to each possible parse of each training sentence—which forces one to use alternative training procedures, such as incremental constraint generation, to slowly converge to a solution (McDonald et al., 2005a; Tsochantaridis et al., 2004). Second, and related, the original loss (3.11) is only evaluated at the global parse tree level, and is not targeted at penalizing any specific component in an incorrect parse. Although (McDonald et al., 2005a) explicitly describes this as an advantage over previous approaches (Ratnaparkhi, 1999; Yamada and Matsumoto, 2003), below I find that changing the loss to enforce a more detailed set of constraints leads to a more effective approach. Third, given the large number of bi-lexical features $\{f_{uv}\}$ in the model, solving (6.1) directly will over-fit any reasonable training corpus. (Moreover, using a large β to shrink the $\boldsymbol{\theta}$ values does not mitigate the sparse data problem introduced by having so many features.) I now present my refinements that address each of these issues in turn.

6.1.2 Training with Local Constraints

Initially, I considered training on just an undirected link model, where each parameter in the model is a weight $\theta_{xx'}$ between two words, x and x', respectively. Since links are undirected, these weights are symmetric $\theta_{xx'} = \theta_{x'x}$, and the score can also be written in an undirected fashion as: $score(x, x') = \boldsymbol{\theta}^{\top}\mathbf{f}(x, x')$. The main advantage of working with the undirected link model is that the constraints needed to ensure correct parses on the training data are *much* easier to specify in this case. Ignoring the projective (no crossing arcs) constraint for the moment, an undirected dependency parse can be equated with a maximum score spanning tree of a sentence. Given a target parse, the set of constraints needed to ensure the target parse is in fact the maximum score spanning tree under the weights $\boldsymbol{\theta}$, by at least a minimum amount, is a simple set of linear constraints: for any edge x_1x_2 that is *not* in the target parse, one simply adds two constraints

$$\begin{aligned} \boldsymbol{\theta}^{\top}\mathbf{f}(x_1, x_1') &\geq \boldsymbol{\theta}^{\top}\mathbf{f}(x_1, x_2) + 1 \\ \boldsymbol{\theta}^{\top}\mathbf{f}(x_2, x_2') &\geq \boldsymbol{\theta}^{\top}\mathbf{f}(x_1, x_2) + 1 \end{aligned} \tag{6.2}$$

where the edges x_1x_1' and x_2x_2' are the adjacent edges that actually occur in the target parse that are also on the path between x_1 and x_2. (These would have to be the only such edges, or there would be a loop in the parse tree.) These constraints behave very naturally by forcing the weight of an omitted edge to be smaller than the adjacent included edges that would form a loop, which ensures that the omitted edge would not be added to the maximum score spanning tree before the included edges.

In this way, I can simply accumulate the set of linear constraints (6.2) for every edge that fails to be included in the target parse for the sentences where it is a candidate. This set of constraints can be denoted as

$$A = \{\boldsymbol{\theta}^\top \mathbf{f}(x_1, x_1') \geq \boldsymbol{\theta}^\top \mathbf{f}(x_1, x_2) + 1\} \tag{6.3}$$

Importantly, the constraint set A is *convex* in the link weight parameters $\boldsymbol{\theta}$, as it consists only of linear constraints.

Ignoring the non-crossing condition, the constraint set A is exact. However, because of the non-crossing condition, the constraint set A is more restrictive than necessary. For example, consider the word sequence $...x_i x_{i+1} x_{i+2} x_{i+3}...$, where the edge $x_{i+1}x_{i+3}$ is in the target parse. Then the edge $x_i x_{i+2}$ can be ruled out of the parse in one of two ways: it can be ruled out by making its score less than the adjacent scores as specified in (6.2), *or* it can be ruled out by making its score smaller than the score of $x_{i+1}x_{i+3}$. Thus, the exact constraint contains a disjunction of two different constraints, which creates a *non*-convex constraint in $\boldsymbol{\theta}$. (The union of two convex sets is not necessarily convex.) This is a weakening of the original constraint set A. Unfortunately, this means that, given a large training corpus, the constraint set A can easily become infeasible.

Nevertheless, the constraints in A capture much of the relevant structure in the data, and are easy to enforce. Therefore, I wish to maintain them. However, rather than impose the constraints exactly, I enforce them approximately through the introduction of slack variables $\boldsymbol{\xi}$. The relaxed constraints can then be expressed as

$$\boldsymbol{\theta}^\top \mathbf{f}(x_1, x_1') \geq \boldsymbol{\theta}^\top \mathbf{f}(x_1, x_2) + 1 - \xi_{x_1x_2, x_1x_1'} \tag{6.4}$$

and therefore a maximum soft margin solution can then be expressed as a quadratic program

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \boldsymbol{\xi}^\top \mathbf{e} \quad \text{subject to} \\ & \boldsymbol{\theta}^\top \mathbf{f}(x_1, x_1') \geq \boldsymbol{\theta}^\top \mathbf{f}(x_1, x_2) + 1 - \xi_{x_1x_2, x_1x_1'} \\ & \text{for all constraints in } A \end{aligned} \tag{6.5}$$

where $\mathbf{e}$ denotes the vector of all 1's.

Even though the slack variables are required because the parameters have been slightly over-constrained, given that there are so many parameters and a sparse data problem as well, it seems desirable to impose a stronger set of constraints. A set of solution parameters achieved in this way will allow maximum weight spanning trees to correctly parse nearly all of the training sentences, even without the non-crossing condition (see the results in Section 6.3).

This quadratic program has the advantage of producing link parameters that will correctly parse most of the training data. Unfortunately, the main drawback of this method thus far is that it does not offer any mechanism by which the link weights $\theta_{ww'}$ can be *generalized* to new or rare words. Given the sparse data problem, some form of generalization is necessary to achieve good test results. I achieve this by exploiting distributional similarities between words to smooth the parameters.

6.2 Laplacian Regularization

I wish to incorporate similarity based smoothing in large margin training, while using the more refined constraints outlined in Section 6.1.2.

Recall that most of the features that are used, and therefore most of the parameters that need to be estimated are based on bi-lexical parameters $\theta_{ww'}$ that serve as undirected link weights between words w and w' in my dependency parsing model (Section 5.2). Here I would like to ensure that two different link weights, $\theta_{w_1 w_1'}$ and $\theta_{w_2 w_2'}$, that involve similar words also take on similar values. The previous optimization (6.5) needs to be modified to take this into account.

Smoothing the link parameters requires one to first extend the notion of word similarity, which is defined in Section 3.6.2, to word-*pair* similarities, since each link involves two words. Given similarities between individual words, computed according to (3.13) and (3.14), the similarity between word pairs are then defined as the geometric mean of the similarities between corresponding words as in Section 5.3:

$$Sim(w_1 w_1', w_2 w_2') \quad = \quad \sqrt{Sim(w_1, w_2) Sim(w_1', w_2')} \qquad (6.6)$$

where $Sim(w_1, w_2)$ is the similarity between w_1 and w_2. Then, instead of just solving the constraint system (6.5), one can also ensure that similar links take on similar parameter values by introducing a penalty on their deviations that is weighted by their similarity value. Specifically, I use

$$\sum_{w_1 w_1'} \sum_{w_2 w_2'} Sim(w_1 w_1', w_2 w_2')(\theta_{w_1 w_1'} - \theta_{w_2 w_2'})^2 \quad = \quad 2\theta'^{\top} L(S) \theta' \qquad (6.7)$$

Here $L(S)$ is the *Laplacian matrix* of the similarity matrix S, which is defined by $L(S) = D(S) - S$ where $D(S)$ is a diagonal matrix such that $D_{w_1 w_1', w_1 w_1'} = \sum_{w_2 w_2'} S(w_1 w_1', w_2 w_2')$.

Also, $\boldsymbol{\theta}'$ corresponds to the vector of bi-lexical parameters. In this penalty function, if two edges w_1w_1' and w_2w_2' have a high similarity value, their parameters will be encouraged to take on similar values. By contrast, if two edges have low similarity, then there will be little mutual attraction imposed on their parameter values.

Note, however, that I do not smooth the parameters, θ_{PMI}, θ_{dist}, θ_{dist2}, corresponding to the pointwise mutual information, distance, and squared distance features described in Section 5.2, respectively. These are not sparse features. I only apply similarity smoothing to the bi-lexical parameters.

The Laplacian regularizer (6.7) provides a natural smoother for the bi-lexical parameter estimates that takes into account valuable word similarity information computed as above. The Laplacian regularizer also has a significant computational advantage: it is guaranteed to be a *convex* quadratic function of the parameters (Zhu et al., 2003). Therefore, by combining the constraint system (6.5) with the Laplacian smoother (6.7), I can obtain a convex optimization procedure for estimating the link parameters

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & \frac{\beta}{2}\boldsymbol{\theta}^\top \tilde{L}(S)\boldsymbol{\theta} + \boldsymbol{\xi}^\top \mathrm{e} \quad \text{subject to} \\ & \boldsymbol{\theta}^\top \mathbf{f}(w_1, w_1') \geq \boldsymbol{\theta}^\top \mathbf{f}(w_1, w_2) + 1 - \xi_{w_1w_2, w_1w_1'} \\ & \text{for all constraints in A (shown in Equation (6.3))} \end{aligned} \tag{6.8}$$

where $\tilde{L}(S)$ does not apply smoothing to θ_{PMI}, θ_{dist}, θ_{dist2}.

Clearly, (6.8) describes a large margin training program for dependency parsing, but one which uses word similarity smoothing for the bi-lexical parameters, and a more refined set of constraints developed in Section 6.1.2. Although the constraints are more refined, they are fewer in number than (6.1). That is, there are only a polynomial number of constraints corresponding to each word pair in (6.2), rather than the exponential number over every possible parse tree in (6.1). Thus, I obtain a polynomial size quadratic program that can be solved for moderately large problems using standard optimization software packages. I used CPLEX in my experiments below. As before, once optimized, the solution parameters $\boldsymbol{\theta}$ can be introduced into the dependency model shown in Equation (2.1) according to Equation (2.2) discussed in Section 2.4.

6.3 Experimental Results

I used the same experimental settings as in Section 5.5, except that I only experimented with CTB4-10 here, which contains sentences with no more than 10 words. The main

Features used	Trained w/ local loss	Trained w/ global loss
Pairs	61.30	56.88
+ Lap	63.90	49.35
+ Dist	63.64	61.30
+ Lap + Dist	64.94	52.99
+ MI + Dist	63.12	61.82
+ Lap + MI + Dist	65.71	56.62

Table 6.1: Accuracy on CTB4-10 dev set (%)

Features used	Trained w/ local loss	Trained w/ global loss
Pairs	64.26	61.84
+ Lap	65.06	56.22
+ Dist	65.46	64.66
+ Lap + Dist	65.86	55.42
+ MI + Dist	67.07	65.46
+ Lap + MI + Dist	68.27	57.43

Table 6.2: Accuracy on CTB4-10 test set (%)

Features used	Trained w/ local loss	Trained w/ global loss
Pairs	98.02	83.93
+ Lap	97.77	72.16
+ Dist	97.55	83.76
+ Lap + Dist	97.47	72.16
+ MI + Dist	97.68	79.85
+ Lap + MI + Dist	97.38	70.62

Table 6.3: Accuracy on CTB4-10 training set (%)

Model	Dev set	Test set
Chapter 6 with local loss	65.7	68.3
Chapter 6 with global loss	56.6	57.4
Chapter 5	61.0	76.3

Table 6.4: Comparison with the approach in Chapter 5 on CTB4-10 (%)

reason for using only short sentences is due to the computational cost in the learning phase. In Chapter 8, we will introduce an efficient optimization strategy that allows us to learn on longer sentences.

I use similarity information in both training and parsing. For training, I smooth the parameters according to their underlying word-*pair* similarities by introducing a Laplacian regularizer (see Section 6.2). For parsing, the link scores in (2.1) are smoothed by word-pair similarities before the maximum score projective dependency tree is computed, i.e., for any unseen link in the new sentences, the weight of the link is computed as the similarity weighted average of similar links seen in the training corpus (similar to the approach used in Chapter 5).

Table 6.1, Table 6.2 and Table 6.3 show the experimental results trained and evaluated on Chinese Treebank sentences of length no more than 10, using the standard data split. The regularization parameter β was set by 5-fold cross-validation on the training set. I evaluate parsing accuracy by comparing the undirected dependency links in the parser outputs against the undirected links in the treebank.

Table 6.1 and Table 6.2 show that training based on the more refined *local loss* is far superior to training with the *global loss* of standard large margin training, on both the test and development sets. Parsing accuracy also appears to increase with the introduction of each new feature. Notably, the pointwise mutual information and distance features significantly improve parsing accuracy—and yet I know of no other research that has investigated these features in this context. Finally, I note that Laplacian regularization improved performance as expected, but not for the *global loss*, where it appears to systematically degrade performance. It seems that the global loss model may have been over-regularized (Table 6.3). However, I have picked the β parameter which gave me the best results in my experiments. One possible explanation for this phenomenon is that the interaction between the Laplacian regularization in training and the similarity smoothing in parsing, since distributional word similarities are used in both cases.

Finally, I compared the results of this work to the probabilistic parsing approach proposed in Chapter 5, which on this data obtained accuracies of 61.0% on the development set and 76.3% on the test set, as shown in Table 6.4. The discrepancy between the two results of the probabilistic approach indicates its non-robustness, at least on the small data set used for the experiments. In fact, a complicated feature set is used in Chapter 5. Although it is trivial to incorporate these features into the current large margin framework presented here, I only used a much simpler feature set. Applying those complicated features to the large margin approach is a direction for my future work, which I will discuss in more detail in Section 9.2.

6.4 Related Work

Several research groups applied large margin criterion to parsing (Yamada and Matsumoto, 2003; Taskar et al., 2004a; Tsochantaridis et al., 2004; McDonald et al., 2005a). Yamada and Matsumoto (2003) use a multi-class SVM to learn a local scoring function for a dependency parser. Their model uses a discriminative method that maximizes the differences between scores of the link labels in the correct parse versus the scores of the top competing link labels. They obtain satisfactory dependency parsing results on English. However, the proposed approach uses a local training criteria, i.e., they only train the model parameters locally, not considering optimizing those parameters based on minimizing the global loss of the entire parse tree.

In the work of (Tsochantaridis et al., 2004), a structured support vector machine learning framework is presented which involves features extracted jointly from both inputs and outputs. The resulting quadratic program has an exponential number of constraints and is solved by an approximation algorithm which finds a small set of active constraints. Taskar et al. (2004a) also present a discriminative approach to parsing inspired by the large-margin criterion and they encountered the same problem as Tsochantaridis et al.—exponential number of constraints. They use a factorization approach which is analogous to the standard dynamic programming for parsing. Unlike Tsochantaridis et al. (2004) and Taskar et al. (2004a), there are only polynomial number of constraints in the large margin framework I presented, so that I can solve the problem much more efficiently. Recently, by using an approximate online large-margin training algorithm and considering only the k-best parse trees, McDonald et al. (2005a) achieve state-of-the-art performance in dependency parsing in English.

6.5 Contributions

Discriminative large-margin training to parsing has achieved the best results on dependency parsing for English. To date, however, in existing large-margin approaches, the training loss is only measured on the global tree level, not considering the local errors in a parse tree. Second, in the standard large-margin approach to parsing, there are an exponential number of constraints, making it impossible to solve the quadratic problem directly. Moreover, smoothing techniques, which are critical for natural language tasks and have been widely used in probability models, have not been applied to the large margin framework. I presented my refined approaches to deal with these issues one by one and obtained state-of-the-art performance, while only requiring word-based features that do not rely on part-of-speech tags nor grammatical categories.

6.6 Conclusion

I have presented two improvements to the standard large margin training approach for dependency parsing. To cope with the sparse data problem, I smooth the parameters according to their underlying word similarities by introducing a Laplacian regularizer. Secondly, I capture local errors of a parse tree instead of the global parse tree error, i.e., I use more refined local constraints in the large margin criterion, rather than the global parse-level losses that are commonly considered. These improvements result in state-of-the-art parsing accuracy for predicting undirected dependencies in test data, competitive with previous large margin and previous probabilistic approaches in experiments.

There are many directions for future work. One extension is to consider directed features, and contextual features like those used in current probabilistic parsers as discussed in Chapter 5. I would also like to apply this approach to parsing English, and possibly re-investigate the use of parts-of-speech features in this context.

Chapter 7

Training Dependency Parsers via Structured Boosting

Recent techniques for learning natural language parsers via coordinated training algorithms, such as conditional random fields and maximum margin Markov networks, have contributed significant progress. Unfortunately, as we have just seen in Chapter 6, these techniques are based on specialized training algorithms, are complex to implement, expensive to run, and require a great deal of refinement and computational resources to apply to parsing. In this chapter I present a much simpler approach to training dependency parsers, by applying a boosting-like procedure to standard local training methods. The idea is to learn a local link predictor using standard methods, such as logistic regression or support vector machines, but then achieve improved global parsing accuracy by "boosting" the influence of misclassified dependency links *after* re-parsing the training data.

By using logistic regression as an efficient base classifier for predicting dependency links between word pairs, I am able to efficiently train a dependency parsing model, via structured boosting, that achieves state-of-the-art results in English, and surpasses state-of-the-art in Chinese.

7.1 Introduction

One drawback with current structured prediction training algorithms for parsing is that they involve new, specialized parameter optimization algorithms, that are complex, non-trivial to implement, and usually require far more computation than standard classification learning methods (Lafferty et al., 2001; Taskar et al., 2003). The main reason for increased complexity is the fact that a parser must be considered in the underlying training principle, which causes the structured inference of output predictions to be tightly coupled with the parameter optimization process during training, as seen in Chapter 6.

In this chapter, I demonstrate the somewhat surprising result that state-of-the-art performance on dependency parsing can be achieved through the use of conventional, local classification methods. In particular, I show how a simple form of structured boosting can be used to improve the accuracy of standard local classification methods, in the structured case, without modifying the underlying training method. The advantage of this approach is that one can use off-the-shelf classification techniques, such as support vector machines or logistic regression, to achieve competitive dependency parsing results with little additional effort. I achieve this through the use of a very simple idea: Specifically, I introduce a very simple form of "structured boosting", where a parser is used to modify the predictions of the local, weak learning algorithm at each boosting round, which then influences the example weightings and subsequent hypotheses, implicitly improving the parser's performance. That is, although dependency parsing is a very complex problem, one can achieve state-of-the-art results by training a local "link predictor" that merely attempts to predict the existence and orientation of a link between two words given input features encoding context—without worrying about coordinating the predictions in a coherent global parse tree. Instead, a wrapper approach, based on structured boosting, is used to successively modify the training data so that the training algorithm is implicitly encouraged to facilitate improved global parsing accuracy.

Below I first introduce the main technique proposed in this chapter, explain its relation to standard boosting approaches, and then specify the detailed approach used in my experiments. Then in Section 7.4, I describe my experiments in learning dependency parsers from treebank data, and show how competitive results can be obtained through the use of standard learning methods. In fact, the results surpass state-of-the-art accuracy in Chinese parsing, and are competitive with state-of-the-art in English. This work was published in Wang et al. (2007).

7.2 Structured Boosting

I now describe the simple idea, structured boosting, that provides a straightforward way to combine parsing with local parameter optimization, without modifying the underlying local training algorithm. In fact, the procedure is a trivial variant of standard boosting algorithms (Freund and Schapire, 1996; Schapire and Singer, 1999; Collins et al., 2002), altered to incorporate a dependency parsing algorithm during the classification phase. The procedure is as follows.

- First, train a standard predictor on the labeled training sentences, as discussed in Section 3.1, to produce a "weak" local link predictor for parsing.
- Then use a dependency parsing algorithm to re-predict the training labels, in a coordinated global fashion, using the learned link predictor as an internal scoring function.
- Based on the resulting misclassifications of the parser output, calculate the ensemble weight for the current weak local link predictor, and update the local example weights, according to any standard boosting method; for example, either exponential loss Adaboost (Freund and Schapire, 1996; Schapire and Singer, 1999) or logistic regression loss boosting (Collins et al., 2002). Note that this will increase the weight of all links incorrectly classified by the parser.
- Repeat the above steps for some number of boosting rounds.

The resulting ensemble of weak local link predictors then provides a combined local predictor that can be used for subsequent global dependency parsing on test sentences.

The advantage of this approach is its simplicity and generality. It can be applied to any standard local training method without requiring any modification of the underlying algorithm, yet via structured boosting, the local learning algorithm is forced to respond to the behavior of the parser. In effect, it is a simple training wrapper, where local examples are reweighted, not based on the predictions of a current hypothesis, but instead on the predictions that the local hypothesis forces the parser to make. Below I find that a structured boosting method of this form can improve the quality of dependency parsers learned from treebank data. Note that only a few boosting rounds are ever feasible in my application, because each round requires the entire corpus to be re-parsed and the local prediction model re-trained. Nevertheless, I still witness some useful improvements and achieve state-of-the-art results.

7.3 Implementation Details

The goal of this chapter is to show that a simple dependency parsing algorithm can provide improved parsing accuracy via structured boosting. Like all the other discriminative training approaches, this algorithm can make use of a rich feature set, including inter-dependent or non-local features. Unlike other global training approaches, structured boosting is very easy to implement and as cheap as local methods to run.

Implementation of this strategy requires one to specify the features used, the local training algorithm, the parsing algorithm and the outer boosting method.

Static Features

For both English and Chinese I used a common set of feature templates as discussed in Section 2.4.2. Specifically, I used the same set of features described in (McDonald et al., 2005a), except the "In Between POS Features" (see Table 2.1). Given a target word pair and their context, these static features consisted of indicators of the individual words, their part-of-speech tags, and also the part-of-speech tags of words in the surrounding context. In addition to the indicator features used in (McDonald et al., 2005a), I also added a distance feature as discussed in Section 2.4.2 that simply measures how far apart the two words are in the sentence, which is highly predictive of link existence, since most links in a dependency parse are short range.

Dynamic Features

For dynamic features, as described in Section 2.4.3, I used the number of previous children of a candidate head word, and an indicator of the part-of-speech, if any, of the previous child word on the same side of the candidate head word. For English, I used one special dynamic feature to try to capture prepositional phrase attachment preference: if a candidate child is tagged as PP (prepositional phrase), then I use a feature that indicates the tag and word of the first grandchild (first child of the child). The experimental results to be presented in the next section show that simple dynamic features easily improve a parser's performance. Here I use a bigger feature set than the work presented in Chapter 5 and Chapter 6. I used both static and dynamic features in Chapter 5, while in Chapter 6, I used the distance and mutual information features as well as static features, but no dynamic features were used. However, in both Chapter 5 and Chapter 6, I did not use any features which involve part-of-speech tags.

Local Training

For the local training algorithm I used a standard logistic regression model (aka maximum entropy model), as discussed in Section 3.4.1. The local learner attempts to predict one of three word pair labels (no link, left link, right link) for each word pair in a sentence, given the features described above. To deal with the problem of overfitting, I use the regularization technique proposed by Goodman (2004) as discussed in Section 3.4.1. The regularization parameter, α, was set to 0.5 in the experiments below. This parameter was selected with some tuning on the English development set, and then used without modification on the other data sets. Unfortunately, the number of features and number of local examples were both so large that training the logistic regression model, even once, took more than a day. So to accelerate the training process, I employed one further trick: I partitioned the set of local examples (determined by word pairs in each sentence) according to the part-of-speech tags of the pair. Within each equivalence class, the number of features could then be further reduced by dropping those features that became constant within the class. This partitioning dropped the overall training cost to a few hours on a few computers, since the separate partitions could then be trained in parallel. Interestingly, the quality of the learned model was not significantly affected by this training procedure. This suggests that the part-of-speech tags of the word pair, which are used to create the partitions, are the most essential piece of information in deciding the existence and orientation of a dependency link.

Parser

There are many dependency parsing algorithms available with differing computational cost, as discussed in Section 2.3. In my experiments, I used the dependency parsing algorithm described in Section 5.4, which allowed me to use all of the features described above, while also enforcing the planarity constraint.

Boosting Method

I experimented with a simplified variant of boosting where the weights of each mis-parsed local example were simply increased by an additive constant, with other weights kept the same, and only the last hypothesis is kept. In fact, in my experiments below I obtain state-of-the-art results just using this simplified procedure, and so I focus on these initial results here. Comparisons to standard boosting algorithms, such as Adaboost M1, M2 (Freund and Schapire, 1997) and the logistic regression form of boosting described in (Collins et al., 2002) remain areas for future research.

7.4 Experimental Results

To determine the effectiveness and generality of my approach I conducted a number of experiments on each of the data sets (English and Chinese). These results were achieved using only the simplified boosting procedure mentioned above (additive weight updates, keeping only the last hypothesis).

I used the English Penn Treebank, the Chinese Treebank 4.0 and the Chinese Treebank

Iter	English (PTB)			Chinese (CTB4)		
	DA	RA	CM	DA	RA	CM
1	87.77	89.61	27.44	82.20	88.58	19.38
2	88.08	89.61	28.97	82.33	89.62	19.72
3	88.10	89.74	28.81	82.22	89.97	18.69
4	88.49	90.62	29.04	82.79	89.97	19.38

Table 7.1: Boosting with static features (%)

5.0 for my experiments as described in Section 4.1. I trained and tested on the full set of all the data sets, while I experimented only on sentences with limited length on the Chinese Treebank 4.0 in Chapter 5 and Chapter 6.

First, to determine the effectiveness of the basic structured boosting idea, I started with a simple local prediction model (static features only) and measured parsing accuracy on the held out test set as a function of the number of boosting rounds. Table 7.1 shows that parsing accuracy is improved in each round of boosting with static features, on both English and Chinese (using the Chinese Treebank 4.0). To explain the improvements more carefully, note that I used dependency accuracy (DA), root accuracy (RA) and complete match (CM) for evaluation, as discussed in Section 4.2. My overall focus in this thesis, however, is on improving the *dependency accuracy* scores, rather than the root accuracy and complete match scores. This fact is reflected in the boosting procedure, since instance reweighting is based only on whether each candidate link is predicted correctly, not whether the root is labeled correctly, nor whether the complete sentence is matched correctly.

Not surprisingly, Table 7.1 shows that the dependency accuracy (DA) improves on each round of boosting for English, and improves on most rounds (and improves overall) for Chinese; while the RA and CM results fluctuate somewhat. Note that although the improvements appear small, the observed DA differences are all statistically significant. For English, the test corpus consists of 564,848 instances (word pairs occurring in a sentence), and differences of 0.02 in the percentages shown in the tables are statistically significant with greater than 99% confidence. For Chinese, the test corpus consists of 99,922 instances, and differences of 0.05 in the percentages shown in the tables are statistically significant with greater than 99% confidence.

Second, to determine the effectiveness of dynamic features, I added these additional features to the local prediction model and repeated the previous boosting experiment. Table 7.2 shows a significant further improvement in parsing accuracy over just using the static fea-

Iter	English (PTB)			Chinese (CTB4)		
	DA	RA	CM	DA	RA	CM
1	89.10	90.36	33.77	86.08	92.39	25.26
2	89.15	89.65	34.56	86.25	92.39	26.64
3	89.20	89.69	34.31	86.45	91.70	28.72
4	89.22	90.20	34.35	86.58	92.04	28.37

Table 7.2: Boosting with dynamic features (%)

tures alone (Table 7.1). Once again, however, boosting provides further improvement over the base model on both English and Chinese with respect to dependency accuracy. In each case the improvement is significant.

Finally, I compare the results I was able to achieve to the state-of-the-art. Table 7.3 shows the best results achieved by my method and other researchers on English and Chinese data. Once again, all of the results on English are obtained on the same standard training and test set splits on the English Penn Treebank. The results on Chinese are obtained on two different data sets, Chinese Treebank 4.0 and Chinese Treebank 5.0 as noted. These treebanks were discussed in Section 2.5. In Table 7.3, Y&M03 refers to (Yamada and Matsumoto, 2003), N&S04 refers to (Nivre and Scholz, 2004), Chap5 refers to my own work discussed in Chapter 5, MIRA05 refers to (McDonald et al., 2005a), MIRA06 refers to (McDonald and Pereira, 2006), BPM06 refers to (Corston-Oliver et al., 2006). Finally, Chap7 refers to the approach outlined in this chapter. From Table 7.3 one can see that on English, the results achieved through the simple boosting method are competitive with the state-of-the-art, but are still behind the best results of (McDonald and Pereira, 2006). However, the results obtained on CTB4 are significantly better than the ones presented in Chapter 5. Moreover, perhaps surprisingly, Table 7.3 shows that the technique I have proposed in this chapter actually achieves state-of-the-art accuracy on Chinese parsing for both treebank collections. [1] [2]

I did not compare the results with those presented in Chapter 6. The reason is that I only parsed sentences (from CTB4) with less than or equal to 15 words in Chapter 6, however, I parsed all the sentences in the entire corpus here.

[1]The results on Chinese Treebank 5.0 are generally worse than on Chinese Treebank 4.0, since the former is a superset of the latter, and moreover the additional sentences come entirely from a Taiwanese Chinese source that is more difficult to parse than the rest of the data (Palmer *et al.*, 2005; Palmer *et al.*, 2004).

[2]In fact, MIRA has been tried on Chinese Treebank 4.0 with the same data split reported above, obtaining a dependency accuracy score of 82.5, which does not match the 86.6 percent dependency accuracy achieved by the boosting technique on this data (Ryan McDonald, personal communication).

Model	English (PTB)			Chinese (CTB4 & CTB5)		
	DA	RA	CM	DA	RA	CM
Y&M03	90.3	91.6	38.4	-	-	-
N&S04	87.3	84.3	30.4	-	-	-
Chap5	-	-	-	79.9*	-	-
MIRA05	90.9	94.2	37.5	-	-	-
MIRA06	91.5	-	42.1	-	-	-
BPM06	90.8	93.7	37.6	73.3†	66.2†	18.2†
Chap7 model	89.2	90.2	34.4	77.6† 86.6*	60.6† 92.0*	13.5† 28.4*

Table 7.3: Comparison with state-of-the-art (%)

* Obtained with Chinese Treebank 4.0 using the data split reported in Chapter 5.
† Obtained with Chinese Treebank 5.0 using the data split reported in (Corston-Oliver et al., 2006).

Computational Complexity

Clearly, there is some computational overhead associated with training by boosting, since each round requires the base learning algorithm to be re-trained on the re-weighted training data. The training cost scales up proportional to the number of boosting iterations however, and reasonable improvements can be achieved with a small number of rounds. Interestingly, I have found for test complexity, the computational cost of using a composite hypothesis for scoring the local predictions does not add much overhead to the parsing complexity (although I report only single hypothesis results here).

7.5 Contributions

Most of the previous parsing work used simple likelihood based approaches or local training methods. Recently, structured training algorithms have been applied to parsing and achieved state-of-the-art accuracy for English dependency parsing. Unfortunately, The main drawback with current structured training techniques is that they are specialized, non-trivial to implement, and require a great deal of refinement and computational resources to apply to a significant task like parsing.

As described earlier in this chapter, I propose a simpler, more general approach, structured boosting, which can be applied to any local link classifier, without requiring the underlying training algorithm be modified, while still ensuring that the training outcome is directly influenced by the resulting accuracy of the parser. Therefore, structured boosting provides a new learning framework, which only needs as much computational cost as local learning algorithms while provides global training accuracy via boosting, as I have shown in Section 7.4.

7.6 Conclusion

I have addressed the problem of learning dependency parsers by using a simple form of boosting to augment the training of standard local link classifiers. The procedure is general, and allows one to improve performance at global parsing accuracy without modifying the underlying training algorithm, nor implementing a complex training algorithm. Further improvements in dependency parsing accuracy are easily obtained by using dynamic features that consider the link labels of surrounding word pairs.

Although the results are very promising, and in fact provide the new state-of-the-art result in Chinese dependency parsing, there remain many directions for future work. One obvious direction is to investigate the effect of using alternative boosting algorithms, and also to investigate the theoretical nature of applying these algorithms to the structured boosting case: under what circumstances do the algorithms converge, and what guarantees can be made about their performance. I would also like to explore further ideas about useful features for dependency parsing, and additional smoothing and regularization techniques for local training.

Chapter 8

Semi-supervised Convex Training for Dependency Parsing

In previous chapters I presented three algorithms for dependency parsing: 1, a maximum likelihood based approach with similarity based smoothing; 2, an improved large margin approach with a refined objective that considers local constraints and uses a Laplacian regularizer; and 3, structured boosting, which uses a simple form of boosting to augment the training of local link classifiers to improve global parsing accuracy. However, all of these training algorithms are fundamentally *supervised*, which means that they require fully labeled data as input. On the other hand, semi-supervised learning has become a major topic in machine learning over the past few years. The goal of of semi-supervised learning is to improve the accuracy of a learned predictor by exploiting auxiliary unlabeled data in addition to labeled data. Unfortunately, the training loss used by standard supervised algorithms, such as support vector machines, becomes non-convex in the presences of missing labels, which causes tremendous difficulty in parameter optimization. Although semi-supervised learning is obviously a critical idea, it is quite difficult to develop an efficient semi-supervised learning algorithm for a complex, large-scale task like parsing.

In this chapter, I present a novel semi-supervised training algorithm for learning dependency parsers. By combining a supervised large margin loss with an unsupervised least squares loss, a discriminative, convex semi-supervised learning algorithm is obtained that can be applied to large-scale problems. To demonstrate the benefits of this approach, I apply the technique to learning dependency parsers from labeled and unlabeled corpora. Using a stochastic gradient decent algorithm, a semi-supervised dependency parsing model can be learned efficiently that significantly outperforms corresponding supervised methods.

8.1 Introduction

As I have introduced in previous chapters, supervised learning algorithms have achieved state-of-the-art accuracy on dependency parsing, as shown in the work of (McDonald et al., 2005a; McDonald and Pereira, 2006) and my own work presented in Chapter 7. However, a key drawback of supervised training algorithms is that they can only take labeled data as input, while labeled data is usually very difficult to obtain. Perceiving the limitation of supervised learning—in particular, the heavy dependence on annotated corpora—many researchers have investigated *semi-supervised* learning techniques that can take both labeled and unlabeled training data as input. Following the common theme of "more data is better data" I also use both limited labeled corpora and a plentiful unlabeled data resource. My goal is to obtain better performance than a purely supervised approach with only modest additional computational effort. Unfortunately, although significant recent progress has been made in the area of semi-supervised learning, the performance of semi-supervised learning algorithms still fall far short of expectations, especially in challenging real-world tasks such as natural language parsing or machine translation.

A large number of distinct approaches to semi-supervised training algorithms have been investigated in the literature (Bennett and Demiriz, 1998; Zhu et al., 2003; Altun et al., 2005; Mann and McCallum, 2007). Among the most prominent approaches are self-training, generative models, semi-supervised support vector machines (S3VM), graph-based algorithms and multi-view algorithms (Zhu, 2005). Self-training is a commonly used technique for semi-supervised learning that has been applied to several natural language processing tasks (Yarowsky, 1995; Charniak, 1997; Steedman et al., 2003). The basic idea is to bootstrap a supervised learning algorithm by alternating between inferring the missing label information and retraining. Recently, McClosky et al. (2006a) successfully applied self-training to parsing by exploiting available unlabeled data, and obtained remarkable results when the same technique was applied to parser adaptation (McClosky et al., 2006b). More recently, Haffari and Sarkar (2007) have extended the work of Abney (2004) and obtained a better mathematical understanding of self-training algorithms. They also show connections between these algorithms and other related machine learning algorithms.

Another approach, generative probabilistic models, are a well-studied framework that can be extremely effective. However, generative models use the EM algorithm for parameter estimation in the presence of missing labels, which is notoriously prone to getting stuck in poor local optima. Moreover, EM optimizes a marginal likelihood score that is not discriminative. Consequently, most previous work that has attempted semi-supervised or unsupervised approaches to parsing have not produced results beyond the state-of-the-art supervised results (Klein and Manning, 2002; Klein and Manning, 2004). Subsequently, alternative estimation strategies for unsupervised learning have been proposed, such as my own work on POS tagging by using better smoothing techniques or added constraints (Wang and Schuurmans, 2005), or *Contrastive Estimation* (CE) by Smith and Eisner (2005a). Contrastive Estimation is a generalization of EM, by defining a notion of learner guidance. It makes use of a set of examples (its *neighborhood*) that are similar in some way to an observed example, requiring the learner to move probability mass to a given example, taking only from the example's neighborhood. Nevertheless, CE still suffers from shortcomings, including local minima.

In recent years, SVMs have demonstrated state-of-the-art results in many supervised learning tasks. As a result, many researchers have put effort into developing algorithms for semi-supervised SVMs (S3VMs) (Bennett and Demiriz, 1998; Altun et al., 2005). However, the standard objective of an S3VM is non-convex on the unlabeled data, thus requiring sophisticated global optimization heuristics to obtain reasonable solutions. A number of researchers have proposed several efficient approximation algorithms for S3VMs (Bennett and Demiriz, 1998; Chapelle and Zien, 2005; Xu and Schuurmans, 2005). For example, Chapelle and Zien (2005) propose an algorithm that smoothes the objective with a Gaussian function, and then performs a gradient descent search in the primal space to achieve a local solution. An alternative approach is proposed by Xu and Schuurmans (2005) who formulate a semi-definite programming (SDP) approach. In particular, they present an algorithm for multi-class unsupervised and semi-supervised SVM learning, which relaxes the original non-convex objective into a close convex approximation, thereby allowing a global solution to be obtained. However, the computational cost of SDP is still quite expensive.

Instead of devising various techniques for coping with non-convex loss functions, I approach the problem from a different perspective. I simply replace the non-convex loss on unlabeled data with an alternative loss that is jointly convex with respect to both the model parameters and the (encoding of) the self-trained prediction targets. More specifically, for the loss on the unlabeled data part, I substitute the original unsupervised structured SVM loss with a least squares loss, but keep constraints on the inferred prediction targets, which avoids trivialization. Although using a least squares loss function for classification appears misguided, there is a precedent for just this approach in the early pattern recognition literature (Duda et al., 2000). The least squares loss function has the advantage that the entire training objective on both the labeled and unlabeled data now becomes convex, since it consists of a convex structured large margin loss on labeled data and a convex least squares loss on unlabeled data. As I will demonstrate below, this approach admits an efficient training procedure that can find a global minimum, and, perhaps surprisingly, can systematically improve the accuracy of supervised training approaches for learning dependency parsers.

Thus, in this chapter, I focus on *semi-supervised* language learning, where I can make use of both labeled and unlabeled data. In particular, I investigate a semi-supervised approach for structured large margin training, where the objective is a combination of two convex functions, the structured large margin loss on labeled data and the least squares loss on unlabeled data. I apply the resulting semi-supervised convex objective to dependency parsing, and obtain significant improvement over the corresponding supervised structured SVM. Note that my approach is different from the self-training technique proposed in (McClosky et al., 2006a), although both methods belong to semi-supervised training category.

In the remainder of this chapter, I first review the supervised structured large margin training technique. Then I introduce the standard semi-supervised structured large margin objective, which is non-convex and difficult to optimize. Subsequently, I will present a new semi-supervised training algorithm for structured SVMs that is based on a convex formulation of the optimization problem. Finally, I apply this algorithm to dependency parsing and show improved dependency parsing accuracy for both Chinese and English. This work has been published in (Wang et al., 2008).

8.2 Supervised Structured Large Margin Training

As I discussed in previous chapters, supervised structured large margin training approaches have been applied to parsing and produce promising results, which has been discussed in the work of (Taskar et al., 2004b; McDonald et al., 2005a), and also in my own work presented in Chapter 6. In particular, as mentioned in Section 3.5.2, Equation (3.11), structured large margin training can be expressed as minimizing a regularized loss (Hastie et al., 2004), which I repeat again below for reference:

$$\min_{\boldsymbol{\theta}} \quad \frac{\beta}{2}\boldsymbol{\theta}^{\top}\boldsymbol{\theta} + \sum_i \max_{L_{i,k}} \Delta(L_{i,k}, Y_i) - (score(\boldsymbol{\theta}, Y_i) - score(\boldsymbol{\theta}, L_{i,k})) \qquad (8.1)$$

where Y_i is the target tree for sentence X_i; $L_{i,k}$ ranges over all possible alternative k trees in $\Phi(X_i)$; $score(\boldsymbol{\theta}, Y_i) = \sum_{(x_m \to x_n) \in Y_i} \boldsymbol{\theta} \cdot \mathrm{f}(x_m \to x_n)$, as shown in Section 2.4.1; and $\Delta(L_{i,k}, Y_i)$ is a measure of distance between the two trees $L_{i,k}$ and Y_i. This is an application of the structured large margin training approach first proposed in (Taskar et al., 2003) and (Tsochantaridis et al., 2004).

The above standard large margin training approach has limitations on finding local parse tree errors and dealing with data sparseness. In Chapter 6, I proposed a new approach (shown in Equation 6.8) which enforces local parsing constraints and incorporates distributional word similarity smoothing via Laplacian regularization. However, I only parsed sentences (from CTB4) with less than or equal to 15 words, since the system is difficult to scale up. McDonald et al. (2005a) used an online large margin training approach and obtained state-of-the-art dependency parsing results on English when trained on the whole corpus. Therefore, in this chapter, I will only consider the standard supervised large margin training approach instead of the one I proposed in Chapter 6 for comparison.

To compare with the new semi-supervised approach I will present in Section 8.4 below, I re-implemented the supervised structured large margin training approach in the experiments in Section 8.7, where I used a much smaller feature set than (McDonald et al., 2005a) and still obtained very promising results. More specifically, I optimize the following convex objective on the supervised data (which is based on Equation 8.1):

$$\min_{\boldsymbol{\theta}} \quad \frac{\alpha}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \sum_i \max_L \sum_{m=1}^{k}\sum_{n=1}^{k} \Delta(L_{i,m,n}, Y_{i,m,n}) - \mathit{diff}(\boldsymbol{\theta}, Y_{i,m,n}, L_{i,m,n}) \qquad (8.2)$$

where $\mathit{diff}(\boldsymbol{\theta}, Y_{i,m,n}, L_{i,m,n}) = \mathit{score}(\boldsymbol{\theta}, Y_{i,m,n}) - \mathit{score}(\boldsymbol{\theta}, L_{i,m,n})$ and k is the sentence length. I represent a dependency tree as a $k \times k$ adjacency matrix. In the adjacency matrix, the value of $Y_{i,m,n}$ is 1 if the word m is the head of the word n, 0 otherwise. Since both the distance function $\Delta(L_i, Y_i)$ and the score function decompose over links, solving (8.2) is equivalent to solve the original constrained quadratic program shown in (3.12).

8.3 Semi-supervised Structured Large Margin Objective

The objective of standard semi-supervised structured SVM is a combination of structured large margin losses on both labeled and unlabeled data. It has the following form:

$$\min_{\boldsymbol{\theta}, \mathbf{Y}_j} \quad \frac{\alpha}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \sum_{i=1}^{N} \mathit{structured\_loss}\,(\boldsymbol{\theta}, X_i, Y_i) + \sum_{j=1}^{U} \mathit{structured\_loss}\,(\boldsymbol{\theta}, X_j, Y_j) \quad (8.3)$$

where

$$\mathit{structured\_loss}\,(\boldsymbol{\theta}, X_i, Y_i) = \max_L \sum_{m=1}^{k}\sum_{n=1}^{k} \Delta(L_{i,m,n}, Y_{i,m,n}) - \mathit{diff}(\boldsymbol{\theta}, Y_{i,m,n}, L_{i,m,n})$$

N and U are the number of labeled and unlabeled training sentences respectively, and $\mathbf{Y}_j$ ranges over guessed targets on the unsupervised data.

Note that in the second term of the above objective shown in (8.3), both $\boldsymbol{\theta}$ and $\mathbf{Y}_j$ are variables. The resulting loss function has a hat shape (usually called hat-loss), which is non-convex. Therefore the whole objective is non-convex, making the search for global optimal difficult. Note that the root of the optimization difficulty for S3VMs is the non-convex property of the second term in the objective function. I will propose a novel approach which can deal with this problem. I introduce an efficient approximation—least squares loss—for the structured large margin loss on unlabeled data below.

8.4 Semi-supervised Convex Training for Structured SVMs

Although semi-supervised structured SVM learning has been an active research area, semi-supervised structured SVMs have not been used in many real applications to date. The main reason is that most available semi-supervised large margin learning approaches are non-convex or computationally expensive (e.g. (Xu and Schuurmans, 2005)). These techniques are difficult to implement and extremely hard to scale up. I present a semi-supervised algorithm for structured large margin training, whose objective is a combination of two convex terms: the supervised structured large margin loss on labeled data and the cheap least squares loss on unlabeled data. The combined objective is still convex, easy to optimize and much cheaper to implement.

8.4.1 Least Squares Convex Objective

Before I introduce the new algorithm, I first introduce a convex loss which I apply it to unlabeled training data for the semi-supervised structured large margin objective which I will introduce in Section 8.4.2 below. More specifically, I use a *structured* least squares loss to approximate the structured large margin loss on unlabeled data. The corresponding objective is:

$$\min_{\boldsymbol{\theta},\mathbf{Y}_j} \quad \frac{\alpha}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \frac{\lambda}{2}\sum_{j=1}^{U}\sum_{m=1}^{k}\sum_{n=1}^{k}\left(\boldsymbol{\theta}^\top \mathbf{f}(X_{j,m} \to X_{j,n}) - Y_{j,m,n}\right)^2 \tag{8.4}$$

subject to constraints on $\mathbf{Y}$ (explained below).

The idea behind this objective is that for each possible link $(X_{j,m} \to X_{j,n})$, the goal is to minimize the difference between the link and the corresponding estimated link based on the learned weight vector. Since this is conducted on unlabeled data, one needs to estimate both $\boldsymbol{\theta}$ and $\mathbf{Y}_j$ to solve the optimization problem. As mentioned in Section 8.2, a dependency tree $\mathbf{Y}_j$ is represented as an adjacency matrix. Thus one needs to enforce some constraints in the adjacency matrix to make sure that each $\mathbf{Y}_j$ satisfies the dependency tree constraints. These constraints are critical because they prevent (8.4) from having a trivial solution in $\mathbf{Y}$. More concretely, suppose one uses rows to denote heads and columns to denote children. Then the following constraints are obtained on the adjacency matrix:

1. All entries in $\mathbf{Y}_j$ are between 0 and 1 (convex relaxation of discrete directed edge indicators);

2. The sum over all the entries on each column is equal to one (one-head rule);

3. All the entries on the diagonal are zeros (no self-link rule);

4. $Y_{j,m,n} + Y_{j,n,m} \leq 1$ (anti-symmetric rule), which enforces directedness.

One final constraint that is sufficient to ensure that a spanning tree is obtained, is connectedness (no-cycle), which can be enforced with an additional semidefinite constraint. Although convex, this last constraint is more expensive to enforce than the others, therefore I drop it in my experiments below.

Critically, the objective (8.4) is *jointly* convex in both the weights $\boldsymbol{\theta}$ and the edge indicator variables $\mathbf{Y}$. This means, for example, that there are no local minima in (8.4)—*any* iterative improvement strategy, if it converges at all, must converge to a global minimum.

8.4.2 Semi-supervised Convex Objective

By combining the convex structured SVM loss on labeled data (shown in Equation (8.1)) and the convex least squares loss on unlabeled data (shown in Equation (8.4)), one can obtain a semi-supervised structured large margin loss

$$\min_{\boldsymbol{\theta},\mathbf{Y}_j} \quad \frac{\alpha}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta}+\sum_{i=1}^{N} \mathit{structured\_loss}\,(\boldsymbol{\theta}, X_i, Y_i)+\sum_{j=1}^{U} \mathit{least\_squares\_loss}\,(\boldsymbol{\theta}, X_j, Y_j) \quad (8.5)$$

subject to constraints on $\mathbf{Y}$ (explained above).

Since the summation of two convex functions is also convex, (8.5) must be jointly convex in $\boldsymbol{\theta}$ and $\mathbf{Y}_j$. Replacing the two losses with the terms shown in Equation (8.2) and Equation (8.4), one obtains the final convex objective as follows:

$$\begin{aligned}\min_{\boldsymbol{\theta},\mathbf{Y}_j} \quad & \frac{\alpha}{2N}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \sum_{i=1}^{N} \max_{L} \sum_{m=1}^{k}\sum_{n=1}^{k} \Delta(L_{i,m,n}, Y_{i,m,n}) - \mathit{diff}(\boldsymbol{\theta}, Y_{i,m,n}, L_{i,m,n}) + \\ & \frac{\alpha}{2U}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \frac{\lambda}{2}\sum_{j=1}^{U}\sum_{m=1}^{k}\sum_{n=1}^{k}\left(\boldsymbol{\theta}^\top \mathbf{f}(X_{j,m} \rightarrow X_{j,n}) - Y_{j,m,n}\right)^2 \qquad (8.6)\end{aligned}$$

subject to constraints on $\mathbf{Y}$ (explained above), where as before $\mathit{diff}(\boldsymbol{\theta}, Y_{i,m,n}, L_{i,m,n}) = \mathit{score}(\boldsymbol{\theta}, Y_{i,m,n}) - \mathit{score}(\boldsymbol{\theta}, L_{i,m,n})$, and N and U are the number of labeled and unlabeled training sentences respectively. Note that in (8.6) the regularizer has been split into two parts; one for the supervised component of the objective, and the other for the unsupervised component. Thus the semi-supervised convex objective is regularized proportionally to the number of labeled and unlabeled training sentences.

8.5 Efficient Optimization Strategy

To solve the convex optimization problem shown in Equation (8.6), I used a gradient descent approach which simply uses stochastic gradient steps. The procedure is as follows.

- Step 0, initialize the $\mathbf{Y}_j$ variables of each unlabeled sentence as a right-branching (left-headed) chain model, i.e. the head of each word is its left neighbor. Hence, the optimization begins with a feasible starting point.
- Step 1, pass through all the labeled training sentences one by one. The parameters $\boldsymbol{\theta}$ are updated based on each labeled sentence.
- Step 2, based on the learned parameter weights from the labeled data, update $\boldsymbol{\theta}$ and $\mathbf{Y}_j$ on each unlabeled sentence alternatively.
  - treat $\mathbf{Y}_j$ as constants, update $\boldsymbol{\theta}$ on each unlabeled sentence by taking a local gradient step.
  - treat $\boldsymbol{\theta}$ as constants, update $\mathbf{Y}_j$ by calling the optimization software package CPLEX to solve for an optimal local solution.
- Repeat the procedure of step 1 and step 2 until maximum iteration number has reached.

This procedure works efficiently on the task of dependency parsing. Although $\boldsymbol{\theta}$ and $\mathbf{Y}_j$ are updated locally on each sentence, the objective shown in Equation (8.6) is globally minimized after each iteration. In the experiments, the objective usually converges within 30 iterations.

8.6 Implementation Details

To investigate the effectiveness of this approach I implemented a version using a simple feature set and parsing algorithm.

Features

For simplicity, in this work, I only used two sets of features—word-pair and tag-pair indicator features, which are part of features used in Chapter 7. Although the algorithm can take arbitrary features, by only using these simple features, I already obtained very promising results on dependency parsing using both the supervised and semi-supervised approaches. Using the full set of features and comparing the corresponding dependency parsing results with my previous work remains a direction for future work.

Dependency Parsing Algorithms

As in previous chapters, I use the dependency parsing algorithm shown in Figure 5.1 presented in Section 5.4, although Eisner's algorithm (Eisner, 1996) and the Spanning Tree algorithm (McDonald et al., 2005b) are also applicable.

8.7 Experimental Results

Given a convex approach to semi-supervised structured large margin training, and an efficient training algorithm for achieving a global optimum, I now investigate its effectiveness for dependency parsing. In particular, I investigate the accuracy of the results it produces. I applied the resulting algorithm to learn dependency parsers for both English and Chinese. Note that I need a different experimental setup from previous chapters because it is a semi-supervised approach.

8.7.1 Experimental Design

Since I use a semi-supervised approach, both labeled and unlabeled training data are needed. For experiment on English, I used the English Penn Treebank (PTB) as described in Section 4.1. The standard training set of PTB was spit into 2 parts: labeled training data—the first 30,000 sentences in section 2-21, and unlabeled training data—the remaining sentences in section 2-21. For Chinese, I experimented on the Penn Chinese Treebank 4.0 (CTB4) as described in Section 4.1. I also divided the standard training set into 2 parts: sentences in section 400-931 and sentences in section 1-270 are used as labeled and unlabeled data respectively. For both English and Chinese, I adopted the standard development and test sets throughout the literature.

As listed in Table 8.1 with greater detail, I experimented with sets of data with different sentence length: PTB-10/CTB4-10, PTB-15/CTB4-15, PTB-20/CTB4-20, CTB4-40 and CTB4, which contain sentences with up to 10, 15, 20, 40 and all words respectively. The

Data split		Training(labeled/unlabeled)	Development	Test
English (PTB)	PTB-10	3026 / 1016	163	270
	PTB-15	7303 / 2370	421	603
	PTB-20	12519 / 4003	725	1034
	Source	Sec. 02-21	Sec. 22	Sec. 23
Chinese (CTB4)	CTB4-10	642 / 347	61	40
	CTB4-15	1262 / 727	112	83
	CTB4-20	2038 / 1150	163	118
	CTB4-40	4400 / 2452	274	240
	CTB4	5314 / 2977	300	289
	Source	Sec. 400-931 / 1-270	Sec. 301-325	Sec. 271-300

Table 8.1: Size of experimental data (# of sentences)

reason we only did experiments with sentence length up to 20 words for English is due to the computational cost. We could train on the full Chinese corpus since CTB4 is smaller than PTB. Note that the optimization strategy we used here is much more efficient than the one we used in Chapter 6 before, where we could only train on sentences with no more than 10 words.

8.7.2 Results

Same as in previous chapters, I evaluate parsing accuracy by comparing the directed dependency links in the parser output against the directed links in the treebank. The parameters α and λ which appear in Equation (8.6) were tuned on the development set. Note that, during training, I only used the raw sentences of the unlabeled data. As shown in Table 8.2 and Table 8.3, for each data set, the semi-supervised approach achieves a significant improvement over the supervised approach in dependency parsing accuracy on both Chinese and English. These positive results are somewhat surprising since a very simple loss function was used on the unlabeled data. A key benefit of the approach is that a straightforward training algorithm can be used to obtain global solutions.

Although promising, the results shown in Table 8.2 and Table 8.3 are still not as good as the ones shown in Chapter 7 (Table 7.3). This is not unexpected, since I used a much smaller feature set here. Interestingly, Table 8.2 also shows that on Chinese (trained on CTB4-10), the accuracies from the supervised and the semi-supervised approaches are 83.0% and 84.5% respectively, which are much better than the results from Chapter 5 and Chapter 6 (as shown in Table 6.4). Note that supervised structured large margin approaches are used

Training data	Test sentence length	Supervised	Semi-supervised
CTB4-10	≤ 10	82.98	**84.50**
CTB4-15	≤ 10	84.80	**86.93**
	≤ 15	76.96	**80.79**
CTB4-20	≤ 10	84.50	**86.32**
	≤ 15	78.77	**80.57**
	≤ 20	74.89	**77.85**
CTB4-40	≤ 10	84.19	**85.71**
	≤ 15	78.03	**81.21**
	≤ 20	76.25	**77.79**
	≤ 40	68.17	**70.90**
CTB4	≤ 10	82.67	**84.80**
	≤ 15	77.92	**79.30**
	≤ 20	77.30	77.24
	≤ 40	70.11	**71.90**
	all	66.30	**67.35**

Table 8.2: Supervised and semi-supervised dependency parsing accuracy on Chinese (CTB4) (%)

Training data	Test sentence length	Supervised	Semi-supervised
PTB-10	≤ 10	87.77	**89.17**
PTB-15	≤ 10	88.06	**89.31**
	≤ 15	81.10	**83.37**
PTB-20	≤ 10	88.78	**90.61**
	≤ 15	83.00	**83.87**
	≤ 20	77.70	**79.09**

Table 8.3: Supervised and semi-supervised dependency parsing accuracy on English (PTB) (%)

in both Chapter 6 and Chapter 8, while the results are very different. The discrepancies are due to the different feature set used. Part-of-speech features are not used in Chapter 6. Although some additional features, such as distance and mutual information features, are used there, these features are not very helpful with short sentences on predicting dependency links. Therefore, on CTB4-10, with only word-pair and tag-pair features, the supervised structured large margin approach generates much better results than the one in Chapter 6.

The results presented in this chapter are not directly comparable with results shown in (McClosky et al., 2006a), since the parsing accuracy shown here is measured in terms of dependency relations while their results are f-score of the bracketings implied in the phrase structure.

8.8 Contributions

In this chapter, I have presented a novel algorithm for semi-supervised structured large margin training. Unlike previous proposed approaches, I introduce a convex objective for the semi-supervised learning algorithm by combining a convex structured SVM loss and a convex least square loss. This new semi-supervised algorithm is much more computationally efficient and can easily scale up. I have supported this hypothesis by applying the algorithm to the significant task of dependency parsing. The experimental results show that the proposed semi-supervised large margin training algorithm outperforms the supervised approach, without much additional computational cost.

8.9 Conclusion

I have addressed the problem of learning dependency parsers by using a novel semi-supervised training algorithm. Although the semi-supervised results are significantly better than the supervised counterpart, there remain many directions for future work. One obvious direction is to use the whole Penn Treebank as labeled data and use some other unannotated data source as unlabeled data for semi-supervised training. Another direction is to apply the semi-supervised idea to other natural language problems, such as machine translation, topic segmentation and chunking. In these areas, there are only limited annotated data available, therefore semi-supervised approaches are necessary for one to achieve better performance. The proposed semi-supervised approach can be easily applied to these tasks. Furthermore, as I mentioned before, a much richer feature set can be used in our algorithms to get better performance.

Chapter 9

Conclusions and Future Directions

9.1 Summary of Contributions

In this thesis, I investigated the problem of natural language parsing. Parsing accuracy has increased significantly due to the introduction of machine learning methods into this area in recent years. In this thesis I investigated many ideas for improving the state-of-the-art in this area. In particular, I presented four projects on dependency parsing in Chinese and English, based on different training algorithms: 1, a maximum likelihood estimation method augmented with similarity-based smoothing, where no POS tags or grammatical categories are needed; 2, an improved structured large margin training approach that uses local constraints and Laplacian regularization; 3, a simple training algorithm for dependency parsing that uses a simplified form of structured boosting to improve the training of a standard local link classifier; and 4, a novel semi-supervised training algorithm for learning dependency parsers.

All of these projects use advanced global machine learning algorithms to train accurate dependency parsers from treebank data efficiently. Consequently, I have been able to obtain state-of-the-art results in English, and surpass state-of-the-art in Chinese on dependency parsing.

9.2 Future Directions

Below I highlight some areas of future research this thesis suggests.

9.2.1 Variations of Structured Boosting

In Chapter 7, I have addressed the problem of learning dependency parsers by using a simple form of boosting to augment the training of standard local link predictors. Although the results obtained are very promising, there remain many directions for future work. One obvious direction is to investigate the effect of using alternative boosting algorithms, such as Adaboost M1, M2 (Freund and Schapire, 1997) and the logistic regression form of boosting described in (Collins et al., 2002). It would be interesting to investigate the theoretical nature of applying these algorithms to the structured boosting case shown in Chapter 7: under what circumstances do the algorithms converge, and what guarantees can be made about their performance?

9.2.2 Multilingual Dependency Parsing

I have applied my parsers to both English and Chinese in this thesis. Due to the rising importance of globalization and multilingualism, there is a need to build natural language parsers for a wider range of languages. A common characteristic in current existing parsers is that, the performance is very good on English, while the performance of the same parsers on other languages, such as Chinese, Czech, Danish or Arabic, still fall far short of expectations (McDonald and Pereira, 2006; Corston-Oliver et al., 2006). One advantage of the dependency parsing approaches I proposed is that I develop them from principled, general machine learning algorithms so they are not restricted to one specific language. Thus, all the techniques I have presented can be easily applied to other languages, if the corresponding treebanks are available.

9.2.3 Domain Adaptation

The generalization properties of most current statistical learning techniques are based on the assumption that the training data and the test data come from the same underlying probability distribution. Unfortunately, in many applications, this assumption is inaccurate. It is often the case that plentiful labeled data exists in one domain, but one desires a statistical model that performs well on another related, but not identical domain. Domain adaptation is an important, interesting, and challenging topic in natural language parsing. Many researchers have worked on adapting their parsers or part-of-speech taggers to a new domain (Blitzer et al., 2006; McClosky et al., 2006b), where there are few or no annotated resources available. I would like to investigate this subject and apply my parsers in other domains (e.g., biomedical data or web data) besides treebank data, to investigate the effectiveness and generality of my approaches.

9.2.4 A Unified View of My Current Work

I have discussed four pieces of my work in detail in Chapter 5, Chapter 6, Chapter 7, and Chapter 8. They all look very different approaches superficially, however, they are actually closely related by "scoring" formulation and, more specifically, by Equation (2.1), introduced in Section 2.4. In other words, they all compute a linear classifier.[1] The only differences among them are:

- What features are used?
- How are the parameters $\boldsymbol{\theta}$ estimated?

First, I would like to incorporate the dynamic features used in Chapter 5 into the large margin framework presented in Chapter 6 and Chapter 8. Then I would like to explore further ideas about useful features for parsing, such as using simple morphological features. I am also interested in the idea of using second-order features (various feature combinations). In fact, combined features have been proved to be useful in dependency parsing with support vector machines (Yamada and Matsumoto, 2003), and I have already obtained some preliminary results on generating useful feature combinations via boosting.

A general perspective I bring to my investigation is the desire to delineate the effects of domain engineering (choosing good features for representing and learning parsing models) from the general machine learning principles (training criteria, regularization and smoothing techniques) that permit good results. Therefore, I would like to consider combining all the projects I presented in previous chapters. That is, I would like to incorporate all the static features shown in Table 2.4.2, the dynamic features discussed in Chapter 5 and the second-order features as discussed above, into the training algorithms presented in Chapter 6, Chapter 7 or Chapter 8, to train a dependency parser globally. Then I would augment the training with the existing smoothing and regularization techniques (as described in Chapter 6, or new developed ones). I expect the resulting parser to have better performance than those I have presented in previous chapters.

[1]In general, for any probabilistic model, the product of probabilities can be converted to sums of scores in the log space, which makes the search identical to a score based discriminative model.

Bibliography

S. Abney, R. Schapire, and Y. Singer. 1999. Boosting applied to tagging and PP attachment. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

S. Abney. 2004. Understanding the yarowsky algorithm. *Computational Linguistics*, 30(3):365–395.

J. Allen. 1987. *Natural Language Understanding.* Benjamin/Cummings Publishing.

Y. Altun, I. Tsochantaridis, and T. Hofmann. 2003. Hidden Markov support vector machines. In *Proceedings of International Conference on Machine Learning.*

Y. Altun, D. McAllester, and M. Belkin. 2005. Maximum margin semi-supervised learning for structured variables. In *Proceedings of Advances in Neural Information Processing Systems 18.*

R. Bellman. 1957. *Dynamic Programming.* Princeton University Press.

K. Bennett and A. Demiriz. 1998. Semi-supervised support vector machines. In *Proceedings of Advances in Neural Information Processing Systems 11.*

S. Bergsma and D. Lin. 2006. Bootstrapping path-based pronoun resolution. In *Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics.*

S. Bergsma and Q. Wang. 2007. Learning noun phrase query segmentation. In *Proceedings of the joint Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning*, pages 819–826.

D. M. Bikel and D. Chiang. 2000. Two statistical parsing models applied to the chinese treebank. In *Proceedings of the Second Chinese Language Processing Workshop*, Hong Kong.

D. Bikel. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4).

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *DARPA Speech and Natural Language Workshop.*

E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

J. Blitzer, R. McDonald, and F. Pereira. 2006. Doman adaptation with structural correspondence learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

S. Buchholz and E. Marsi. 2006. The Conference On Computational Natural Language Learning-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*.

G. Carroll and E. Charniak. 1992. Two experiments on learning probabilistic dependency grammars from corpora. Technical Report CS-92-16, Brown University.

O. Chapelle and A. Zien. 2005. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Inteligence and Statistics*.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

E. Charniak. 1996. Tree-bank grammars. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pages 1031–1036.

E. Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pages 598–603.

E. Charniak. 2000. A maximum entropy inspired parser. In *Proceedings of North American Annual Meeting of the Association for Computational Linguistics*, pages 132–139.

C. Cherry and D. Lin. 2003. A probability model to improve word alignment. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 88–95.

Y. Chu and T. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

S. Clark and J. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

S. Clark and J. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

S. Clark, J. Hockenmaier, and M. Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

M. Collins, R. Schapire, and Y. Singer. 2002. Logistic regression, Adaboost and Bregman distances. *Machine Learning*, 48.

M. Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 184–191.

M. Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

M. Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of International Conference on Machine Learning*.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

A. Copestake and D. Flickinger. 2000. An open-source grammar development environment and broad-coverage English grammar using hpsg. In *Proceedings of the Second Conference on Language Resources and Evaluation*.

S. Corston-Oliver, A. Aue, K. Duh, and E. Ringger. 2006. Multilingual dependency parsing using Bayes' point machines. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

K. Crammer and Y. Singer. 2001. On the algorithmic interpretation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

I. Dagan, L. Lee, and F. Pereira. 1999. Similarity-based models of word cooccurrence probabilities. *Machine Learning*, 34(1-3):43–69.

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

R. Duda, P. Hart, and D. Stork. 2000. *Pattern Classification*. Wiley, second edition.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the International Conference on Computational Linguistics*.

H. Fox. 2002. Phrasal cohesion and statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 304–311.

Y. Freund and R. Schapire. 1996. Experiments with a new boosting algorithm. In *Proceedings of International Conference on Machine Learning*.

Y. Freund and R. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Computer and System Sciences*, 55(1):119–139.

H. Gaifman. 1965. Dependency systems and phrase structure systems. *Information and Control*, 8:304–337.

D. Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

J. Goodman. 2004. Exponential priors for maximum entropy models. In *Proceedings of North American the Annual Meeting of the Association for Computational Linguistics.*

D. Graff and K. Chen. 2003. *Chinese Gigaword.* Linguistic Data Consortium.

D. Graff. 2003. *English Gigaword.* Linguistic Data Consortium.

G. Grefenstette. 1994. Corpus-derived first, second and third-order word affinities. In *Proceedings of Euralex.*

G. Haffari and A. Sarkar. 2007. Analysis of semi-supervised learning with the yarowsky algorithm. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence.*

J. Hajic. 1998. Building a syntactically annotated corpus: The Prague dependency treebank. In *Issues of Valency and Meaning.*

Z. Harris. 1968. *Mathematical Structures of Language.* Wiley, New York.

M. Haruno, S. Shirai, and Y. Ooyama. 1999. Using decision trees to construct a practical parser. *Machine Learning*, 34:131–149.

T. Hastie, R. Tibshirani, and J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.

T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. 2004. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415.

D. Hindle. 1990. Noun classification from predicate-argument structures. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 268–275.

F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Patnaparkhi, and S. Roukos. 1994. Decision tree parsing using a hidden derivational model. In *Proceedings of the Human Language Technology Workshop*, pages 272–277.

F. Jiao, S. Wang, C. Lee, R. Greiner, and D. Schuurmans. 2006. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics.*

M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler. 1999. Estimators for stochastic “unification-based” grammars”. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

A. Joshi, L. Levy, and M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and Systems Science*, 21(2):136–163.

D. Jurafsky and J. Martin. 2000. *Speech and Language Processing*. Prentice Hall.

D. Klein and C. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

D. Klein and C. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

D. Klein and C. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedingsof the Annual Meeting of the Association for Computational Linguistics*.

J. Lafferty, D. Sleator, and D. Temperley. 1992. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the Association for the Advancement of Artificial Intelligence Fall Symposium on Probabilistic Approaches to Natural Language*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of International Conference on Machine Learning*.

R. Levy and C. D. Manning. 2003. Is it harder to parse chinese, or the chinese treebank? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics 2003*, Sapporo, Hokkaido, Japan.

D. Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1420–1425.

D. Lin. 1998. Automatic retrieval and clustering of similar words. In *Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics*, pages 768–774.

D. Magerman and M. Marcus. 1990. Parsing a natural language using mutual information statistics. In *National Conference on Artificial Intelligence*, pages 984–989.

D. Magerman. 1995. Statistical decision-tree model for parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 276–283.

G. S. Mann and A. McCallum. 2007. Simple, robust, scalable semi-supervised learning via expectation regularization. In *Proceedings of International Conference on Machine Learning*.

C. Manning and H. Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

A. McCallum and W. Li. 2003. Early results for named-entity extraction with conditional random fields. In *Proceedings of the Conference on Computational Natural Language Learning*.

A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy Markov methods for information extraction and segmentation. In *Proceedings of International Conference on Machine Learning*.

D. McClosky, E. Charniak, and M. Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology: the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

D. McClosky, E. Charniak, and M. Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of European Chapter of the Annual Meeting of the Association for Computational Linguistics*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technologies and Conference on Empirical Methods in Natural Language Processing*.

Y. Miyao and J. Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*.

J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings the International Conference on Computational Linguistics*.

M. Palmer *et al.* 2004. *Chinese Treebank 4.0.* Linguistic Data Consortium.

M. Palmer *et al.* 2005. *Chinese Treebank 5.0.* Linguistic Data Consortium.

F. Pereira and Y. Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 128–135.

F. Pereira, N. Tishby, and L. Lee. 1993. Distributional clustering of English words. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 183–190.

C. Pinchak and D. Lin. 2006. A probabilistic answer type model. In *Proceedings of the European Chapter of the Annual Meeting of the Association for Computational Linguistics*.

C. Pollard and I. Sag, editors. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago.

A. Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

A. Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

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

R. Schapire and Y. Singer. 1999. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37.

R. Schapire and Y. Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

R. Schapire. 2001. The boosting approach to machine learning: An overview.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the Human Language Technology Conference and the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

L. Shen, A. Sarkar, and A. Joshi. 2003. Using LTAG based features in parse reranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

N. Smith and J. Eisner. 2005a. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

N. Smith and J. Eisner. 2005b. Guiding unsupervised grammar induction using contrastive estimation. In *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on Grammatical Inference Applications*.

N. Smith and J. Eisner. 2006. Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of the International Conference on Computational Linguistics and the Annual Meeting of the Association for Computational Linguistics*.

M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proceedings of the European Chapter of the Annual Meeting of the Association for Computational Linguistics*, pages 331–338.

M. Steedman, editor. 2000. *The Syntactic Process*. MIT Press.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *Proceedings of Advances in Neural Information Processing Systems 16*.

B. Taskar, V. Chatalbasher, and D. Koller. 2004a. Learning associative Markov networks. In *Proceedings International Conference on Machine Learning*.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004b. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

K. Toutanova, C. Manning, S. Shieber, D. Flickinger, and S. Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *First Workshop on Treebanks and Linguistic Theories*.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Human Language Technology Conference and the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of International Conference on Machine Learning*.

V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer Verlag, New York.

K. Vijay-Shanker. 1993. Using descriptions of trees in a Tree Adjoining Grammar. *Computational Linguistics*, 18(4).

Q. Wang and D. Schuurmans. 2005. Improved estimation for unsupervised part-of-speech tagging. In *Proceedings of the 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering*, pages 219–224.

Q. Wang, D. Schuurmans, and D. Lin. 2005. Strictly lexical dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies*, pages 152–159.

Q. Wang, C. Cherry, D. Lizotte, and D. Schuurmans. 2006. Improved large margin dependency parsing via local constraints and Laplacian regularization. In *Proceedings of the Conference on Computational Natural Language Learning*, pages 21–28.

Q. Wang, D. Lin, and D. Schuurmans. 2007. Simple training of dependency parsers via structured boosting. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1756–1762.

Q. Wang, D. Lin, and D. Schuurmans. 2008. Semi-supervised convex training for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

F. Xia and M. Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the Human Language Technology Conference*.

L. Xu and D. Schuurmans. 2005. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings the Association for the Advancement of Artificial Intelligence*.

L. Xu, D. Wilkinson, F. Southey, and D. Schuurmans. 2006. Discriminative unsupervised learning of structured predictors. In *Proceedings of International Conference on Machine Learning*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the International Workshop on Parsing Technologies*.

R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. 2000. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of Applied Natural Language Processing and the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts.

D. Yuret. 1998. *Discovery of Linguistic Relations Using Lexical Attraction*. Ph.D. thesis, MIT.

X. Zhu, Z. Ghahramani, and J. Lafferty. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of International Conference on Machine Learning*.

X. Zhu. 2005. Semi-supervised learning literature survey. Technical report, Computer Sciences, University of Wisconsin-Madison.