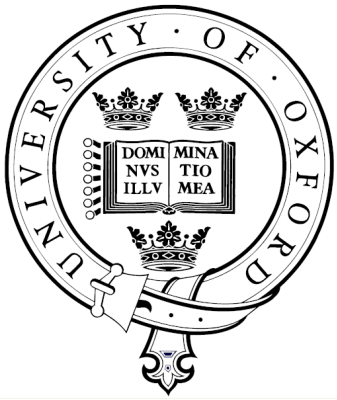
DISCRIMINATIVE LEARNING APPROACHES FOR THE STATISTICAL PROCESSING OF CHINESE



Yue Zhang

Oxford University Computing Laboratory

October 2009

Acknowledgments

I am indebted to many people who gave me great help and support during my DPhil studies. First I would like to thank my supervisor, Stephen Clark, who allowed me complete freedom in pursuing research topics while providing guidance with insightful ideas and discussions. I am particularly grateful for the encouragements and opportunities that Stephen gave me to meet other researchers and discuss with them, including the chances to participate in the 2009 Johns Hopkins summer workshop, and to present and discuss my work in Cambridge.

Thanks also go to Stephen Pulman, who gave me my first lecture about computational linguistics, and recent inspirations in the use of linguistic knowledge, John Carroll, my external examiner, who gave many detailed comments and very helpful discussions, and Mark Steedman, who invited me to talk about my work in Edinburgh, and discuss with students and researchers in the group.

I have benefited from input from people inside and outside Oxford about research topics and life in doctoral studies. I would like to mention my fellow students Karo Moilanen, Brian Harrington, and Rachele de Felice from Oxford, and people ouside Oxford who were kind enough to spend some time discussing with me or giving me help, including Mengqiu Wang, Adam Lopez, Emily Thomforde, Prachya Boonkwan, Thomas Kwiatkowski, Miles Osborne, Aurelie Herbelot, Johanna Geiss, Diarmuid O'Seaghdha, Joakim Nivre and Hwee-Tou Ng.

I was funded by the ORS scholarship and Clarendon Fund; Oxford University Computing Laboratory and Mansfield College gave me support for presenting at conferences.

Finally but most importantly, I would like to thank my parents for their constant love and support all these years, and my wife, Qian, for her immense love and care, which enabled me to finish my DPhil within three years.

Abstract

We study discriminative approaches to statistical Chinese processing, including word segmentation, POS-tagging and parsing with constituent and dependency grammars.

As the main approach of this thesis, we use a global linear model, trained by the generalized perceptron algorithm, together with beam-search decoding, to build our statistical systems. The combination of perceptron training and beam-search decoding leads to highly competitive accuracy and efficiency for all the tasks we investigate.

For word segmentation, we propose a word-based approach that achieves competitive accuracy to the best character-based systems, without mapping segmentation into a sequence labeling problem. For POS-tagging, we built a joint system that performs word segmentation and tagging simultaneously, showing that it improves the accuracy over the traditional pipeline approach by enabling information interaction and avoiding error propagation. For constituent parsing, we develop our model based on a shift-reduce parsing algorithm, which currently provides state-of-the-art performance for Chinese, using a global model and beam-search, showing that it achieves comparable accuracy to the best scores in the literature. For dependency parsing, we combine the two predominant statistical methods into a single system using a global linear model, and achieve the current best accuracy for Chinese.

One main advantage of the discriminative approach to statistical NLP is the freedom to define features that represent contextual information. For all the problems studied in this thesis, we achieve state-of-the-art accuracies by utilizing a wide range of information in a discriminative model. We conclude that the discriminative method is a competitive choice for the statistical processing of the Chinese language.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Natural language processing (NLP), or *computational linguistics*, is the study of artificial intelligence concerning the processing of human languages. It facilitates communication not only between humans and computers, but also between humans (e.g. via automatic machine translation between different languages). Typical NLP problems include grammatical analysis of natural languages, and applications such as automatic question answering and machine translation.

NLP has been a challenging field in computer science. The earliest NLP systems were constructed by using rules that encode linguistic knowledge. However, natural languages are so dynamic and ambiguous that no set of linguistic rules is enough to cover their usage. With the advance of computing power, statistical methods that use machine learning have gained increased attention, and have become the dominant approach in the computational linguistics literature since the mid-1990s. By collecting large amounts of statistical information from hand-crafted text corpora, statistical methods have been shown to outperform rule-based systems in almost every NLP application.

Compared to rule-based systems, statistical models are often less dependent on a specific language. In a recent CoNLL shared task (Buchholz and Marsi, 2006), for example, each participating parser model is tested with 13 different languages. Until now, the majority of NLP literature has been focused on the English language. Other natural languages are often processed by methods that have been successfully applied to English.

Chinese is one of the languages that has gained the most attention in NLP. Many Chinese NLP

problems are investigated by applying successful methods for English. For example, English POS-tagging models and syntactic parsing algorithms have been applied to Chinese. Even for Chinese word segmentation, which does not have a counterpart for English, the dominant approach has been sequence labeling, which is also used by English POS-taggers.

There are structural similarities between Chinese and English, but the two languages are also different in many aspects. Recent research has suggested that the most effective methods to process Chinese are not necessarily the most effective for English. For example, though giving lower performance for English, shift-reduce parsing techniques have given comparatively higher accuracies for Chinese (Wang et al., 2006; Duan et al., 2007).

This thesis focuses on the automatic processing of the Chinese language. The problems studied in this thesis include *word segmentation*, POS-*tagging* and *parsing* in constituent and dependency formalisms. In particular, we show that instead of being mapped into a character labeling problem, word segmentation can be performed using a direct word-based model. Moreover, we illustrate that as a language-specific problem, word segmentation can be performed with POS-tagging as a single task. By utilizing POS information to help segmentation, joint word segmentation and POS-tagging achieves higher accuracy than the standard pipeline approach. Finally, we build both a constituent and a dependency parser, which confirm that the shift-reduce approach is a competitive choice for parsing Chinese.

We use discriminative machine learning models to build our statistical systems. An advantage of these models is that they allow the use of arbitrary and overlapping features, trained in a consistent process, to represent the statistics of syntactic structures. We apply beam-search to most of the problems studied in this thesis, finding that it is a reasonable choice that gives linear running time complexity and competitive output accuracy.

1.1 About the Chinese language

Chinese is one of the most ancient languages in the world. Among the earliest records of the language are *characters* carved on ancient relics such as items of pottery, dating back to over four thousand years ago. These characters are identified as stylized pictures of physical objects, and are therefore called pictographs. Having gone through thousands of years of development, most

Chinese characters are now considerably different from their ancient origins. However, the use of characters as the basis of the Chinese writing system is preserved. Chinese is therefore recognized as an ideographical language, in contrast to alphabetical languages.

Apart from the shapes of characters, syntactic rules – the rules by which characters are combined to express meaning – have also evolved. An example is the appearance of *words*. In the ancient scripts, each character alone carries a meaning. In time, however, two or more characters began to be used together regularly to express a particular meaning. These character combinations can be taken as words, because they form inseparable units in the sentences. Consequently, characters can be seen as morphemes inside words (Sproat et al., 1996). Most words in Modern Chinese consist of one or two characters, although there can be very long words such as numbers or proper names. According to the experimental data of this thesis, the typical average word length is roughly 1.6.

Though words exist in Chinese sentences, there are no explicit delimiters for word boundaries. Therefore, the identification of words is an important part of the understanding process of a Chinese sentence. This process is so easy and natural to a human reader that no deliberate effort is needed for it. However, for a computer it is found to be as difficult as many other problems in NLP, largely because of the intrinsic ambiguities of natural languages. Table 1.1 gives a number of character sequences. When put in different contexts, they may represent different words and meanings. Such ambiguities can normally be resolved by looking at the neighboring characters or words in the sentence, but sometimes it is necessary to refer to a larger context outside the sentence, or even some domain and world knowledge. In short, correct word identification requires necessary understanding of the context and the world.

Most linguistic theories of Chinese syntactic analysis are based on words. However, for a few types of words, there is no universal agreement on the delimitation. For example, some functional characters (like "們") can be appended to nouns to give plural meaning. Such a character can be taken as a suffix morpheme in a plural noun, or a stand-alone functional word.[1] Neither choice is clearly more correct than the other. For another example, people still hold different views on whether the surname and given name should be separated or taken as a single word. Such delimitation ambiguity only occurs in a small proportion of the words, and it does not appear to be

[1]The two cases occur in the Hong Kong City University Corpus and the Peking University Corpus in the First International Chinese Word Segmentation Bakeoff (Sproat and Emerson, 2003), respectively.

Number	Character sequence	Possible interpretations
1	的	语言学(linguistics) 的(about) 书(book) 目的(goal) 明确(clear)
2	已经	已经(already) 检查(check) 过(done) 已(already) 经过(gone through) 检查(check)
3	这里面	这里(here) 面(flour) 和(and) 米(rice) 很(quite) 贵(expensive) 这(here) 里面(inside) 很(quite) 冷(cold)
4	以前天下	以前(before) 天下(the world) 太平(peaceful) 以(take) 前天(two days ago) 下雨(rain) 为例(for example)
5	中国外企业	中国(China) 外企(foreign company) 业务(business) 其中(among which) 国外(foreign) 企业(company)
6	洽淡会很成功	洽淡(discussion) 会(will) 很(very) 成功(successful) 洽淡会(discussion meeting) 很(very) 成功(successful)

Table 1.1: Examples of ambiguous character sequences

of much importance linguistically. However, it can add to the complexity of word identification computationally.

The above facts showed some uniqueness and difficulties of Chinese NLP from the perspective of word formation and boundaries. Once a Chinese sentence is turned into words, it can be analyzed in the same way as the alphabetical languages. Many formal grammars and computational models can be used to analyze word-segmented Chinese sentences, reflecting a degree of general similarity between the syntax of Chinese and other languages. However, the Chinese grammar is also unique in many ways. For example, below are some important differences between the syntax of Chinese and English (and possibly other alphabetical languages):

1. The Chinese language has unique syntactic ambiguities. The most important example is perhaps that the relationship between POS and grammatical roles is flexible. For example, a Chinese adjective can be used as the predicate, attribute, adverbial modifier or even the subject and object of a sentence, while a Chinese noun can also act as the predicate in a sentence. Besides POS, word order is also a weaker indicator of grammatical roles. Here are four example sentences:

(a) "我 (I) 经常 (often) 吃 (eat) 面条 (noodles) ";

(b) "我 (I) 经常 (often) 吃 (eat) 饭馆 (restaurant) ";

(c) "他们 (they) 洗 (wash) 衣服 (clothes) 了 (past tense)";

(d) "衣服 (clothes) 洗 (wash) 干净 (clean) 了 (past tense)".

The word "面条 (noodle)" in sentence (a) and the word "饭馆 (restaurant)" in (b) are in the same

position, but their roles are different. "面条 (noodle)" is the object of "吃 (eat)", while "饭馆 (restaurant)" is the argument referring to the location (at restaurants). Similarly, the object "衣服 (clothes)" in sentence (c) and the adjunct "干净 (clean)" in sentence (d) are in parallel positions. Moreover, the subject "他们 (they) " in sentence (c) and the object "衣服 (clothes)" in sentence (d) are also in parallel positions. Though Chinese sentences are usually in the "subject" + "predicate" + "object" order, the object can sometimes be moved to the front of the sentence. "衣服 (clothes)" in sentence (d) is such a case, where it is highlighted as the topic. Another example of object movement is the sentence "你 (you) 我 (I) 还 (auxiliary) 不 (not) 了解 (know) ? ", where the object "你 (you)" is put to the front.

On the other hand, though very frequent for English and many other languages, attachment ambiguities, such as the prepositional phrase attachment ambiguity between the sentences "He eats Pizza with mushrooms." and "He eats Pizza with a fork.", are not common for Chinese.

2. Omissions are comparatively frequent. The subject or object of a sentence can often be omitted, provided that there is no extra ambiguity. The omitted subjects or objects are also called zero pronouns. Verbs can also be omitted. For example, in the sentence "他 (he) 高中 (middle school)，我 (I) 本科 (college)", the omitted predicate can be "has the education level of". In the sentence "他 (he) 大卫 (David)，我 (I) 马莉 (Mary)", however, the omitted verb is "be". Characters in words can also be omitted, resulting in abbreviated words. Abbreviation is common in Chinese, especially for proper names. For example, "中" can often be used for "中国 (China)" in a clear context. Similarly, "港" is used conventionally for "香港 (Hong Kong)", and "英" for "英国 (The UK)".

3. Chinese words do not have morphology. The plural meaning, and tense and the passive voice, are all expressed by special words. On the other hand, characters as the components strongly indicate the meaning of words. The meaning of an unknown Chinese word can often be guessed by its characters. In comparison, the meaning of most English words cannot be decided by the spelling.[2] The only phenomenon that is related to morphology is reduplicated words. Reduplication is quite common in Chinese; it applies to nouns, verbs, adjectives, adverbs, numbers, measure words

[2]In contrast, the pronunciation of English words can often be decided by their spelling, while that of Chinese words cannot. This once led to an interesting phenomenon in a historical period: though written Chinese was understandable in many countries across Asia, people from different areas were unable to communicate orally in the language.

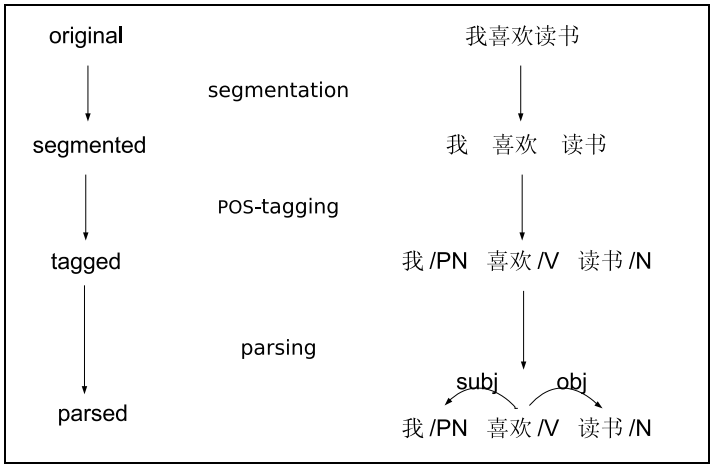


Figure 1.1: An illustration of the problems studied in this thesis

and other POS. There are various patterns of reduplication, including AA ("好好学习", study hard), ABB ("一次次", time and time again), ABAB ("尝试尝试", have a try) and so on. They are used to express specific degrees of illocutionary force (e.g. highlight) or feelings (e.g. happiness).

In summary, Chinese sentences are written as character sequences, and a preliminary step for the syntactic analysis is the recognition of words. After being turned into word sequences, Chinese sentences share grammatical similarities with English, but also differ in morphological and syntactical structures.

1.2 The scope of this thesis

In this thesis we study algorithms for the syntactic analysis of the Chinese language. Given a Chinese sentence, such algorithms produce its syntactic structure automatically. Figure 1.1 gives a simple illustration of the main problems addressed. The input sentence, which is a continuous sequence of characters, is analyzed by three processes. In the first step, it is segmented into a sequence of words. This process is called *word segmentation*, and is unique for Chinese and other character-based languages such as Japanese and Thai. The second step is called POS-*tagging*, in which a POS-tag is assigned to each word from the segmented sentence. The last step is called *parsing*, in which the syntactic structure of the POS-tagged sentence is given according to a particular form of grammar. Figure 1.1 shows the parsed structure according to dependency grammar. In this thesis, parsing with

a context free grammar is also studied. Though focusing on Chinese, the statistical parsing models that we studied are language-independent. We also applied our dependency parser to English.

1.3 The approach

We use the statistical approach for all the problems covered by this thesis. A statistical system typically finds the output by search: given an input, different candidate outputs are generated and scored, with the highest scored one chosen for output. The search process is also called the *decoding* process. We use beam-search as the main algorithm for decoding.

During decoding, different candidate outputs are compared according to their scores, which are given by a *statistical model*. The score of a candidate output is usually computed by extracting *feature*s, which represent the occurrences of a particular pattern in the output. The feature values are passed to the statistical model to calculate the score. We use a *discriminative model* for all the problems studied in this thesis. In contrast to a *generative model*, which defines a stochastic generation process for output structures and scores candidate outputs by the probability of them being generated in the process, a discriminative model does not give a generative story, and scores candidate outputs only for comparison. One of the main advantages of discriminative models is the freedom of defining arbitrary features without making independence assumptions. Discriminative methods have been shown to be competitive compared to generative models for most NLP problems.

Machine learning methods are used to define a statistical model, so that it gives a higher score to a more correct candidate output. A supervised *learning algorithm* collects statistical data from manually annotated text, setting parameters of the statistical model by using these data. We chose the generalized perceptron algorithm as the main discriminative learning method for this thesis.

More details about the background of the statistical approaches used in this thesis are given in Chapter 2.

1.4 Contributions

For word segmentation, we propose a word-based model that enables the use of arbitrary features, including word-based features. Compared to the standard character-based approach, which works

by mapping the segmentation problem into a sequence labeling problem, our word-based approach is a direct solution to the segmentation problem. We construct a word segmentor using a word-based discriminative model, trained by the generalized perceptron algorithm, together with beam-search decoding, and show that it achieves competitive accuracy with the best recorded in the literature. Our word-based segmentation model does not impose restrictions on the learning or decoding algorithms. It can be seen as a superset of the character-based models. The details of the word-based segmentation model are given in Chapter 3.

Word segmentation is the first step among the three in Figure 1.1, and POS-tagging is dependent on segmented input. This is the traditional, pipelined approach for Chinese POS-tagging. However, this approach has two disadvantages. First, POS-information, which is potentially useful for segmentation, is not used in the word segmentation step. Second, segmentation errors will propagate to the POS-tagging step. One solution is to treat segmentation and POS-tagging as a single task, where the two steps are performed simultaneously, and POS information is used to improve segmentation. We construct a joint segmentor and POS-tagger and a pipelined baseline using identical feature templates, and show that the joint system outperforms the baseline in both segmentation accuracy and the overall segmentation and tagging accuracy. We use global discriminative models for both the joint system and the pipelined baseline, trained by the averaged perceptron algorithm. The joint word segmentation and POS-tagging system is described in detail in Chapter 4.

Parsing can be performed under different grammar formalisms, and in this thesis we study constituent-based and dependency-based parsing. For constituent parsing, state-of-the-art accuracies have been achieved previously by a deterministic shift-reduce parsing model on parsing the Chinese Treebank 2 data. We propose a global discriminative model based on the shift-reduce constituent parsing process, combined with a beam-search decoder, obtaining competitive accuracies on CTB2. We also report the performance of the parser on CTB5 data, obtaining the highest scores in the literature for a dependency-based evaluation. The details of our constituent parser are given in Chapter 5.

For dependency parsing, the two dominant approaches are graph-based and transition-based. Given an input sentence, graph-based dependency parsers find the output by scoring each possible parse graph for the input, scoring each of them and choosing the highest scored candidate as the

output parse. Because the number of possible parse graphs for a particular sentence is exponential in the sentence length, graph-based parsers often use dynamic programming to solve the search problem in polynomial time. In contrast, transition-based dependency parsers build the output by using a stack and set of transition actions, such as "shift" and "reduce". Greedy local search is often applied to transition-based parsers for fast deterministic decoding.

Beam-search lies between dynamic programming and greedy local search in terms of accuracy and running speed. Compared to local search, it is global. But compared to dynamic programming, which is exact inference, it is approximate. Unlike dynamic programming, it does not have the "overlapping subproblems" and "optimal substructure" restrictions on the search problem, and therefore does not impose limitations on the form of features. We use beam-search to construct a graph-based and a transition-based parser, showing that beam-search is a competitive choice for both graph-based and transition-based dependency parsing. Moreover, we show that by using beam-search, it is possible to combine the graph-based and the transition-based methods into a single system, which outperforms the graph-based and transition-based systems alone. While giving the highest reported dependency parsing score for Chinese, our combined parser is also applied to English and shows competitive accuracies. Chapter 6 describes our work on dependency parsers.

Chapter 2

Background

This chapter gives background on statistical models and machine learning algorithms used in the thesis. Background on word-segmentation, POS-tagging, phrase-structure parsing and dependency parsing is given in the corresponding chapters.

NLP problems can be divided into two types in general: *classification* problems and *structural prediction* problems. In a classification problem, the output is a single discrete value. Examples of classification problems in NLP include text classification, which maps an input document to a class such as "news" or "science"; spam filtering, which maps an input document to one of the two values "spam" or "non-spam"; and a type of sentiment analysis which maps an input (normally sentence or document) to one of the three values "positive", "negative" and "neutral". In a structural prediction problem, the output is a structure, which can consist of a number of inter-related labels, or a more complex structure such as a parse tree. All the problems studied in this thesis (i.e. word segmentation, POS-tagging and parsing) are structural prediction problems. For word segmentation, the output is essentially a sequence of binary decisions on whether to separate two consecutive characters in the input. With an input sentence of length n, the output contains $n-1$ inter-dependent decisions. For POS-tagging, a POS-tag is assigned to each word from the segmented input sentence, and therefore the output is a sequence of inter-dependent labels. For parsing, the output is usually a parse tree structure, with connected nodes that depend on each other. Another example of structural prediction is machine translation, for which the output is a natural language sentence.

An early statistical model for NLP is the *generative probabilistic model*, which regards the mapping between the input and the output as a stochastic process, where different outputs can be generated given an input. According to a generative probabilistic model, the output O for an arbitrary input I should be the most probable output from the stochastic process, which can be written as $O = \arg\max_{O'} P(O', I)$ (the output is the one among all possibilities that has the highest joint probability with the input). The probabilities are estimated according to statistical information collected over a large amount of data, usually by the maximum likelihood principle.

For the structural prediction problem, the output O can contain many inter-dependent elements, which make the estimation of $P(O, I)$ difficult due to data sparseness (i.e. lack of statistical information for complex data). A typical solution to this problem by a generative model is to define a series of independent steps to generate different elements in the complex output, and then make possible simplifications to each generation step using independence assumptions. For example, denoting the elements in O as $o_1, o_2, .., o_n$, $P(O, I)$ can be written as $P(o_1, o_2, .., o_n, I)$. Suppose that the input I is simple; according to the probabilistic chain rule, a generation process can be defined as: $P(o_1, o_2, .., o_n, I) = P(I)P(o_1|I)P(o_2|o_1, I)..P(o_n|o_1, .., o_{n-1}, I)$. The factor probabilities $P(o_k|o_1, .., o_{k-1}, I)$ can then be simplified by making independence assumptions. For example, assuming that o_k is dependent only on o_{k-1} $(k > 1)$, we have $P(o_k|o_1, .., o_{k-1}, I) = P(o_k|o_{k-1})$, and $P(o_1, o_2, .., o_n, I) = P(I)P(o_1|I)P(o_2|o_1)..P(o_n|o_{n-1})$, where each factor is much easier to estimate. A typical generative model is the hidden Markov model. Generative models have also been used in seminal work in statistical parsing (Collins, 1997).

A potential disadvantage of generative models is the need for independence assumptions, which can lead to over-simplification of NLP tasks. Moreover, generative stories are not always obvious to find. It is more straightforward to model statistical information that influences the probability of a complex output directly. *Discriminative probabilistic models* address this problem by computing the conditional probability of the output O given the input I directly, often using the maximum entropy principle. Here the prediction problem can be written as $O = \arg\max_{O'} P(O'|I)$ (the output has the highest conditional probability among all possibilities given the input). Defining important statistical information as *features*, a discriminative model estimates the probability $P(O|I)$ directly according to the feature restrictions. The advantage of this method is the freedom to define arbitrary features

without making independence assumptions. An important example of a probabilistic discriminative model is the conditional random field (Lafferty et al., 2001), which has been shown to be more effective than the generative hidden Markov model for various NLP problems. A disadvantage of the conditional random field is the comparatively high time complexity for training.

While probabilistic models are still widely used, non-probabilistic models have been applied to NLP recently. All non-probabilistic models studied in this thesis are discriminative models, which maximize the score difference between correct structures and possible incorrect structures. The difference between discriminative models, regardless of their being probabilistic or non-probabilistic, include the output accuracy on particular problems and datasets, and the time complexity for training. We chose a global linear model trained by the generalized perceptron (Collins, 2002) as the main discriminative approach for this thesis. Although a comparatively simple and fast learning algorithm, the generalized perceptron often gives comparable accuracy to more complex and slower learning algorithms.

Besides the statistical model and the learning algorithm, the decoding search algorithm also plays an important role in an NLP system for structural prediction. The choice of decoder not only affects the output accuracy, but is also important for the system's efficiency. It is common for an NLP problem to have an exponentially large search space, making decoding efficiency an important issue. A common solution with no accuracy loss and reasonable efficiency is dynamic programming, which enables exact decoding with an exponential search space in polynomial time. However, the requirement that the search problem exhibits the properties of overlapping subproblems and optimal substructure, which is necessary for dynamic programming, also limits the features that can be used in the model. Moreover, because the generalized perceptron learning algorithm that we use is based on decoding, the speed of the decoder also influences the learning efficiency. In this thesis, we chose beam-search, which is an approximate search typically with linear time complexity, as the main decoding algorithm. For our word segmentation system in Chapter 3, beam-search gives competitive accuracy compared to dynamic programming approaches. For the joint segmentation and tagging problem in Chapter 4, and the combined dependency parser in Chapter 6, where dynamic programming cannot achieve practical running speed, beam-search gives reasonable running speed and output accuracy.

2.1 A review of statistical models for NLP

In this section we review important statistical models and learning methods for NLP, placing the perceptron algorithm into the category of discriminative, non-probabilistic approaches.

2.1.1 Probabilistic models

The earliest statistical models for NLP, *probabilistic* models compare candidate outputs by their probabilities, and take the most probable candidate as the predicted output.

The probability of an output given an input is estimated by statistical information collected from training data. Because of the sparseness of natural language data, the probability of a complex output must be broken into components that are easier to estimate. All probabilistic models can be seen as methods to reduce data sparseness. Generative models compute the probability of complex structures by using the probability chain rule and independence assumptions, and estimate the probability of simple structures by using the maximum likelihood principle, while discriminative models define a set of features as constraints and derive the conditional output probability by constrained maximization, using the maximum entropy principle. A typical generative model used in NLP is the hidden Markov model (HMM), while the most important probabilistic discriminative model is the conditional random field (CRF).

The input I of an NLP problem is typically a complex structure, and Bayes rule is often used to transform the computation of the conditional probability $P(O|I)$ into the estimation of $P(I|O)$ or $P(O, I)$, for which the data sparseness of I can be addressed.[1]

Two important machine learning principles for probabilistic statistical models

Maximum likelihood estimation (MLE) is a general probability estimation principle, which is used by many probabilistic models for NLP. The underlying principle for MLE is to find the distribution P

[1] Another common technique to address data sparseness is smoothing, which is often used with other techiques such as the generative approach or the Bayes method. For example, smoothing can be used to further reduce the sparseness of the probability of simple structures in generative models. However, because our main approach is a non-probabilistic model, we are concerned only with the model's aspect for the probabilistic methods, and therefore do not study smoothing methods in this thesis.

that maximizes the likelihood, or probability of training examples:

$$P = \arg\max_{P'} P(training|P')$$

For a binomial distribution, where each training example is an independent binary value, MLE has a very intuitive conclusion: the probability of a random example being 0/1 is equal to the relative frequency of 0/1 in the training data. For example, given a set of training data with S examples, from which the number of examples having the value b ($b \in \{0, 1\}$) being $Count(b)$, the probability of a random example having the value b can be estimated by:

$$P(b) = \frac{Count(b)}{S} \tag{2.1}$$

Now we give a derivation for Equation 2.1. Because the values of training examples are independent of each other, the probability of the training data under the distribution $P(b)$ is:

$$P(training|P(b)) = P(b)^{Count(b)}(1 - P(b))^{S-Count(b)}$$

Finding the $P(b)$ that maximizes $P(training|P(b))$ is equivalent to finding the $P(b)$ that maximizes the logarithm of $P(training|P(b))$, which is:

$$\begin{aligned}\log P(training|P(b)) &= \log\left(P(b)^{Count(b)}(1 - P(b))^{S-Count(b)}\right) \\ &= Count(b)\log P(b) + (S - Count(b))\log(1 - P(b))\end{aligned}$$

The maximum point for the function $\log P(training|P(b))$ can be found by solving the equation $\frac{d}{dP(b)}(\log P(training|P(b))) = 0$:

$$\begin{aligned}&\frac{d}{dP(b)}(\log P(training|P(b))) = 0 \\ &\frac{Count(b)}{P(b)} - \frac{(S - Count(b))}{(1 - P(b))} = 0 \\ &P(b) = \frac{Count(b)}{S},\end{aligned}$$

and this is the maximum likelihood estimate as expressed by Equation 2.1.

Equation 2.1 can be generalized to multiple classes, where the set of class values is not restricted to $\{0, 1\}$. Collins (1999) gave a proof of the generalization. MLE has been used in the hidden Markov model for POS-tagging, and generative models for parsing (Collins, 1997).

Another important method for the estimation of a probability distribution is the *maximum entropy* (ME) principle (Berger et al., 1996). According to the ME principle, given some partial knowledge about a distribution, the best estimation of the whole distribution function should be the one with the maximum entropy among all functions that are consistent with the partial knowledge, or, by the definition of entropy, the most uniform distribution among those consistent with the given partial knowledge.

Take the classification problem for example, where an input $x \in X$ is mapped to a class $c \in C$. The predicted class $c_{out}(x)$ for an arbitrary input x is the one with the highest conditional probability:

$$c_{out}(x) = \arg\max_{c' \in C} P(c'|x).$$

According to the ME principle, the probability distribution $P(c|x)$ is estimated using statistical information from training examples represented by a set of features $f_1(x, c), f_2(x, c), .., f_n(x, c)$. Here a feature indicates the occurrence of a pattern in x and c. For example, in the spam filtering problem, where the input is a document and the output is from $\{spam, nonspam\}$, a feature value can be "the document contains the word FREE", or "the document does not have capital letters". The set of features represents the partial knowledge of the distribution to estimate.

In ME estimation, it is assumed that the expected value of each feature according to the distribution is the same as the actual value of the feature according to the relative frequency in the training data:

$$\sum_{c,x} \frac{Count(x)}{T} P(c|x) f_j(x, c) = \sum_{c,x} \frac{Count(x, c)}{T} f_j(x, c), j \in 1..n,$$

where x represents any possible input value and c represents any possible class from the set C; $Count(x)$ represents the count of the input value x from the training data, $Count(x, c)$ represents the number of occurrences of the pair (x, c) in the training data, and T represents the total number

of training examples.

The above equations provide the basic constraints under which ME estimation searches for the probability distribution $P(c|x)$ with the highest entropy. Besides these, there are also implicit constraints including $P(c|x) > 0$, and $\sum_c P(c|x) = 1$. The target function of the constraint maximization tasks, the entropy of the distribution $P(c|x)$, is:

$$H(c|x) = -\sum_{x,c} \frac{Count(x)}{T} P(c|x) \log P(c|x)$$

The solution $P_\lambda(c|x)$ to the above constraint maximization problem has the form[2]:

$$P_\lambda(c|x) = \frac{1}{Z_\lambda(x)} \exp\left((\sum_j \lambda_j f_j(x,c)) \right), \tag{2.2}$$

which is the equation of the ME model. In the above equation, $Z_\lambda(x) = \sum_c \exp\left(\sum_j \lambda_j f_j(x,c)\right)$ is the normalizing constant that guarantees $\sum_c P_\lambda(c|x) = 1$.

Equation 2.2 is parameterized and easy to compute on the basis of features. The parameters in the ME model are the λ values. There is no closed form solution to compute them. Instead, several numerical methods can be used to find the λ values, including the improved iterative scaling (IIS) method (Berger et al., 1996; Della Pietra et al., 1997) and the generalized iterative scaling (GIS) method (Darroch and Ratcliff, 1972; Curran and Clark, 2003). In addition, more general numerical optimization methods have become popular (Malouf, 2002).

Besides the classification problem, ME is also used in the MEMM and CRF models.

The naive Bayes classifier

The naive Bayes classifier is one of the simplest probabilistic models for the classification task, where the output is a discrete value from a set of classes C. For example, in the document classification problem, the input is a document and the set of output classes can be {*fiction*, *scientific report*, *news article*, *others*}. Denoting the set of inputs with X, the conditional probability of the specific input $x \in X$ belonging to the class $c \in C$ can be written as $P(c|x)$. The goal of the classification

[2]More details about the mathematical deduction for Equation 2.2 are expressed by Berger et al. (1996).

task is to find the output $c_{out}(x)$ that satisfies:

$$c_{out}(x) = \arg\max_{c' \in C} P(c'|x)$$

The probability $P(c|x)$ can not be estimated directly from training data, since documents are statistically very sparse. On the other hand, the probability chain rule can not be applied here, because x is the condition in the conditional probability equation. The naive Bayes classifier addresses this problem by removing x from the condition position, using Bayes rule:

$$\begin{aligned} c_{out}(x) &= \arg\max_{c'} P(c'|x) \\ &= \arg\max_{c'} \frac{P(c')P(x|c')}{P(x)} \\ &= \arg\max_{c'} P(c')P(x|c') \end{aligned}$$

Now the conditional probability $P(x|c')$ can be estimated using the probability chain rule and independence assumptions. For example, the "bag of words" approach treats the document as a collection of independent individual words $w_1, w_2, .., w_n$ given the document class. According to this approach, the prediction problem can be simplified as:

$$\begin{aligned} c_{out}(x) &= \arg\max_{c'} P(c')P(x|c') \\ &= \arg\max_{c'} P(c')P(w_1, w_2, .., w_n|c') \\ &= \arg\max_{c'} P(c')P(w_1|c')P(w_2|c')..P(w_n|c') \end{aligned}$$

In the above equation, the output probability consists of two types components: the prior probability $P(c)$ and the conditional probabilities $P(w_i|c), i \in 1..n$. These two types of probabilities are comparatively easy to estimate, and both can be estimated by supervised learning using maximum likelihood estimation, and normally a smoothing method for unseen data.

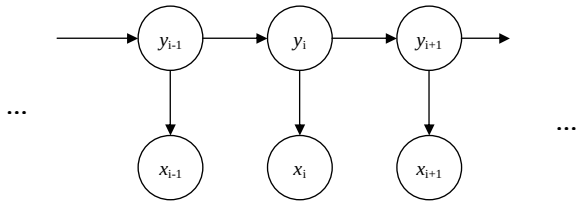


Figure 2.1: Probabilistic dependencies in an HMM

The hidden Markov model

The hidden Markov model (HMM) has been used for many years as a generative model for NLP (Rabiner, 1989). It is commonly used for the sequence labeling problem. Figure 2.1 gives an illustration of the stochastic generation process according to the first-order HMM. In this model, a sequence of states y is generated according to the Markov process – each state being dependent only on its predecessor (the first order assumption). The whole state sequence is hidden, and the value of each state y can only be inferred from a corresponding observation x, which is generated as a result of y.

A typical problem solved by the HMM is finding the most likely state sequence given the corresponding observation sequence. Unlike classification problems, where the output is a single discrete value, the output of this problem is a sequence of inter-related states. In the most probable state sequence, each individual state is not necessarily the most probable if considered in isolation. Therefore, finding the best state sequence in an HMM is a *structural prediction* problem.

Denoting the observations with $X_1^n = x_1, x_2, ..., x_n$, the most probable state sequence can be written as $(Y_1^n)_{out}(X_1^n) = \arg\max_{y'_1,..,y'_n} P(y'_1, .., y'_n | x_1, .., x_n)$, where $y'_1, .., y'_n$ can be any possible state value sequence. Direct estimation of this probability is difficult due to data-sparseness. Because the input is also complex, we can first turn the conditional probability into a joint probability, so that the probability chain rule and independence assumptions can be used to break the joint probability

into less sparse components:

$$\begin{aligned}
(Y_1^n)_{out}(X_1^n) &= \arg\max_{y'_1,..,y'_n} P(y'_1,..,y'_n|x_1,..,x_n) \\
&= \arg\max_{y'_1,..,y'_n} \frac{P(y'_1,..,y'_n,x_1,..,x_n)}{P(x_1,..,x_n)} && \text{(def. of conditional probabilities)} \\
&= \arg\max_{y'_1,..,y'_n} P(y'_1,..,y'_n,x_1,..,x_n) && \text{(denominator fixed)}
\end{aligned}$$

It can be seen that the state sequence with the highest conditional probability also has the highest joint probability. The joint probability $P(y'_1,..,y'_n,x_1,..,x_n)$ can be reduced to less sparse components according to the probability chain rule and first-order Markov assumption in the following way:

$$\begin{aligned}
P(y'_1,..,y'_n,x_1,..,x_n) &= P(y'_1,..,y'_{n-1},x_1,..,x_{n-1},y'_n,x_n) \\
&= P(y'_1,..,y'_{n-1},x_1,..,x_{n-1})P(y'_n,x_n|y'_1,..,y'_{n-1},x_1,..,x_{n-1}) \\
&= P(y'_1,..,y'_{n-1},x_1,..,x_{n-1})P(y'_n|y'_1,..,y'_{n-1},x_1,..,x_{n-1}) \\
&\quad P(x_n|y'_1,..,y'_{n-1},x_1,..,x_{n-1},y'_n) \\
&= P(y'_1,..,y'_{n-1},x_1,..,x_{n-1})P(y'_n|y'_{n-1})P(x_n|y'_n) && \text{(cond. independence)} \\
&= P(y'_2|y'_1)..P(y'_n|y'_{n-1})P(x_1|y_1)..P(x_n|y_n)
\end{aligned}$$

Now the computation of $P(y'_1,..,y'_n,x_1,..,x_n)$ is reduced to the product of two basic types of probabilities $P(y'_n|y'_{n-1})$ and $P(x_n|y'_n)$, both of which are much less sparse than the full joint probability, and can be estimated according to the maximum likelihood principle, resulting in simple relative frequency estimates of the two probabilities. They are also called the transition probabilities and the observation probabilities in a first-order HMM, respectively.

Finding the most probable class in a classification problem can be achieved by simple enumeration of possible classes. However, finding the most probable state sequence for a given observation sequence is not as straightforward, since the number of possible state sequences given an observation sequence is exponential in the length of the sequence. A common solution is to use the Viterbi algorithm (Rabiner, 1989), a dynamic programming algorithm that explores the exponential search

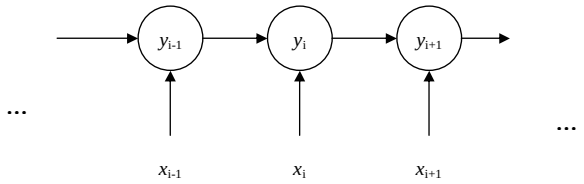


Figure 2.2: Probabilistic dependencies in an MEMM

space in polynomial time by means of caching. Denoting the best state sequence ending with $y_n = y$ as $(Y_1^n)_{max}(y, X_1^n)$, since the finding of $(Y_1^n)_{max}(y, X_1^n)$ depends only on $(Y_1^{n-1})_{max}(y', X_1^{n-1})$, $(Y_1^{n-1})_{max}(y', X_1^{n-1})$ can be cached for all possible y', and then reused in finding $(Y_1^n)_{max}(y, X_1^n)$. Starting from the first state, the Viterbi algorithm incrementally builds a chart of $(Y_1^i)_{max}(y, X_1^n)$ for all y and $i \in 1..n$.

The maximum entropy Markov model

The maximum entropy Markov model (MEMM) (McCallum et al., 2000) is another model for a sequence of states generated by the Markov process. As for an HMM, each state in an MEMM is related to an observation, and a typical problem is finding the most probable state sequence given the observation sequence. But unlike an HMM, an MEMM does not assume that each observation is generated by its corresponding state. Instead, it regards the probability of each state item as dependent on the probability of its predecessors and its observation. Therefore, the MEMM can be viewed as designed particularly for the problem of predicting state sequences. An illustration of the first-order MEMM is shown in Figure 2.2. It can be noticed that the direction of dependencies is different from an HMM.

In contrast to an HMM, which computes the probability of a state sequence by two types of basic probabilities, the transition probability and the observation probability, an MEMM breaks the computation of state sequence probabilities into only one basic probability – the state transition probability. Suppose that the state sequence is $Y_1^n = y_1, y_2, .., y_n$ and the observation sequence is $X_1^n = x_1, x_2, .., x_n$. The state transition probability can be written as $P(y_i|x_i, y_{i-1})$. Note that the observation x_i here is a condition, as compared to the observation probability $P(x_i|y_i)$ in an HMM.

To find the most probable state sequence given the observation sequence, an MEMM uses the probability chain rule and independence assumptions to turn the joint probability $P(y_1, .., y_n, x_1, .., x_n)$ into components:

$$\begin{aligned} P(y_1, .., y_n|x_1, .., x_n) &= P(y_1, .., y_{n-1}|x_1, .., x_n)P(y_n|y_1, .., y_{n-1}, x_1..x_n) \\ &= P(y_1, .., y_{n-1}|x_1, .., x_{n-1})P(y_n|y_{n-1}, x_n) \qquad \text{(cond. independence)} \\ &= P(y_2|y_1, x_2)..P(y_n|y_{n-1}, x_n) \end{aligned}$$

The above equation consists of only one simple component, the state transition probability $P(y_n|y_{n-1}, x_n)$. The values of $P(y_i|x_i, y_{i-1})$ are estimated from training data using the ME principle. However, instead of being estimated directly, the distribution $P(y_i|x_i, y_{i-1})$ is separated into independent sub distributions according to different values of y_{i-1}, each being labeled as $P_{y_{i-1}}(y_i|x_i)$ and trained separately. Notice that the format of a particular sub distribution $P_{y_{i-1}}(y_i|x_i)$ is similar to that of the ME classification problem introduced earlier, with y_i being class c and x_i being input x. Therefore, by the same equation derivation as the ME classification problem, the mathematical form of $P_{y_{i-1}}(y_i|x_i)$ is:

$$P_{y_{i-1}}(y_i|x_i) = \frac{1}{Z(x_i, y_{i-1})} \exp\left(\sum_{j=1}^{m} \lambda_j f_j(x_i, y_i)\right).$$

In the above equation, $f_1, ..., f_m$ represent features extracted from x_i and y_i. The normalizing factor $Z(x_i, y_{i-1})$ ensures that $\sum_{y_i} P_{y_{i-1}}(y_i|x_i) = 1$. It has y_{i-1} as a parameter because $P_{y_{i-1}}(y_i|x_i)$ is a set of probabilities that are defined separately for each possible value of y_{i-1}. The set of λ values, which is also dependent on y_{i-1}, can be trained by the generalized iterative scaling method (McCallum et al., 2000).

Both MEMM and HMM are probabilistic models for Markov processes, and the difference between them is mainly in the way the conditional probability of a sequence structure is turned into basic components in order to simplify the estimation problem. While an HMM defines a generation story for the observations, an MEMM uses the maximum entropy principle and simplifies the computation of state transition probabilities by means of features. The main advantage of the MEMM is that

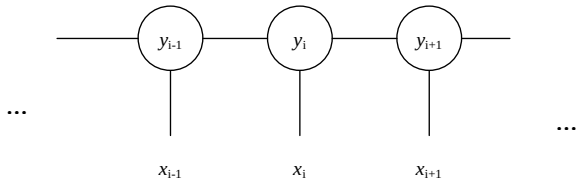


Figure 2.3: Probabilistic dependencies in a CRF

a much wider range of contextual information can be included. Besides, an MEMM computes the conditional probability of a state sequence given an observation directly, without transforming it to the computation of joint state and observation probabilities, as an HMM does.

Like the HMM, the Viterbi algorithm can be used to compute the most probable sequence from all possible state sequences.

Conditional random fields

One of the major contributions of the MEMM is the introduction of rich features by using the maximum entropy state transition model. However, the transition model also brings disadvantages, an important example being the label bias problem (Lafferty et al., 2001), where a label with few sucessors can be preferred than one with many due to higher transition probabilities. The conditional random field (CRF) model has been proposed to address this issue.

CRFs are defined on general graphs that obey the Markov property, of which the linear chain is the simplest and most common example. An illustration of the first-order linear-chain CRF is shown in Figure 2.3. Unlike the HMM and MEMM, a CRF does not break the conditional probability of a state sequence into the production of basic probabilities, but computes it directly by using the ME principle.

Denoting the observation sequence by $X_1^n = x_1, .., x_n$, and the state sequence by $Y_1^n = y_1, .., y_n$, the CRF model estimates the probability distribution $P(Y_1^n|X_1^n)$ using the ME principle. Here features are defined globally over Y_1^n and X_1^n, while according to the first-order Markov assumption, all features are limited to the context of a state, its observation and its previous state. Because the same pattern can occur multiple times with different states in the sequence, a CRF feature represents the count of occurrences of a particular pattern in the structure. This is different from an MEMM,

where features are defined for each state transition separately, and are typically binary pattern indicators. We use the terms *global features* and *local features* to differentiate global count features $F(X_1^n, Y_1^n)$ from local binary features $f(y_i, x_i, y_{i-1})$. The relationship between global and local features is:

$$F_j(X_1^n, Y_1^n) = \sum_{i=1}^{n} f_j(y_i, x_i, y_{i-1}),$$

where j is a number that represents the identity of a particular feature value.

Using the ME model to find $P(Y_1^n|X_1^n)$, the distribution has the form:

$$P(Y_1^n|X_1^n) = \frac{1}{Z} \exp\left(\sum_j \lambda_j F_j(X_1^n, Y_1^n)\right) \tag{2.3}$$

In this equation, Z is the normalizing constant that makes $P(Y_1^n|X_1^n)$ sum to 1. Lafferty et al. (2001) used IIS to estimate the parameters λ_j. However, recent research has found more efficient numerical methods to estimate these values (Wallach, 2002; Malouf, 2002).

Equation 2.3 for the CRF is very similar to the ME classifier (Equation 2.2). But it is different in that the target is a state sequence Y_1^n rather than a single class c. Consequently, the computations for a CRF are more complex than for the simple ME classifier. For example, the normalizer $Z = \sum_{Y_1^n} \exp\left(\sum_j \lambda_j F_j(X_1^n, Y_1^n)\right)$ requires summation over all possible state sequences Y_1^n, and is computationally more challenging.

HMM, MEMM and CRF solve the same structural prediction problem of finding the most probable state sequence. While an HMM defines a generation process for the state and observation sequences, estimating transition and observation probabilities, a CRF treats the whole state sequence as a single unit, estimating its conditional probability using the ME principle. The difference between an HMM and a CRF reflects the difference between generative models and discriminative models. The main advantage of a CRF over a HMM is the freedom to define features without making independence assumptions. CRFs have been reported to outperform both HMMs and MEMMs in many NLP tasks (Lafferty et al., 2001; Sha and Pereira, 2003; Peng et al., 2004).

Inputs: training examples (x_i, c_i)
Initialization: set $\vec{w} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = sgn(\Phi(x_i) \cdot \vec{w})$
    $\vec{w} = \vec{w} + \eta(c_i - z_i)\Phi(x_i)$
Outputs: $\vec{w}$

Figure 2.4: The traditional perceptron learning algorithm

2.1.2 Non-probabilistic models

The comparison between HMMs and CRFs above reflected the general difference between discriminative probabilistic models and generative probabilistic models. An important advantage of discriminative models is the use of features, which are able to capture a wide range of contextual information.

The non-probabilistic models in this thesis are all discriminative: they collect statistical information from the training data by using a set of global features, and score output candidates for a given input according to them. A higher score from a non-probabilistic model represents a more correct candidate according to the model. However, the score values are not probabilities. This is the most important difference between non-probabilistic models and probabilistic models.

The generalized perceptron algorithm is a non-probabilistic discriminative model that achieved competitive accuracies compared to a CRF in a POS-tagging task (Collins, 2002). Compared to a CRF, the perceptron learning algorithm is conceptually more straightforward, and often more efficient in time and space complexity, depending on the decoder. Moreover, the perceptron algorithm does not make any assumptions about the structure, and therefore can be used in a much larger range of structural prediction problems than a CRF. Though other margin-based models sometimes outperform the perceptron algorithm in certain tasks, they typically have worse time complexity.

The traditional, generalized and averaged perceptron algorithms

The traditional perceptron model (Rosenblatt, 1958) is a linear model for binary classification. Given an input x, and a set of local features $f_1(x), f_2(x), ..., f_m(x)$ defined similarly to the previous

sections, the output of a perceptron is

$$c_{out}(x) = \begin{cases} 1 & \text{if } w_0 + w_1 f_1(x) + .. + w_m f_m(x) > 0 \\ 0 & \text{otherwise} \end{cases}$$

w_i represents the set of linear weights for the features; they are the parameters in the perceptron model and are computed from training data. Adding a constant $f_0(x) = 1$, the features can be put into a vector $\Phi(x) = (f_0(x), f_1(x), .., f_m(x))$. Similarly, the corresponding weights for each feature can be put into a vector $\vec{w} = (w_0, w_1, .., w_m)$, thereby simplifying the perceptron function as $c_{out}(x) = sgn(\Phi(x) \cdot \vec{w})$, where the function sgn is defined as:

$$sgn(m) = \begin{cases} 1 & \text{if } m > 0 \\ 0 & \text{otherwise} \end{cases}$$

The perceptron learning algorithm is illustrated in Figure 2.4. It is an *online learning* algorithm that uses the current weight vector to predict training examples. For a training example, if the prediction is correct, $\vec{w}$ remains unchanged because $c_i = z_i$. Otherwise, the parameters are adjusted in the opposite direction of the error. η is a positive factor called the learning rate; it controls the scale by which $\vec{w}$ is modified each time. The same process can be performed over the N training examples for T training iterations.

The generalized perceptron algorithm (Collins, 2002) extends the traditional perceptron algorithm to solve structural prediction problems. For the generalized perceptron, both the input x and the output y can be structured, and features are defined over both x and y. Denoting the set of candidate outputs for the input x as $\text{GEN}(x)$, the features vector as $\Phi(x, y)$, and the parameter vector as $\vec{w}$, the prediction output is:

$$\begin{aligned} y_{out}(x) &= \underset{y' \in \text{GEN}(x)}{\arg\max}\ Score(y') \\ &= \underset{y' \in \text{GEN}(x)}{\arg\max}\ \Phi(x, y') \cdot \vec{w} \end{aligned}$$

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \vec{w}$
    if $z_i \neq y_i$
      $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

Figure 2.5: The generalized perceptron learning algorithm

The training process for the generalized perceptron is illustrated in Figure 2.5. Similarly to Figure 2.4, predictions are made for training examples using the current parameter vector. If the prediction is correct, the trainer goes on to the next example without modifying the parameter values; otherwise it adjusts the parameter vector according to the correct example and the incorrect prediction, by adding the feature values from the correct example and subtracting the feature values from the prediction. The effect of this procedure is that features from the correct example but not from the prediction are given increased weight, while features from the prediction but not from the correct example are given decreased weight, so that the model is forced to make correct predictions for the training example. T training iterations can be performed over N examples.

The averaged perceptron algorithm (Collins, 2002) is a standard alternative to the generalized perceptron algorithm to reduce overfitting on the training data. It was motivated by the voted-perceptron algorithm (Freund and Schapire, 1999) and has been shown to give improved accuracy over the non-averaged perceptron on a number of tasks. Let N be the number of training sentences, T the number of training iterations, and $\vec{w}^{n,t}$ the parameter vector immediately after the nth sentence in the tth iteration. The averaged parameter vector $\vec{\gamma} \in R^d$ is defined as:

$$\vec{\gamma} = \frac{1}{NT} \sum_{n=1..N, t=1..T} \vec{w}^{n,t}$$

The averaged perceptron uses $\vec{\gamma}$ instead of $\vec{w}$ as the parameter vector for the final model.

The generalized perceptron algorithm can be seen as a natural alternative to the CRF method in solving the state sequence problem (Collins, 2002). In both cases a feature vector consists of the global sum of different features over the whole state sequence, and the weights $w_1, ..., w_n$ in the

perceptron correspond to the parameters $\lambda_1, .., \lambda_n$ in the CRF. The two models were reported to achieve comparable accuracy for many NLP problems. Compared to a CRF, the advantage of the perceptron algorithm is online learning, which is typically less demanding on memory. The training of the perceptron is dependent on the decoding procedure, while the computation in parameter updating is trivial. With an efficient decoding algorithm, the perceptron algorithm can achieve fast learning speeds.

Margin infused relaxed algorithm (MIRA)

MIRA (Crammer and Singer, 2003; McDonald et al., 2005a) is a margin-based online learning algorithm that shares similarities with the perceptron algorithm, but the parameter update for MIRA is more complex.

Crammer and Singer (2003) studied the parameter update for online algorithms that make predictions for training examples and adjust parameters according to the predicted output and correct output. For binary classification, algorithms that do not adjust parameters when the prediction is correct are *conservative*. Conservativeness can be extended to the multi-class case. Suppose that there are m classes $c_1, .., c_m$. For each input x, we want the score of the correct class c to be higher than the rest of the classes, i.e. $Score(x, c) > Score(x, c')$, $c' \neq c$. Now define the error set E to be the set of classes $\{c' \neq c : Score(x, c') \geq Score(x, c)\}$. An algorithm that updates its parameters only when E is non-empty, and only using classes in $E \cup \{c\}$, is called *ultraconservative* (Crammer and Singer, 2003). The generalized perceptron algorithm is ultraconservative, because it adjusts the parameters only using $\arg\max_{c' \in E} Score(x, c')$ and c.

The generalized perceptron algorithm adjusts the parameters by adding the feature vector of the correct output and subtracting the feature vector of the predicted output:

$$\vec{w}_{t+1} = \vec{w}_t + \Phi(x, c) - \Phi(x, \arg\max_{c' \in E} Score(c'))$$

This formula can also be written as:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_c \times \Phi(x, c) + \tau_{c'} \times \Phi(x, c'), \quad c' = \arg\max_{c'' \in E} Score(c'')$$

where $\tau_c = 1$ is the weight for the correct vector and $\tau_{c'} = -1$ is the weight for the decoder output. Crammer and Singer (2003) gave a generalization of the formula and showed that a family of ultraconservative algorithms have the same mistake bound as the generalized perceptron algorithm, if they update the weights in the following way:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_c \times \Phi(x, c) + \sum_{c' \in E} \tau_{c'} \times \Phi(x, c')$$

where $\tau_c = 1$ and $\sum_{c' \in E} \tau_{c'} = -1$. For example, the following online algorithm is a member of this family:

$$\vec{w}_{t+1} = \vec{w}_t + \Phi(x, c) - \frac{1}{\|E\|} \sum_{c' \in E} \Phi(x, c')$$

For each c' in the above algorithm, $\tau_{c'} = 1/\|E\|$, where $\|E\|$ is the size of E.

MIRA is the ultraconservative learning algorithm that makes the smallest adjustment for the parameters in each update in order to give a correct prediction for the current training example. In equation form, MIRA finds:

$$\vec{w}_{t+1} = \arg\min_{\vec{w}} \|\vec{w} - \vec{w}_t\|^2$$

under the constraints that:

$$\vec{w}_{t+1} \cdot \Phi(x, c) - \vec{w}_{t+1} \cdot \Phi(x, c') \geq 0, \quad \forall c' \in E$$

MIRA has been shown to outperform the generalized perceptron by a small amount in a number of NLP problems. However, because the updating of parameters is a constraint minimization problem, it is much more expensive than the perceptron algorithm.

Online passive aggressive algorithms

The passive-aggressive algorithms (Crammer et al., 2006) are simplifications to the MIRA algorithm. While the updating of parameters in MIRA is a constraint minimization problem, which requires expensive numerical calculations, the online passive-aggressive algorithms have closed form update functions, which are much simpler to compute.

The passive-aggressive algorithm is the same as MIRA for binary classification problems. The

constraint minimization problem can be stated as below:

$$\vec{w}_{t+1} = \arg\min_{\vec{w}} \frac{1}{2}\|\vec{w} - \vec{w}_t\|^2 \text{ s.t. } loss(\vec{w}; (x_t, c_t)) = 0,$$

where the loss function $loss(\vec{w}; (x_t, c_t))$ is:

$$loss(\vec{w}; (x, c)) = \begin{cases} 0 & \text{if } c(\vec{w} \cdot x) \geq 1 \\ 1 - c(\vec{w} \cdot x) & \text{otherwise} \end{cases}$$

The solution to this constraint minimization problem is a closed-form equation:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_t c_t x_t \text{ where } \tau_t = \frac{loss(\vec{w}; (x, c))}{\|x_t\|^2}$$

In the case of multi-class classification, however, MIRA does not have a closed form solution, largely because there are multiple constraints. In comparison, the passive-aggressive algorithms ignore all but the most violated constraint. Suppose that the correct output for the input x_t is c_t, while the wrong output with the highest score is c'; then the update method can be written as:

$$\vec{w}_{t+1} = \arg\min_{\vec{w}} \frac{1}{2}\|\vec{w} - \vec{w}_t\|^2 \text{ s.t. } loss(\vec{w}; (x_t, c_t)) = 0.$$

The loss function is the hinge loss for the margin between the correct output and the highest-ranked wrong output only:

$$loss(\vec{w}; (x_t, c_t)) = \begin{cases} 0 & \text{if } \vec{w} \cdot \Phi(x, c_t) - \vec{w} \cdot \Phi(x, c') \geq 1 \\ 1 - (\vec{w} \cdot \Phi(x, c_t) - \vec{w} \cdot \Phi(x, c')) & \text{otherwise} \end{cases}$$

Now this equation has a closed form solution:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_t c_t x_t \text{ where } \tau_t = \frac{l_t}{\|\Phi(x, c_t) - \Phi(x, c')\|^2},$$

which is much simpler to compute than the learning problem in MIRA.

Passive-aggressive algorithms can be extended with two additional features. Firstly, the slack

variable ξ (see the sub section on SVM for more details of this variable) can be applied to the update equations, so that the learning algorithms become more resistant to noise in the training data:

$$\vec{w}_{t+1} = \arg\min_{\vec{w}} \frac{1}{2}\|\vec{w} - \vec{w}_t\|^2 + C\xi \quad \text{s.t. } loss(\vec{w}; (x_t, c_t)) \leq \xi \text{ and } \xi \geq 0.$$

The closed form solution for this equation is:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_t c_t x_t \quad \text{where } \tau_t = \min\left(C, \frac{loss(\vec{w}, (x_t, c_t))}{\|\Phi(x, c_t) - \Phi(x, c')\|^2}\right)$$

The slack variable can be introduced in another way:

$$\vec{w}_{t+1} = \arg\min_{\vec{w}} \frac{1}{2}\|\vec{w} - \vec{w}_t\|^2 + C\xi^2 \quad \text{s.t. } loss(\vec{w}; (x_t, c_t)) \leq \xi \text{ and } \xi \geq 0.$$

The closed form solution for the above equation is:

$$\vec{w}_{t+1} = \vec{w}_t + \tau_t c_t x_t \quad \text{where } \tau_t = \frac{loss(\vec{w}, (x_t, c_t))}{\|\Phi(x, c_t) - \Phi(x, c')\|^2 + \frac{1}{2C}}$$

The passive-aggressive algorithms with slack variables introduced in the above two ways are called PA-I and PA-II, respectively. The algorithm without slack variables is called PA.

Secondly, a real-valued loss function $\rho(c_t, c')$ can be used to measure the difference between the incorrect and correct outputs. It is a sensible choice when the error of an output is measurable. For example, in the POS-tagging problem, ρ may represent the number of words that are tagged incorrectly. The loss with $\rho(c_t, c')$ introduced is:

$$loss(\vec{w}; (x_t, c_t)) = \begin{cases} 0 & \text{if } \vec{w} \cdot \Phi(x, c_t) - \vec{w} \cdot \Phi(x, c') \geq \rho(c_t, c') \\ \rho(c_t, c') - (\vec{w} \cdot \Phi(x, c_t) - \vec{w} \cdot \Phi(x, c')) & \text{otherwise} \end{cases}$$

And the rest of the computation can be done in the same way as the previously introduced methods.

The above passive-aggressive algorithms can also be applied to structural prediction problems. Crammer et al. (2006) showed that for several tasks, the passive-aggressive algorithms gave slightly lower performance than MIRA. The passive-aggressive algorithms have the same time complexity

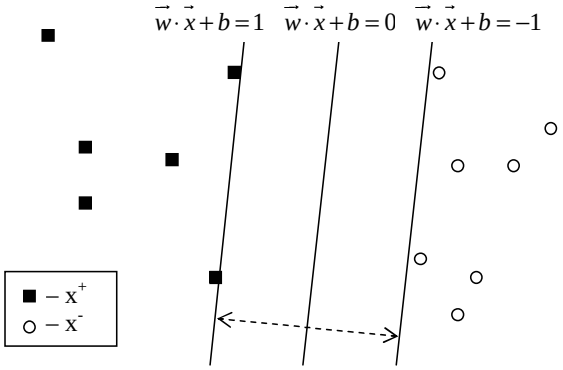


Figure 2.6: An illustration of SVM

for parameter updating as the perceptron algorithm, performing only closed form calculations. We implemented the passive-aggressive algorithm for the joint segmentor and tagger in Chapter 4, and compared its accuracy with the perceptron algorithm. In our experiments, the passive-aggressive algorithm gave lower accuracy than the perceptron algorithm.

Support vector machines

The standard support vector machine (SVM) (Boser et al., 1992) is a binary classifier, with identical score computation as all the non-probabilistic binary classifiers reviewed previously. Denoting the input by x, the global feature vector by $\Phi(x) = (f_1(x), ..., f_n(x))$, the output of an SVM is $c_{out}(x) = sgn(\Phi(x) \cdot \vec{w} + w_0)$, where $\vec{w} = (w_1, ..., w_n)$ is the parameter vector.

Unlike the perceptron algorithm, the standard SVM is not an online learning algorithm. It estimates the parameter values $\vec{w}$ by considering all training examples simultaneously. SVM maps the parameter estimation problem into a geometric problem, where each training example x is mapped into a point $(f_1(x), ..., f_n(x))$ in an n dimensional vector space (Figure 2.6). The equation $\Phi(x) \cdot \vec{w} = y, y \in R$ corresponds to a hyperplane in the space. Suppose that the training data is linearly separable; then the equation $\Phi(x) \cdot \vec{w} + w_0 = 0$ represents a hyperplane that separates the two classes of training data. Specifically, for all points x^+ on one side, $\Phi(x) \cdot \vec{w} + w_0 > 0$, and $c(x^+) = sgn(\Phi(x) \cdot \vec{w} + w_0) = 1$, while for all points x^- on the other side, $c(x^-) = -1$. By varying the values of $\vec{w}$ and w_0, there can be many such hyperplanes that divide the two classes of training data. SVM learning finds the one that maximizes the distance between the hyperplane and

the nearest point to it on either side, and uses the corresponding $\vec{w}$ values as its model parameters. Intuitively, the chosen hyperplane maximizes the margin between the two classes of training points, thereby allowing the most freedom to possible unseen points on either side. The points nearest to the maximum-margin hyperplane are called the *support vectors*. Because they are the most important determining factors for the model, the classifier is named after them.

Define the support vectors as $\Phi(x) \cdot \vec{w} + w_0 = \pm 1$ (Figure 2.6). By geometric reasoning, the maximum margin for the hyperplane $\Phi(x) \cdot \vec{w} + w_0 = 0$ is equal to $2/\|\vec{w}\|$. Hence the SVM learning problem can be formulated as:

find the $\vec{w}$ and w_0 that minimizes $\frac{1}{2}\|\vec{w}\|^2$

under the restriction that for any (x_i, y_i) in the training data, $y_i \times c_{out}(x_i) \geq 1$

An important limitation of the above version of SVM is the assumption that the training data are separable, whilst the training data in real applications are often non-separable. This problem is addressed by the soft-margin variation of SVM (Cortes and Vapnik, 1995), which uses a slack variable to reformulate the learning problem as:

find the $\vec{w}, w_0$ and ξ that minimizes $\frac{1}{2}\|\vec{w}\|^2 + C\sum_i \xi_i$

under the restriction that for any (x_i, y_i) in the training data, $y_i \times c_{out}(x_i) \geq 1 - \xi_i$

In the above equation, a slack variable ξ_i is introduced for each training example (x_i, x_y) to represent its difference from the "ideal" situation, while C is the parameter that controls the balance between maximizing the margin (i.e. minimizing $\|\vec{w}\|$) and minimizing exceptional points (or training error $\sum_i \xi_i$). Now the training data is no longer required to be completely separable. Nonetheless, the trained model is still expected to separate most of the training examples. Therefore, the soft-margin SVM is tolerant of errors in the training data.

Both the original SVM and the SVM with slack variables are reduced to the constraint minimization problem, which can be solved by various methods. The traditional solution is quadratic programming, while recently other efficient methods have been proposed, such as the sequential

minimal optimization (Platt, 1998) and the gradient based method (Ratliff et al., 2007).

Structural SVM (Tsochantaridis et al., 2004) is an extension of the binary SVM for structural prediction problems. The relationship between structural SVM and the standard SVM resembles that between the generalized and the traditional perceptron algorithms. Denote the set of possible outputs for the input x with GEN(x); the prediction problem for the structural SVM can be formulated as:

$$\begin{aligned} y_{out}(x) &= \underset{y \in \mathrm{GEN}(x)}{\arg\max}\ Score(y) \\ &= \underset{y \in \mathrm{GEN}(x)}{\arg\max}\ \Phi(x, y) \cdot \vec{w} \end{aligned}$$

where $\vec{w}$ is the vector of model parameters.

To extend SVM learning to the structural case, two problems need to be considered. Firstly, when mapping the training examples into the vector space, the concept of margin needs to be clarified, so that the maximum-margin hyperplane can be defined. In the structural case, the margin is less obvious to define than in the binary case, because there can be more than two classes of output points. Secondly, with structured outputs, the difference between a wrong answer and the correct answer can be measurable. In other words, some answers can be more wrong than others. For example, in the parsing problem, an incorrect parse tree may be similar to the correct parse, while another may be very different; or in the tagging problem, a tag sequence can contain 5 incorrect tags, while another only 2.

To address the first issue, structural SVM defines the margin as the score difference between the correct output and the highest-scored incorrect output. In other words, the margin is the smallest score difference between the correct output and any incorrect output. According to this margin, the mathematical equation for structural SVM learning can be formulated as:

find the $\vec{w}$ that minimizes $\frac{1}{2}\|\vec{w}\|^2$

s. t. for any $(y \neq y_i)$ in the training data, $\vec{w} \cdot \Phi(y_i, x) - \vec{w} \cdot \Phi(y, x) \geq 1$

Similar to the binary case, the slack variable can be introduced to structural SVM, so that inconsistency in the training data can be tolerated. There are many ways to incorporate the slack variable into the mathematical equation. The structural SVM adds one slack variable to each linear constraint:

$$\text{find the } \vec{w}, \xi \text{ that minimizes } \frac{1}{2}\|\vec{w}\|^2 - \frac{C}{n}\sum_{i=1}^{n}\xi_i, \text{ s.t. } \xi_i \geq 0$$

$$\text{s. t. for any } (y \neq y_i) \text{ in the training data, } \vec{w}\cdot\Phi(y_i, x) - \vec{w}\cdot\Phi(y, x) \geq 1 - \xi_i$$

To address the second issue, a specific loss function ρ that reflects the actual difference between two structured outputs can be put into the constraint functions. One way to introduce ρ is using it to replace the fixed margin 1 in the constraint functions, so that the score difference between the correct output and a wrong output is scaled by the actual difference between the two structures. Using this method, the learning equation with slack variables is:

$$\text{find the } \vec{w}, \xi \text{ that minimizes } \frac{1}{2}\|\vec{w}\|^2 - \frac{C}{n}\sum_{i=1}^{n}\xi_i, \text{ s.t. } \xi_i \geq 0$$

$$\text{s. t. for any } (y \neq y_i) \text{ in the training data, } \vec{w}\cdot\Phi(y_i, x) - \vec{w}\cdot\Phi(y, x) \geq \rho(y_i, y) - \xi_i$$

The above method for structural SVM learning is also called *maximum margin Markov networks* (Taskar et al., 2003). Alternatively, the loss function ρ can also be used to scale the slack variable (Tsochantaridis et al., 2004):

$$\text{find the } \vec{w}, \xi \text{ that minimizes } \frac{1}{2}\|\vec{w}\|^2 - \frac{C}{n}\sum_{i=1}^{n}\xi_i, \text{ s.t. } \xi_i \geq 0$$

$$\text{s. t. for any } (y \neq y_i) \text{ in the training data, } \vec{w}\Phi(y_i, x) - \vec{w}\Phi(y, x) \geq 1 - \frac{\xi_i}{\rho(y_i, y)}$$

Compared to the perceptron algorithm, SVM learning solves a constrained minimization problem, and typically has much higher complexity.

2.2 Choice of methods for this thesis

The model, the learning algorithm and the decoding algorithm are the most important aspects of a statistical NLP system. The choices are often dependent on each other. For example, a probabilistic model is usually trained with the MLE or ME principles, while a global linear discriminative model can be trained with a CRF, perceptron and other algorithms. Statistical information used by the model often influences the decoder. For example, the independence assumptions for generative models or the definition of feature context for discriminative models determines the time complexity of a dynamic programming algorithm. On the other hand, the speed and accuracy of the decoding algorithm influences the speed and accuracy of learning algorithms such as the perceptron.

We chose the discriminative statistical approach for structural prediction problems. The main theoretical advantage of the discriminative approach over the alternative, generative approach is the freedom to define features that represent important statistical patterns from structural data, without specifying a particular probabilistic generation process based on independence assumptions. Empirically, discriminative models have shown superior performance compared to generative models in most NLP problems.

We chose the generalized perceptron algorithm as the main learning approach. While giving comparable performance to alternative algorithms for discriminative learning such as CRF and SVM, the perceptron typically has less time and space complexity. Being an online learning approach, the perceptron has a simple parameter updating process, and is much faster than its margin-oriented alternatives such as MIRA, which require complex numerical calculations. The difference in time complexity has a practical influence on the training of large linear models. For example, for our joint word segmentation and POS-tagging system, the perceptron algorithm converged in practical running times while the structural SVM algorithm failed to do so.

Perceptron learning is based on the decoding process, and we chose beam-search as the search algorithm for all problems in this thesis. Compared to dynamic programming, which is a common choice for structural prediction problems, beam-search has two important advantages. First, beam-search runs faster: for the decoding problems in this theis, beam-search typically works in linear time in the length of the sentence. Second, there is no limitation to the range of features that can be used with beam-search. Dynamic programming requires the overlapping subproblems and optimal

substructure properties, which limits the range of features that can be used by a discriminative method. For example, in our dependency parsing research, beam-search allowed us to combine two completely different sets of features in a single system. This is difficult to achieve with an efficient dynamic programming decoder. Even with simpler problems such as POS-tagging, a larger range of features leads to slower dynamic programming decoders by increasing the power in a polynomial complexity. The main disadvantage of beam-search is it being approximate, which potentially leads to inferior learning quality than exact inference, and reduced accuracies at test time due to search errors. However, our experiments on word segmentation showed that decoding with beam-search gave no less accuracy than decoding with dynamic programming, with accuracy being increased on some datasets. This can possibly be explained by the self-adjustment of the perceptron, which tunes its parameters according to mistakes made by the decoder.

In summary, the main approach of this thesis is a global discriminative model, trained by the generalized perceptron algorithm and decoded using beam-search. Alternative learning and decoding algorithms were occasionally used, for the purpose of comparison only.

Chapter 3

Word Segmentation

Standard approaches to Chinese word segmentation treat the problem as a tagging task, assigning labels to the characters in the sequence indicating whether the character marks a word boundary. Discriminatively trained models based on local character features are used to make the tagging decisions, with Viterbi decoding finding the highest scoring segmentation. We propose an alternative, word-based segmentor, which uses features based on complete words and word sequences. The generalized perceptron algorithm is used for discriminative training, and we use a beam-search decoder. Closed tests on the first and second SIGHAN bakeoffs show that our system is competitive with the best in the literature, achieving the highest reported F-scores for a number of corpora. Based on the word-based segmentor, we describe further experiments and discuss various aspects of word-based segmentation.

3.1 Introduction and background

Chinese word segmentation (CWS) is the problem of transforming a Chinese sentence from a character sequence to a word sequence. It is an important first step for many NLP tasks including POS-tagging and parsing.

CWS is a process of ambiguity resolution. An important source of ambiguity is out-of-vocabulary (OOV) words. Suppose that the two-character word "清华 (Tsinghua University)" is OOV, while

both "清 (clean water)" and "华 (China)" are in vocabulary as single-character words. Here the segmentor, if it is overly-reliant on knowledge of in-vocabulary words, may incorrectly split "清华" into two words. Typical examples of OOV words include Chinese names, translated foreign names and idioms.

In-vocabulary (IV) words can also be ambiguous. For example, the three characters "这里面" can be segmented as the two words "这里 (here) 面 (flour)" or the two words "这 (here) 里面 (inside)". The ambiguity can be resolved only by using more contextual information. Table 1.1 in Chapter 1 gave more examples of such ambiguities.

Another challenge of CWS is the lack of a fixed standard. It has been shown that there is only about 75% agreement among native speakers about correct segmentation (Sproat et al., 1996). Also, specific NLP tasks may require different segmentation criteria. For example, "北京银行" could be treated as a single word (Bank of Beijing) for machine translation, while it is more naturally segmented into "北京 (Beijing) 银行 (bank)" for tasks such as text-to-speech synthesis.

Many of the first word segmentors were rule-based; they work with a word dictionary and a set of manual rules. A classical example is the *forward maximum matching* (FMM) algorithm, which scans through the input sentence and, starting from the first character, repeatedly matches character sequences to its word dictionary. Once the longest match from the current position is found, it is taken as a word, and matching starts again from the next character. This process is repeated until the last character is considered. As an influential rule-based approach, FMM has several alternatives, such as backward maximum matching and joint forward and backward maximum matching. All are described in Sproat et al. (1996). Other typical rule-based segmentors include Teahan et al. (2000).

The major drawback of rule-based word segmentors is the difficulty in finding enough rules to make effective use of contextual information. Take the forward maximum matching algorithm for example; OOV words tend to be over segmented, simply because there are no matches in the dictionary for OOV words.

The statistical approach to CWS has now become the dominant approach in the literature. These methods work by creating a statistical model from text corpora, and then applying search to find the best segmentations according to the model. The statistical methods can be classified as *supervised* or *unsupervised*. While supervised methods build statistical models from manually annotated corpora,

unsupervised methods gather statistical information from raw text. Supervised learning has been shown to be more effective in standard tests, and we focus on it in this thesis. Typical unsupervised models include automatic clustering, and calculating mutual information between characters. They are normally used on top of a supervised learning model to further improve the accuracy (Liang, 2005).

Following Xue (2003), the standard approach for building a supervised-learning CWS model is to treat CWS as a sequence labeling task. A tag is assigned to each character in the input sentence, indicating whether the character is a single-character word or the start, middle or end of a multi-character word. The context for disambiguation is normally a five-character window with the current character in the middle. In this thesis, we call these methods *character-based* segmentation. The advantage of character-based segmentation is that well-known tagging approaches can be applied directly to the CWS problem.

There are various character-based models in the literature. They differ mainly in the learning algorithm and the features used. Several discriminative learning algorithms have been applied to the character-based systems. Examples include Xue (2003), Peng et al. (2004) and Wang et al. (2006), which use maximum entropy and conditional random field models, and Jiang et al. (2008), which uses the perceptron model. The standard feature set is that defined by Ng and Low (2004), though other feature sets are reported to improve the accuracy (Zhao et al., 2006). Zhao et al. (2006) also showed that the best accuracy for CRF models is given by using a set of six character segmentation tags, which is different from the standard set $\{beginning, middle, end, single\}$ shown previously. Standard search algorithms for sequence tagging have been applied for the decoding process, such as the Viterbi algorithm and beam search.

A disadvantage of character-based models is the use of limited contextual information. For these methods, context is confined to the neighboring characters. Other contextual information, in particular the surrounding words, are not included. Consider the sentence "中国外企业" in Table 1.1 back in Chapter 1, which can be from "其中 (among which) 国外 (foreign) 企业 (companies)", or "中国 (in China) 外企 (foreign companies) 业务 (business)". Note that the five-character window surrounding "外" is the same in both cases, making the tagging decision for that character difficult given the local window. However, the correct decision can be made by comparison of the two

three-word windows containing this character.

In order to explore the potential of explicit word information for CWS, we propose an alternative model that does not map segmentation into a tagging problem. It allows direct use of word information, and therefore we call it *word-based* segmentation. The advantage of this method is that no limitation of information is imposed by the model.[1] We build a word-based segmentor using a global linear model, trained by the generalized perceptron, and use a beam-search decoder. Our segmentor gave competitive accuracies compared to the best systems in the literature on standard SIGHAN data.

We argue that the word-based approach is more flexible compared to the character-based approach. In fact, word-based segmentors can be seen as a superset of character-based segmentors, because of their capability to include character-based features.

In the following sections, we give a description of a word-based segmentation model that has been published in Zhang and Clark (2007), and then present more discussions and experiments concerning various aspects of the word-based segmentor.

3.2 A word-based segmentation algorithm

We build a word-based segmentor using a global linear model, trained by the generalized perceptron algorithm, and a beam-search decoder. In contrast to character-based methods, our word-based model does not map the segmentation problem to a tag sequence learning problem, but defines features on segmented sentences directly. We study several factors that influence the performance of the perceptron word segmentor, including the averaged perceptron method, the size of the beam and the importance of word-based features. We compare the accuracy of our final system to the state-of-the-art CWS systems in the literature using the first and second SIGHAN bakeoff data. Our system is competitive with the best systems, obtaining the highest reported F-scores on a number of the bakeoff corpora. These results demonstrate the importance of word-based features for CWS.

[1] In theory no limitation of information is imposed, but the choice of features should be considered together with the decoder. For example, if dynamic programming is used, the range of features directly affects the efficiency of the decoding algorithm.

3.2.1 The model and the training algorithm

We formulate the CWS problem as finding a mapping from an input sentence $x \in X$ to an output sentence $y \in Y$, where X is the set of possible raw sentences and Y is the set of possible segmented sentences. Given an input sentence x, the predicted output segmentation $y_{out}(x)$ satisfies:

$$y_{out}(x) = \underset{y \in \text{GEN}(x)}{\arg\max}\ \text{Score}(y)$$

where, following Collins (2002), GEN(x) denotes the set of possible outputs for an input sentence x.[2]

The score for a segmented sentence is computed by first mapping it into a set of features. Here a feature is an indicator of the occurrence of a certain pattern in a segmented sentence. For example, a feature can be the occurrence of "读书" as a single word, or the occurrence of "读" separated from "书" in two adjacent words. By defining features, a segmented sentence is mapped into a global feature vector, in which each dimension represents the count of a particular feature in the sentence. If a feature does not occur in a sentence, the corresponding dimension in the global feature vector is zero.

Denote the global feature vector for segmented sentence y by $\Phi(y) \in Z^d$, where d is the total number of features in the model; then Score(y) is computed by the dot product of vector $\Phi(y)$ and a parameter vector $\vec{w} \in Z^d$, where w_i is the weight for the ith feature:

$$\text{Score}(y) = \Phi(y) \cdot \vec{w}$$

The perceptron training algorithm is used to determine the weight values $\vec{w}$.

The training algorithm initializes the parameter vector as all zeros, and updates the vector by decoding the training examples. Each training sentence is turned into the raw input form, and then decoded with the current parameter vector. The output segmented sentence is compared with the original training example. If the output is correct, no change is made to the parameter vector. Hence

[2]Because of the use of beam-search, the actual output is not guaranteed to be exactly the highest scored output in practice. This is also true for the global linear models in Chapters 4, 5 and 6, since we apply beam-search to all the problems studied in this thesis. Other examples in the literature using beam-search as an approximate decoder for a global linear model include Collins and Roark (2004).

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = segment(x_i)$
    if $z_i \neq y_i$
      $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

Figure 3.1: The perceptron learning algorithm for the segmentor

it is a passive training algorithm. If the output is incorrect, the parameter vector is updated by adding the global feature vector of the training example and subtracting the global feature vector of the decoder output. The algorithm can perform multiple passes over the same training sentences. Figure 3.1 gives the algorithm, where N is the number of training sentences and T is the number of passes over the data.

Figure 3.1 is almost identical to the generalized perceptron algorithm shown in Figure 2.5 in Chapter 2. However, because the output sequence y (the segmented sentence) contains all the information from the input sequence x (the raw sentence), the global feature vector $\Phi(x, y)$ in Figure 2.5 is replaced with $\Phi(y)$, which is extracted from the candidate segmented sentences directly.

Another difference between the perceptron training algorithm for the segmentor in Figure 3.1 and the original perceptron training algorithm in Figure 2.5 is that $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot \vec{w}$ is replaced with $z_i = segment(x_i)$, which represents the output from the decoder. Because the decoder can be approximate, $z_i = segment(x_i)$ is not necessarily $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot \vec{w}$ since the highest scoring output may get pruned in the search process.

The averaged perceptron

The perceptron algorithm updates the parameters in order to make correct predictions on the training examples. However, it can overfit the training examples and give less accuracy on unseen data. As introduced in Section 2.1.2, the *averaged perceptron* is a means to reduce overfitting. It uses the averaged value of the parameter vectors after each training example as the final model.

To compute the averaged parameters $\vec{\gamma}$, the training algorithm in Figure 3.1 can be modified by keeping a total parameter vector $\vec{\sigma}$. After the nth training sentence is processed in the tth training

iteration, $\vec{\sigma}^{n,t} = \sum_{n'=1}^{n} \sum_{t'=1}^{t} \vec{w}^{n',t'}$. $\vec{\sigma}$ is updated together with $\vec{w}$ after each training example. After the final iteration, $\vec{\gamma}$ is computed as $\vec{\sigma}^{n,t}/NT$.

With a large number of features, however, calculating the total parameter vector $\vec{\sigma}^{n,t}$ after each training example is expensive. Since the number of changed dimensions in the parameter vector $\vec{w}$ after each training example is a small proportion of the total vector, we use a lazy update optimization for the training process.[3] Define an update vector $\vec{\tau}$ to record the number of the training sentence n and iteration t when each dimension of the averaged parameter vector was last updated. Then after each training sentence is processed, only update the dimensions of the total parameter vector corresponding to the features in the sentence. (Except for the last example in the last iteration, when each dimension of $\vec{\tau}$ is updated, no matter whether the decoder output is correct or not).

Denote the sth dimension in each vector before processing the nth example in the tth iteration as $w_s^{n-1,t}$, $\sigma_s^{n-1,t}$ and $\tau_s^{n-1,t} = (n_{\tau,s}, t_{\tau,s})$. Suppose that the decoder output $z_{n,t}$ is different from the training example y_n. Now $w_s^{n,t}$, $\sigma_s^{n,t}$ and $\tau_s^{n,t}$ can be updated in the following way:

$$\begin{aligned}
\sigma_s^{n,t} &= \sigma_s^{n-1,t} + w_s^{n-1,t} \times (tN + n - t_{\tau,s}N - n_{\tau,s}) \\
w_s^{n,t} &= w_s^{n-1,t} + \Phi(y_n) - \Phi(z_{n,t}) \\
\sigma_s^{n,t} &= \sigma_s^{n,t} + \Phi(y_n) - \Phi(z_{n,t}) \\
\tau_s^{n,t} &= (n, t)
\end{aligned}$$

We found that this lazy update method was significantly faster than the naive method.

3.2.2 The beam-search decoder

The decoder reads characters from the input sentence one at a time, and generates candidate segmentations incrementally. At each stage, the next incoming character is combined with an existing candidate in two different ways to generate new candidates: it is either appended to the last word in the candidate, or taken as the start of a new word. This method guarantees exhaustive generation of possible segmentations for any input sentence.

[3] Daume III (2006) describes a similar algorithm.

```
Input: raw sentence sent – a list of characters
Initialization: set agendas src = [[]], tgt = []
Variables: candidate sentence item – a list of words
Algorithm:
    for index = 0..sent.length−1:
        var char = sent[index]
        foreach item in src:
            // append as a new word to the candidate
            var item1 = item
            item1.append(char.toWord())
            tgt.insert(item1)
            // append the character to the last word
            if item.length > 1:
                var item2 = item
                item2[item2.length−1].append(char)
                tgt.insert(item2)
        tgt.keepFirstB()
        src = tgt
        tgt = []
Outputs: src.best_item
```

Figure 3.2: The decoding algorithm for the agenda based word segmentor

Two agendas are used: the source agenda and the target agenda. Initially the source agenda contains an empty sentence and the target agenda is empty. At each processing stage, the decoder reads in a character from the input sentence, combines it with each candidate in the source agenda and puts the generated candidates onto the target agenda. After each character is processed, the items in the target agenda are copied to the source agenda, and then the target agenda is cleared, so that the newly generated candidates can be combined with the next incoming character to generate new candidates. After the last character is processed, the decoder returns the candidate with the best score in the source agenda. Figure 3.2 gives the decoding algorithm.

For a sentence with length l, there are 2^{l-1} different possible segmentations. To guarantee reasonable running speed, the size of the target agenda is limited, keeping only the B best candidates.

3.2.3 Feature templates

The feature templates are shown in Table 3.1. Features 1 and 2 contain only word information, 3 to 5 contain character and length information, 6 and 7 contain only character information, 8 to 12 contain word and character information, while 13 and 14 contain word and length information. To

1	word w
2	word bigram $w_1 w_2$
3	single-character word w
4	a word starting with character c and having length l
5	a word ending with character c and having length l
6	space-separated characters c_1 and c_2
7	character bigram $c_1 c_2$ in any word
8	the first and last characters c_1 and c_2 of any word
9	word w immediately before character c
10	character c immediately before word w
11	the starting characters c_1 and c_2 of two consecutive words
12	the ending characters c_1 and c_2 of two consecutive words
13	a word of length l and the previous word w
14	a word of length l and the next word w

Table 3.1: Feature templates for the agenda based word segmentor

reduce overfitting, the length features are normalized to 7, treating any larger values as having the value 7. Any segmented sentence is mapped to a global feature vector according to these templates. There are $356,337$ features with non-zero values after 6 training iterations using the development data.

For this particular feature set, the longest range features are word bigrams. Therefore, among partial candidates ending with the same bigram, the best one will also be in the best final candidate. The decoder can be optimized accordingly: when an incoming character is combined with candidate items as a new word, only the best candidate is kept among those having the same last word.

3.2.4 Experiments

Two sets of experiments were conducted for the word-based segmentor. The first, used for development, was based on the part of Chinese Treebank 4 that is not in Chinese Treebank 3. This corpus contains 240K characters (150K words and 4798 sentences). 80% of the sentences (3813) were randomly chosen for training and the rest (985 sentences) were used as development testing data. The convergence of the non-averaged and averaged perceptron was observed and compared. The influence of particular features and the agenda size was also studied.

The second set of experiments used training and testing sets from the first and second international Chinese word segmentation bakeoffs (Sproat and Emerson, 2003; Emerson, 2005). The accuracies are compared to other models in the literature.

Iteration	1	2	3	4	5	6	7	8	9	10
P (non-avg)	89.0	91.6	92.0	92.3	92.5	92.5	92.5	92.7	92.6	92.6
R (non-avg)	88.3	91.4	92.2	92.6	92.7	92.8	93.0	93.0	93.1	93.2
F (non-avg)	88.6	91.5	92.1	92.5	92.6	92.6	92.7	92.8	92.8	92.9
P (avg)	91.7	92.8	93.1	92.2	93.1	93.2	93.2	93.2	93.2	93.2
R (avg)	91.6	92.9	93.3	93.4	93.4	93.5	93.5	93.5	93.6	93.6
F (avg)	91.6	92.9	93.2	93.3	93.3	93.4	93.3	93.3	93.4	93.4
#Incorrect sentences	3401	1652	945	621	463	288	217	176	151	139

Table 3.2: The accuracy using non-averaged and averaged perceptron
P – precision (%), R – recall (%), F – F-measure.

B	2	4	8	16	32	64	128	256	512	1024
Tr	660	610	683	830	1111	1645	2545	4922	9104	15598
Seg	18.65	18.18	28.85	26.52	36.58	56.45	95.45	173.38	325.99	559.87
F	86.90	92.95	93.33	93.38	93.25	93.29	93.19	93.07	93.24	93.34

Table 3.3: The influence of agenda size
B - agenda size, Tr - training time (seconds), Seg - testing time (seconds), F - F-measure.

F-measure is used as the accuracy measure. Define precision p as the percentage of words in the decoder output that are segmented correctly, and recall r as the percentage of gold standard output words that are correctly segmented by the decoder. The (balanced) F-measure is $2pr/(p+r)$.

CWS systems are evaluated using two types of tests. The *closed* tests require that the system is trained only with a designated training corpus. Any extra knowledge is not allowed, including common surnames, Chinese and Arabic numbers, European letters, lexicons, part-of-speech, semantics and so on. The *open* tests do not impose such restrictions.

Open tests measure a model's capability to utilize extra information and domain knowledge, which can lead to improved performance, but since this extra information is not standardized, direct comparison between open test results is more difficult.

The convergence of the perceptron

In this experiment, the agenda size was set to 16 for both training and testing. Table 3.2 shows the precision, recall and F-measure for the development set after 1 to 10 training iterations, as well as the number of mistakes made in each iteration. Here a mistake refers to a sentence with any incorrectly segmented word. The accuracy curves by different numbers of training iterations for both the non-averaged and averaged perceptron are given in Figure 3.3.

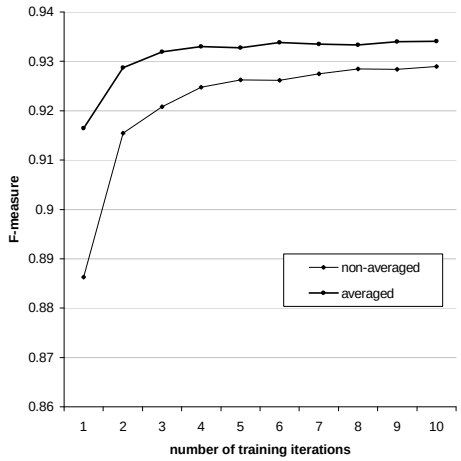


Figure 3.3: The accuracy curves of the averaged and non-averaged perceptron algorithms

The number of mistakes made in each iteration decreases when the number of training iteration increases, reflecting the convergence of the learning algorithm. The averaged perceptron improves the segmentation accuracy at each iteration, compared with the non-averaged perceptron. The accuracy curve was used to fix the number of training iterations at 6 for the remaining experiments.

The influence of agenda size

Reducing the agenda size increases the decoding speed, but it could cause loss of accuracy by eliminating potentially good candidates. The agenda size also affects the training time, and resulting model, since the perceptron training algorithm uses the decoder output to adjust the model parameters. Table 3.3 shows the accuracies with ten different agenda sizes, each used for both training and testing.

Accuracy does not increase beyond $B = 16$. Moreover, the accuracy is quite competitive even with B as low as 4. This reflects the fact that the best segmentation is often within the current top few candidates in the agenda.[4] Since the training and testing time generally increases as N increases, the agenda size is fixed to 16 for the remaining experiments.

[4]The optimization in Section 3.2.3, which has a pruning effect, was applied to this experiment. Similar observations were made in separate experiments without such optimization.

Features	F	Features	F
All	93.38	w/o 1	92.88
w/o 2	93.36	w/o 3, 4, 5	92.72
w/o 6	93.13	w/o 7	93.13
w/o 8	93.14	w/o 9, 10	93.31
w/o 11, 12	93.38	w/o 13, 14	93.23

Table 3.4: The influence of features
F: F-measure. Feature numbers are from Table 3.1

The influence of particular features

Our CWS model is highly dependent upon word information. Most of the features in Table 3.1 are related to words. Table 3.4 shows the accuracy with various features from the model removed.

Among the features, vocabulary words (feature 1) and length prediction by characters (features 3 to 5) showed strong influence on the accuracy, while word bigrams (feature 2) and special characters in them (features 11 and 12) showed comparatively weak influence.

Closed test on the sighan bakeoffs

Four training and testing corpora were used in the first bakeoff (Sproat and Emerson, 2003), including the Academia Sinica Corpus (AS), the Penn Chinese Treebank Corpus (CTB), the Hong Kong City University Corpus (CU) and the Peking University Corpus (PU). However, because the testing data from the Penn Chinese Treebank Corpus is currently unavailable, we excluded this corpus. The corpora are encoded in GB (PU, CTB) and BIG5 (AS, CU), which use different ways to encode the same character into machine bytes. In order to test them consistently in our system, they are all converted to UTF8 encoding without loss of information.

The results are shown in Table 3.5. We follow the format from Peng et al. (2004). Each row represents a CWS model. The first eight rows represent models from Sproat and Emerson (2003) that participated in at least one closed test from the table, row "Peng" represents the CRF model from Peng et al. (2004), and the last row represents our model. The first three columns represent tests with the AS, CU and PU corpora, respectively. The best score in each column is shown in bold. The last two columns represent the average accuracy of each model over the tests it participated in (SAV), and our average over the same tests (OAV), respectively. For each row the best average is shown in bold.

	AS	CU	PU	SAV	OAV
S01	93.8	90.1	**95.1**	93.0	**95.0**
S04			93.9	93.9	**94.0**
S05	94.2		89.4	91.8	**95.3**
S06	94.5	92.4	92.4	93.1	**95.0**
S08		90.4	93.6	92.0	**94.3**
S09	96.1		94.6	**95.4**	95.3
S10			94.7	**94.7**	94.0
S12	95.9	91.6		93.8	**95.6**
Peng	95.6	92.8	94.1	94.2	**95.0**
	96.5	**94.6**	94.0		

Table 3.5: The accuracies over the first SIGHAN bakeoff data

	AS	CU	PK	MR	SAV	OAV
S14	94.7	94.3	95.0	96.4	95.1	**95.4**
S15b	**95.2**	94.1	94.1	95.8	94.8	**95.4**
S27	94.5	94.0	95.0	96.0	94.9	**95.4**
Zh-a	94.7	94.6	94.5	96.4	95.1	**95.4**
Zh-b	95.1	**95.1**	**95.1**	97.1	**95.6**	95.4
	94.6	**95.1**	94.5	**97.2**		

Table 3.6: The accuracies over the second SIGHAN bakeoff data

We achieved the best accuracy in two of the three corpora, and better overall accuracy than the majority of the other models. The average score of system S10 is 0.7% higher than our model, but S10 only participated in the PU test.

Four training and testing corpora were used in the second bakeoff (Emerson, 2005), including the Academia Sinica corpus (AS), the Hong Kong City University Corpus (CU), the Peking University Corpus (PK) and the Microsoft Research Corpus (MR). Different encodings were provided, and the UTF8 data for all four corpora were used in this experiment.

Following the format of Table 3.5, the results for this bakeoff are shown in Table 3.6. We chose the three models that achieved at least one best score in the closed tests from Emerson (2005), as well as the sub-word-based model of Zhang et al. (2006) for comparison. Row "Zh-a" and "Zh-b" represent the pure sub-word CRF model and the confidence-based combination of the CRF and rule-based models from Zhang et al. (2006), respectively. The last row represents our model.

Again, our model achieved better overall accuracy than the majority of the other models. One system to achieve comparable accuracy with our system is Zh-b, which improves upon the sub-word CRF model (Zh-a) by combining it with an independent dictionary-based submodel and improving

the accuracy of known words. In comparison, our system is based on a single perceptron model.

In summary, closed tests for both the first and the second bakeoff showed competitive results for our system compared with the best results in the literature. Our word-based system achieved the best F-measures over the AS (96.5%) and CU (94.6%) corpora in the first bakeoff, and the CU (95.1%) and MR (97.2%) corpora in the second bakeoff.

3.3 Further studies of word-based segmentation

The previous section described the word segmentor we proposed in Zhang and Clark (2007). The main contribution of the segmentor is the use of a word-based, global linear model, which allows arbitrary features to be defined for CWS.

There are alternative configurations for the word-based segmentor. For example, structural SVM or MIRA can be used as alternative learning algorithms to the generalized perceptron, and dynamic programming can be applied rather than beam-search. In this section, we study more details of the various parts of the word segmentor, giving more discussion and experiments related to the learning algorithm in Section 3.3.1, and the decoding algorithm in Section 3.3.2.

Open knowledge that is not in the training data has been shown to improve the accuracy of CWS. Such knowledge includes information about special characters (Ng and Low, 2004), character clustering information (Shi and Wang, 2007; Liang, 2005) and character mutual information (Liang, 2005). We study a method to include rule-based knowledge in the statistical system using the perceptron algorithm in Section 3.3.3.

3.3.1 The training algorithm

We used the averaged perceptron to train the parameters for the word-based model. During training, *negative features* (i.e. those features that occur only in the incorrect parser outputs, but not in the training data) are built into the model. We did further experiments to observe the effect of negative features, by removing them from the system. The observation was that negative features had a tiny positive impact on the accuracies. Because the number of features for the CWS task is comparatively small, we conclude that negative features should be used.

	Bakeoff 1						Bakeoff 2									
	AS		CU		PU		AS		CU		PU		MS		Avg	
M	Ac	It	Ac	It	Ac	It	Ac	It	Ac	It	Ac	It	Ac	It	Ac	It
1	96.5	6	**94.6**	6	94.0	6	94.6	6	95.1	6	94.5	6	97.2	6	95.2	6
2	**96.9**	**3**	94.5	**5**	94.0	**5**	**95.0**	**3**	95.1	**5**	94.5	**5**	97.2	5	95.3	**4**
3a	**96.9**	4	94.5	**5**	**94.2**	8	**95.0**	4	95.1	9	**94.6**	8	97.2	5	**95.4**	6
3b	**96.9**	4	**94.6**	7	94.1	7	**95.0**	4	95.1	8	**94.6**	9	97.2	10	**95.4**	7

Table 3.7: A comparison of different methods to determine the number of training iterations
M - method, Ac - Accuracy, It - the number of iterations, Avg - Average
Method 1: Fix the number of training iterations according to development test
Method 2: Stop training when the number of errors is below 7.6% training data
Method 3a: Set aside one in every 10 sentences from the training data and draw the accuracy curve
Method 3b: Set aside the first 10% of the training data and draw the accuracy curve

Another approach to reduce the number of features is setting a frequency threshold, so that only features with a frequency higher than the threshold are included in the system. Collins (2002) experimented with a threshold value of 5, and showed that it had a negative impact on the accuracy. We therefore did not set such a threshold.

In our previous experiments, the number of training iterations is fixed to 6 according to the development test. An alternative way to decide the number of training iterations is to use the convergence of the perceptron: when the number of incorrect outputs is small enough in a training iteration, we stop the training process to prevent overfitting on the training data. Using this method, the number of training iterations is adjusted flexibly according to different training data. Another alternative method is to adjust the number of training iterations by experiment: instead of using a separate set of development data, we split the training data into development training and testing sets, and then find the best iteration number from the corresponding accuracy curve on the development data. This method was used by Carreras et al. (2006) in their parsing model. The following subsection gives details of our experiments comparing the effect of the above methods.

Deciding the number of training iterations

We compare the effect of different methods to decide the number of training iterations by using the first and second international bakeoff data. The results are shown in Table 3.7. Method 1 is the same method that was applied in our previous experiments. For method 2, we find a fixed ratio between the number of training errors and the total number of training examples. When the number

of training errors falls below this ratio in a training iteration, the training process is stopped. By observation from the development test (Table 3.2 and Figure 3.3), this ratio is set to 288/3813 = 7.6% (data from iteration 6). For method 3, 10% of the training examples are extracted from each dataset as the development test examples. Two different ways of sampling are experimented. Method 3a extracts one in every 10 training sentences, while method 3b takes the first 10% of the training examples.

For each dataset, the highest accuracy and the smallest number of training iterations are highlighted with bold font. It can be seen from the table that both Method 2 and Method 3 performed better than Method 1, while Method 3 achieved slightly higher accuracy than Method 2. The two sampling methods for Method 3 gave almost identical results. The effect of both Method 1 and Method 2 is dependent on a separate set of development data. In this experiment, Method 2 took the smallest number of training iterations for every dataset.

As a conclusion, in order to decide the number of training iterations for different datasets, the best choice for optimal accuracy is experimenting with the training data in each dataset separately (Method 3). On the other hand, when there are many training and testing datasets, this method requires much more experimental effort than using a separate set of development data (Method 1 and Method 2). When separate development data are used, the ratio of training errors (Method 2) is more informative than the absolute iteration number (Method 1) to help estimate the best number of training iterations for multiple sets of test data. It can be used as a time-saving alternative to the optimal approach.

3.3.2 The decoding algorithm

In the word-based segmentor in Section 3.2, a single agenda (Figure 3.2) is used to limit the size of the search space. The method showed reasonable accuracy with a small agenda size. However, comparisons have been made only between different agenda sizes for the same decoder. In this section, we compare the effect of different decoders.

Input: raw sentence *sent* – a list of characters
Variables: candidate sentence *item* – a list of words
max word-length record *maxlen*
the list of agendas *agendas*
Initialization: $agendas[0] = [\text{“”}]$; $agenda[i] = [],\ i \neq 0$
Algorithm:
for $index = 1$ .. $sent$.length:
for $start\_index = \max(0..index - maxlen + 1)$:
$word = sent[start\_index..index]$
for $item \in agendas[start\_index]$:
$item_1 = item$
$item_1$.append($word$)
$agendas[index]$.insert($item_1$)
$agendas[index]$.B_best_cutoff()
Outputs: $agendas[sent$.length$]$.best_item

Figure 3.4: The multiple beam decoding algorithm

Method	Training time	Decoding time
Beam	53 sec	8 sec
Multiple beam	353 sec	73 sec

Table 3.8: Speed comparison of the single and multiple beam decoders for the word segmentor

A decoder with multiple beams

We propose a multiple beam decoder as shown in Figure 3.4. In order to enlarge the search space, an agenda is kept for each character in the input sentence, recording the best partial candidates ending with the character. Like the single beam decoder, the input sentence is processed incrementally. However, at each stage, partial sequence candidates are available at all previous characters. Therefore, the decoder can examine all possible words ending with the current character. These possible words are combined with the relevant partial candidates from the previous agendas to generate new candidates, which are then inserted into the current agenda. The output of the decoder is the top candidate in the last agenda, representing the best segmentation for the whole sentence. To improve the running speed, a maximum word length record is kept to limit the length of candidate words.

We chose 16 for the size of each agenda, and used exactly the same features as shown in Table 3.1, so that a direct comparison between the two decoders can be made. The training (one iteration) and decoding time of the single-beam and multiple-beam search decoders are shown in Table 3.8. As expected, the multiple beam decoder is slower.

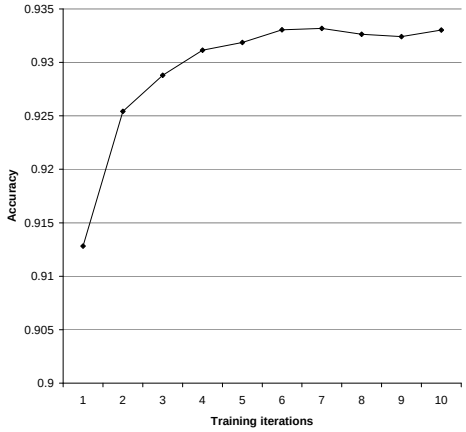


Figure 3.5: The perceptron convergence with the multiple beam decoder

	Bakeoff 1			Bakeoff 2				
	AS	CU	PU	AS	CU	PU	MS	Average
Beam	96.9	**94.5**	94.0	95.0	95.1	**94.5**	97.2	95.3
Multiple beam	**97.0**	94.4	**94.5**	**95.1**	95.1	94.4	97.2	**95.4**

Table 3.9: Accuracy comparison of the single and multiple beam decoders for the segmentor

The accuracies of the two decoders are compared using the standard training and testing data from the international bakeoffs. Considering both the accuracy and the experimental effort, we use Method 2 in Table 3.7 (the ratio of training errors) to determine the number of training iterations for each dataset. The accuracy curve for the development data is shown in Figure 3.5. According to this figure, we set the threshold ratio to be $204/3813 = 0.054$ (the 7th iteration from the development test).

Test results using the first and second international bakeoff data are shown in Table 3.9, following the format of the previous sections. Compared to the single beam decoder, the multiple beam decoder achieved higher accuracy with the AS and PU corpora in the first bakeoff, and the AS corpus in the second bakeoff. However, it performs slightly worse with the CU corpus in the first bakeoff and the PU corpus in the second bakeoff. The overall accuracy is slightly improved over the single beam decoder.

In conclusion, the multiple beam decoder gave small improvements on the accuracies, but due to the much larger search space, it is much slower than the single beam decoder. It provides an alternative choice for the word segmentor. However, for the joint POS tagging problem in Chapter 4,

a very large search space is inevitable. There the advantage of the multiple beam search method will be shown clearly, for the single beam search method can not give comparable accuracy.

Dynamic programming

Given the feature set from Table 3.1, a dynamic programming decoder with $O(n^2)$ time complexity can be implemented. The potential advantage of the dynamic programming decoding algorithm over single-beam or multiple-beam search is that the search is exact. We used the same feature templates and tested the accuracies of this decoder, but did not find significant improvement in accuracy. In addition, the dynamic programming decoder ran much slower than the multiple-beam search decoder. Therefore, we conclude that beam-search is a competitive choice in both speed and accuracy for the word segmentor.

Discriminating full words and partial words during decoding

A potential limitation of the beam search decoding algorithm in Figure 3.2 is the simple scoring mechanism for partial sequences, treating partial words equally as full words. Here a partial word refers to a part of a full word. For example, "加拿大 (Canada)" is a full word, and "加" and "加拿" are partial words from it. While "加拿" does not make any sense as a word, "加 (add)" can be interpreted as a single-character word. At each processing stage, the decoder compares candidate items ending with the same character, and keeps the B best according to the score. Since the last word in a candidate can be a partial word such as "加拿", it is possible for a potentially correct candidate to be discarded from the agenda because of a low score from the partial ending word.

One solution to this problem is to take partial words and full words differently in score assignments. In particular, full words are scored in exactly the same way as the original decoder (using all feature templates from Table 3.1), while partial words are scored only according to the existing information (feature template 7 from Table 3.1). Correspondingly, candidate items in the agenda need to be classified into two types – full items and partial items, according to their ending words. To give some heuristic about the potential of candidates, features are extracted from the current sequence as well as the next character. Therefore, the features for a full item also include templates 6, 9 and 11 from the last word, and the features for a partial item also include template 7 from the

	Bakeoff 1			Bakeoff 2				
	AS	CU	PU	AS	CU	PU	MS	Average
original	96.9	94.5	94.0	95.0	**95.1**	**94.5**	**97.2**	95.3
discriminating full/partial words	**97.0**	**94.6**	**94.2**	**95.1**	95.0	94.4	97.1	95.3

Table 3.10: Discriminating partial words and full words in the single beam decoder

	Bakeoff 1			Bakeoff 2				
	AS	CU	PU	AS	CU	PU	MS	Average
w/o knowledge	96.9	94.5	94.0	95.0	95.1	94.5	97.2	95.3
with knowledge	96.9	94.5	**94.8**	**95.3**	95.1	**94.7**	97.2	**95.5**

Table 3.11: Knowledge about English letters and Arabic numbers

last word.

The data from the first and second international bakeoffs are used to test the effect of the new decoding algorithm. Method 2 in Table 3.7 was used to determine the number of training iterations. By a development test, the ratio of training errors is set to 0.050. It can be seen from Table 3.10 that by discriminating full words and partial words, the accuracy increased for four out of the seven datasets, while dropped for the rest. The averaged accuracy is slightly better than the original decoding algorithm, but not significant.

3.3.3 Knowledge of English letters and Arabic numbers

The previous sections studied various aspects of the word-based segmentation model using closed-set knowledge – knowledge from the training data. However, there are many sources of open knowledge that can help segmentation. For example, knowledge about foreign letters, numbers and common surnames have been widely used in word segmentation. Moreover, semantic knowledge has also been applied to improve the accuracy (Shi and Wang, 2007). In this section, we explore the application of specific knowledge to the word-based segmentation algorithm.

Foreign languages can be embedded in Chinese sentences, typical examples being English (or European) letters and Arabic numbers. In news articles, for example, they are often used in original name references, website links, technical terms, quoted speech, telephone numbers and IDs. From the second international segmentation bakeoff, the proportions of sentences that contain English letters and Arabic numbers in the four training datasets are 13861/708953 = 2.0% (Academia Sinica), 3096/53019 = 5.8% (Hong Kong City University), 1097/86959 = 1.3% (Microsoft Research) and

517/43018 = 1.2% (Peking University), respectively.

The segmentation of foreign letters should follow rules from the original languages. For example, English words are explicitly separated, and adjacent letters are never segmented. The same is true for Arabic numbers. Consequently, knowledge about foreign letters can help a word segmentor to treat them differently from Chinese characters, therefore improving the segmentation accuracy.

Knowledge of foreign letters can be built into features in a pure statistical system (Ng and Low, 2004). For example, tokens in an input sentence can be classified into different categories, including Arabic numbers, Chinese characters and English letters. N-grams of such categories can then be used as features in the word segmentor. However, because the linguistic rules for segmenting English letters and Arabic numbers are very clear, it is more straightforward to build these rules into the segmentor directly. While the processing of Chinese characters stays unchanged, foreign letters are processed by rules separately. This method not only guarantees correct segmentation between foreign letters, but also reduces the size of the statistical data.

In this section we add knowledge to the statistical word segmentor in Section 3.2.2 by using the following rules:

(1) Adjoining English letters and Arabic numbers in the input sentence are not separated;

(2) Spaces between English words and Arabic numbers mark word boundaries.

The above rules can be incorporated into the statistical segmentor, by modifications to the decoder. In particular, when an input sentence is processed, the rules are used to prune wrong candidates. Because the perceptron learning algorithm is based on the decoding process, the statistical system will collect information for only the correctly segmented foreign words.

Table 3.11 shows the comparison with the pure statistical model. As can be seen from the table, the use of knowledge and rules improves the accuracy for 3 out of 7 the datasets, giving an absolute increase of 0.2% on the overall accuracy. It is worth noticing that in some of the datasets, spaces between English letters are removed from the input, which may explain why rules did not help in these cases.

3.4 Conclusion and discussion

We proposed a word-based CWS model as an alternative to the existing character-based tagging models, which allows word information to be used as features. We built a word-based segmentor using the discriminative perceptron learning algorithm and a beam-search decoder. Closed tests using the first and second SIGHAN CWS bakeoff data demonstrated our system to be competitive with the best in the literature.

Based on the word segmentor we built, we explored alternative decoding algorithms, different configurations for training, and the ability to incorporate extra knowledge besides the training data. These experiments showed the flexibility of word-based segmentation.

Our classification of character-based and word-based approaches is not strict. Character-based systems can also make limited use of word information. For example, Ng and Low (2004) incorporate full word information into a feature, and Zhang et al. (2006) combine a CRF and a rule-based model. Unlike the character-tagging models, the CRF submodel assigns tags to *sub-words*, which include single-character words and the most frequent multiple-character words from the training corpus. Thus it can be seen as a step towards a word-based model. However, sub-words do not necessarily contain full word information. Moreover, sub-word extraction is performed separately from feature extraction.

In conclusion, we showed that a word-based segmentor can achieve comparable accuracy to the best character-based segmentors. Moreover, the word-based approach is a direct solution to the CWS problem, having the generality to include any character-based features.

Chapter 4

Part-of-speech Tagging

Traditionally, Chinese word segmentation and POS-tagging are performed in a pipeline. The output from the word segmentor is taken as the input for the POS-tagger. A disadvantage of pipelined segmentation and POS-tagging is that POS-tag information, which is potentially useful for segmentation, is not used during the segmentation step. In addition, word segmentation errors are propagated to the POS-tagger, leading to lower quality of the overall segmented and POS-tagged output. We propose a global linear model that performs word segmentation and POS-tagging simultaneously, and show that it outperforms a pipelined baseline significantly using the same feature templates. In cross-validation tests using CTB, our joint segmentor and POS-tagger gave overall segmentation and tagging accuracies that are comparable to the best reported in the literature.

4.1 Introduction and background

Parts-of-speech (POS) represent the basic lexical-grammatical information relating to words in a sentence. The definition of POS depends on a specific language or grammar; common Chinese POS include noun, verb, adjective, adverb, preposition, and measure word (Huang and Liao, 2002). The set of POS used by this thesis is defined by the Penn Chinese Treebank (Xia, 2000), the training data for the statistical models.

A POS-*tagger* assigns a part-of-speech tag to each word in the input sentence. Since POS-tagging is

a typical sequence labeling problem, the HMM, MEMM and CRF models introduced back in Chapter 2 can be applied directly to it, with the input words being the observation sequence and POS-tags being the states.

Besides the standard labeling models, an early model for POS-tagging is the maximum entropy tagger of Ratnaparkhi (1996). The Ratnaparkhi tagger calculates the conditional probability of each POS-tag separately, based on a trigram ME model. Features are extracted from the five-word window with the current word at the middle and the three-tag window with the current tag at the end. The Ratnaparkhi (1996) tagger can be seen as similar to an MEMM model, but with a wider range of features.

More recent statistical POS-taggers have made improvements on the English Penn Treebank data by using different statistical models, including the CRF (Lafferty et al., 2001), the generalized perceptron (Collins, 2002) and SVM (Giménez and Màrquez, 2003).

Traditional POS-taggers process an input sentence from left to right, assuming that each POS-tag is dependent only on its predecessors. However, sometimes it is easier to decide a POS-tag when its successors have been given. The recent bidirectional POS-taggers (Toutanova et al., 2003; Tsuruoka and Tsujii, 2005) make use of both left-to-right and the right-to-left POS dependencies for POS-tagging. Instead of working from left to right, the Tsuruoka and Tsujii (2005) tagger assigns POS-tags in a best-first order, tagging the word with the smallest POS ambiguity at each processing stage, given the current contextual information. For example, initially when an input sentence is given, no POS-tags are available in the sentence, and the first tag is assigned to the word with the smallest POS ambiguity according to word information only. After a POS-tag is assigned, it provides contextual information to its neighboring words on both sides. The tagging process is repeated until all input words are tagged.

Shen et al. (2007) extended the idea of bidirectional POS-taggers further by developing a learning algorithm that supports the best-first decoding process. Shen et al. (2007) applied a perceptron training algorithm to guide a bidirectional beam-search decoder, and reported the current best accuracy in POS-tagging on the Penn Treebank data using a single model.

The aforementioned POS-tagging algorithms assume that the input sentences have explicit word boundaries, and therefore can be applied to Chinese only after word segmentation is performed.

However, as discussed in the previous chapter, word segmentation is not a trivial task. State-of-the-art word segmentors have 90% to 98% accuracy on standard training and testing data, and the error rate significantly increases for out-of-domain data. Segmentation errors will propagate to the pipelined POS-tagger, reducing the quality of POS-tagged outputs.

On the other hand, POS-information can be used to improve Chinese word segmentation. For example, the POS + word pattern "number word" + "个 (a common measure word)" can help in segmenting the character sequence "一个人" into the word sequence "一 (one) 个 (measure word) 人 (person)" instead of "一 (one) 个人 (personal; adjective)". Moreover, the comparatively rare POS pattern "number word" + "number word" can help to prevent segmenting a long number word into two words.

In this chapter, we study Chinese POS-tagging by considering word segmentation and POS-tagging simultaneously. To avoid error propagation and make use of POS information in word segmentation, segmentation and POS-tagging can be viewed as a single task: given an input sentence as a character sequence, a joint segmentor and POS-tagger considers all possible segmented and tagged sequences, and chooses the overall best output by using both segmentation and POS information. A major challenge for such a joint system is the large search space. For an input sentence with n characters, the number of possible output sequences is $O(2^{n-1} \cdot T^n)$, where T is the size of the tag set. Due to the nature of the combined candidate items, decoding can be inefficient even with dynamic programming.

We propose a novel joint solution for Chinese word segmentation and POS-tagging. In order to decode efficiently, a novel beam search algorithm is applied. We use a discriminative joint model to score segmented and POS-tagged candidate outputs, where features are extracted from words and POS-tags simultaneously, and trained consistently by a single generalized perceptron. In experiments with the Chinese Treebank data, our joint system gave an error reduction of 14.6% in segmentation accuracy and 12.2% in the overall segmentation and tagging accuracy, compared to the traditional pipeline approach. In addition, our overall segmentation and POS-tagging accuracies are comparable to the best in the literature, which exploit knowledge outside the training data, even though our system is fully data-driven.

In the following sections, we give the details of the pipelined word segmentor and POS-tagger, and the joint segmentor and POS-tagger, and report the experimental results. Most of the following

1	word w
2	word bigram w_1w_2
3	single-character word w
4	a word of length l with starting character c
5	a word of length l with ending character c
6	space-separated characters c_1 and c_2
7	character bigram c_1c_2 in any word
8	the first / last characters c_1 / c_2 of any word
9	word w immediately before character c
10	character c immediately before word w
11	the starting characters c_1 and c_2 of two consecutive words
12	the ending characters c_1 and c_2 of two consecutive words
13	a word of length l with previous word w
14	a word of length l with next word w

Table 4.1: Feature templates for the baseline word segmentor

content has been previously published (Zhang and Clark, 2008a).

4.2 The baseline system

In this section, we describe the pipelined word segmentor and POS-tagger that serves as our *baseline system*. We use *baseline segmentor* and *baseline* POS-*tagger* to refer to the word segmentor which does only segmentation, and the POS-tagger which performs POS-tagging on segmented input sentences, in the baseline system.

We use the word-based segmentor from Chapter 3 as the baseline segmentor, and implement a perceptron POS-tagging system similar to the tagger from Collins (2002). Our baseline segmentor is the same as that described in Section 3.2, and the feature templates are shown in Table 4.1 (repeated from Table 3.1). The features are extracted from word bigrams, capturing word, word length and character information in the context. Similarly to the last chapter, the word length features are normalized. However, in contrast to the last chapter, the threshold for normalization is set to 15 according to development tests, and any word length larger than 15 is normalized to 15.

Our baseline POS-tagger is based on the perceptron model. Given a segmented input sentence x, the tagged output $y_{out}(x)$ satisfies:

$$y_{out}(x) = \arg\max_{y \in \mathrm{GEN}(x)} \mathrm{Score}(y)$$

1	tag t with word w
2	tag bigram t_1t_2
3	tag trigram $t_1t_2t_3$
4	tag t followed by word w
5	word w followed by tag t
6	word w with tag t and previous character c
7	word w with tag t and next character c
8	tag t on single-character word w in character trigram c_1wc_2
9	tag t on a word starting with char c
10	tag t on a word ending with char c
11	tag t on a word containing char c (not the starting or ending character)
12	tag t on a word starting with char c_0 and containing char c
13	tag t on a word ending with char c_0 and containing char c
14	tag t on a word containing repeated char cc
15	tag t on a word starting with character category g
16	tag t on a word ending with character category g

Table 4.2: Feature templates for the baseline POS-tagger

where GEN(x) represents the set of possible outputs for x. The averaged perceptron learning algorithm and beam-search are applied to the training and decoding, respectively.

We define a set of features for the baseline POS-tagger according to previous research on Chinese POS-tagging, as shown in Table 4.2. The contextual information includes the POS-tag trigram ending with the current tag, and the neighboring three-word window with the current word in the middle. To reduce overfitting and increase the decoding speed, templates 4, 5, 6 and 7 only include words with less than 3 characters. Like the baseline segmentor, the baseline tagger uses normalized word length features.

Templates 15 and 16 in Table 4.2 are inspired by the CTBMorph feature templates of Tseng et al. (2005), which gave the most accuracy improvement in their experiments. Here the category of a character is the set of tags seen on the character during training. Other morphological features of Tseng et al. (2005) are not used because they require extra web corpora besides the training data.

During training, the baseline POS-tagger stores special word-tag pairs into a *tag dictionary* (Ratnaparkhi, 1996), which is used by the decoder to prune unlikely tags. For each word occurring more than N times in the training data, the decoder can only assign a tag the word has been seen with in the training data. This method led to improvement in the decoding speed as well as the output accuracy for English POS-tagging (Ratnaparkhi, 1996). Besides tags for frequent words, our baseline POS-tagger also uses the tag dictionary to store closed-set tags (Xia, 2000) – those associated only

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = segment\_tag(x_i)$
    if $z_i \neq y_i$
      $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

Figure 4.1: The generalized perceptron learning algorithm for the joint segmentor and POS-tagger

with a limited number of Chinese words.

4.3 The joint segmentor and POS-tagger

We build the joint word segmentation and POS-tagging system using exactly the same source of information as the baseline system, by applying the feature templates from the baseline word segmentor and POS-tagger. No extra knowledge is used by the joint model. However, because word segmentation and POS-tagging are performed simultaneously, POS information participates in word segmentation.

4.3.1 Formulation of the joint model

We formulate joint word segmentation and POS-tagging as a single problem, which maps a raw Chinese sentence to a segmented and POS tagged output. Given an input sentence x, the output $y_{out}(x)$ satisfies:

$$y_{out}(x) = \arg\max_{y \in \mathrm{GEN}(x)} \mathrm{Score}(y)$$

where $\mathrm{GEN}(x)$ represents the set of possible outputs for x.

Score(y) is computed by a global linear model. Denoting the global feature vector for the tagged sentence y by $\Phi(y)$, we have:

$$\mathrm{Score}(y) = \Phi(y) \cdot \vec{w}$$

where $\vec{w}$ is the parameter vector in the model. Each element in $\vec{w}$ gives a weight to its corresponding element in $\Phi(y)$, which is the count of a particular feature over the whole sentence y. We calculate

the $\vec{w}$ value by supervised learning, using the averaged perceptron algorithm given in Figure 4.1.[1] This algorithm is similar to the perceptron algorithm given in Figure 2.5, except that z_i is calculated by the decoder output $z_i = segment\_tag(x_i)$ instead of the exact highest scored output $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot \vec{w}$, for the same reason as given in the last chapter.

We take the union of feature templates from the baseline segmentor (Table 4.1) and POS-tagger (Table 4.2) as the feature templates for the joint system. All features are treated equally and processed together according to the linear model, regardless of whether they are from the baseline segmentor or tagger. In fact, most features from the baseline POS-tagger, when used in the joint model, represent segmentation patterns as well. For example, the aforementioned pattern "number word" + "个", which is useful only for the POS "number word" in the baseline tagger, is also an effective indicator of the segmentation of the two words (especially "个") in the joint model.

4.3.2 The decoding algorithm

One of the main challenges for the joint segmentation and POS-tagging system is the decoding algorithm. The speed and accuracy of the decoder is important for the perceptron learning algorithm, but the system faces a very large search space of combined candidates. Given the linear model and feature templates, efficient exact inference is difficult to achieve even with dynamic programming.

Experiments with the standard beam-search decoder described in Chapter 3, which was applied to word segmentation, resulted in low accuracy. This beam search algorithm processes an input sentence incrementally. At each stage, the incoming character is combined with existing partial candidates in all possible ways to generate new partial candidates. An agenda is used to control the search space, keeping only the B best partial candidates ending with the current character. The algorithm is simple and efficient, with a linear time complexity of $O(BTn)$ when applied to the joint segmentation and tagging problem, where n is the size of input sentence, and T is the size of the tag set. It worked well for word segmentation alone (Chapter 3), even with an agenda size as small as 4, and a simple beam search algorithm also works well for POS-tagging (Ratnaparkhi, 1996). However, when applied to the joint model, it resulted in a reduction in segmentation accuracy (compared to the baseline segmentor) even with B as large as 1024.

[1] In order to provide a comparison for the perceptron algorithm we also tried SVM$^{\text{struct}}$ (Tsochantaridis et al., 2004) for parameter estimation, but this training method was prohibitively slow with the freely available software.

Input: raw sentence *sent* – a list of characters
Variables: candidate sentence *item* – a list of
(word, tag) pairs;
maximum word-length record
maxlen for each tag;
the agenda list *agendas*;
the tag dictionary *tagdict*;
start_index for current word;
end_index for current word
Initialization: $agendas[0] = [\text{“”}]$,
$agendas[i] = []$ $(i! = 0)$
Algorithm:
for $end\_index = 1$ to $sent$.length:
  foreach *tag*:
    for $start\_index =$
    $\max(1,\ end\_index - maxlen[tag] + 1)$
    to end_index:
      $word = sent[start\_index..end\_index]$
      if $(word, tag)$ consistent with *tagdict*:
        for $item \in agendas[start\_index - 1]$:
          $item_1 = item$
          $item_1$.append((*word*,*tag*))
          $agendas[end\_index]$.insert($item_1$)
Outputs: $agendas[sent.\text{length}]$.best_item

Figure 4.2: The decoding algorithm for the joint word segmentor and POS-tagger

One possible cause of the poor performance of the standard beam search method is the combined nature of the candidates in the search space. In the baseline POS-tagger, candidates in the beam are tagged sequences ending with the current word, which can be compared directly with each other. However, for the joint problem, candidates in the beam are segmented and tagged sequences up to the current character, where the last word can be a complete word or a partial word. A problem arises in whether to give POS-tags to incomplete words. If partial words are given POS-tags, it is likely that some partial words are "justified" as complete words by the current POS information. On the other hand, if partial words are not given POS-tag features, the correct segmentation for long words can be lost during partial candidate comparison (since many short completed words with POS-tags are likely to be preferred to a long incomplete word with no POS-tag features).[2]

Another possible cause is the exponential growth in the number of possible candidates with

[2]We experimented with both assigning POS features to partial words and omitting them; the latter method performed better but both performed significantly worse than the multiple beam search method described below.

increasing sentence size. The number increases from $O(T^n)$ for the baseline POS-tagger to $O(2^{n-1}T^n)$ for the joint system. As a result, for an incremental decoding algorithm, the number of possible candidates increases exponentially with the current word or character index. In the POS-tagging problem, a new incoming word enlarges the number of possible candidates by a factor of T (the size of the tag set). For the joint problem, however, the enlarging factor becomes $2T$ with each incoming character. The speed of search space expansion is much faster, but the number of candidates is still controlled by a single, fixed-size beam at any stage. If we assume that the beam is not large enough for all the candidates at at each stage, then, from the newly generated candidates, the baseline POS-tagger can keep $1/T$ for the next processing stage, while the joint model can keep only $1/2T$, and has to discard the rest. Therefore, we can see that the chance for the overall best candidate to fall out of the beam is largely increased. Since the search space growth is exponential, increasing the fixed beam size is not effective in solving the problem.

To solve the above problems, we develop a multiple beam search algorithm, which compares candidates only with complete tagged words, and enables the size of the search space to scale with the input size. The algorithm is shown in Figure 4.2. In this decoder, an agenda is assigned to each character in the input sentence, recording the B best segmented and tagged partial candidates ending with the character. The input sentence is still processed incrementally. However, now when a character is processed, existing partial candidates ending with any previous characters are available. Therefore, the decoder enumerates all possible tagged words ending with the current character, and combines each word with the partial candidates ending with its previous character. All input characters are processed in the same way, and the final output is the best candidate in the final agenda. The time complexity of the algorithm is $O(WTBn)$, with W being the maximum word size, T being the total number of POS-tags and n the number of characters in the input. It is also linear in the input size. Moreover, the decoding algorithm gives reasonable accuracy with a small agenda size of $B = 16$.

To further limit the search space, two optimizations are used. First, the maximum word length for each tag is recorded and used by the decoder to prune unlikely candidates. Because the majority of tags only apply to words with length 1 or 2, this method has a strong effect. Development tests showed that it improves the speed significantly, while having a very small negative influence on the

accuracy. Second, like the baseline POS-tagger, the tag dictionary is used for Chinese closed set tags and the tags for frequent words. To words outside the tag dictionary, the decoder still tries to assign every possible tag.

4.3.3 Online learning

Apart from features, the decoder maintains other types of information, including the tag dictionary, the word frequency counts used when building the tag dictionary, the maximum word lengths by tag, and the character categories. The above data can be collected by scanning the corpus before training starts. However, in both the baseline tagger and the joint POS-tagger, they are updated incrementally during the perceptron training process, consistent with online learning.[3]

The online updating of word frequencies, maximum word lengths and character categories is straightforward. For the online updating of the tag dictionary, however, the decision for frequent words must be made dynamically because the word frequencies keep changing. This is done by caching the number of occurrences of the current most frequent word M, and taking all words currently above the threshold $M/5000+5$ as frequent words. 5000 was chosen to control the number of frequent words. The parameter 5 is used to force all tags to be assigned before a word is seen more than 5 times.

4.4 Experiments

The Chinese Treebank 4 was used for the experiments. It was separated into two parts: CTB3 (420K characters in 150K words / 10364 sentences) was used for the final 10-fold cross validation, and the rest (240K characters in 150K words / 4798 sentences) was used as training and test data for development.

The standard F-scores were used to measure both the word segmentation accuracy and the overall segmentation and POS-tagging accuracy, where the overall accuracy is $TF = 2pr/(p+r)$, with the precision p being the percentage of correctly segmented and tagged words in the decoder output, and the recall r being the percentage of gold-standard tagged words that are correctly identified by

[3]We took this approach because we wanted the whole training process to be online. However, for comparison purposes, we also tried precomputing the above information before training and the difference in performance was negligible.

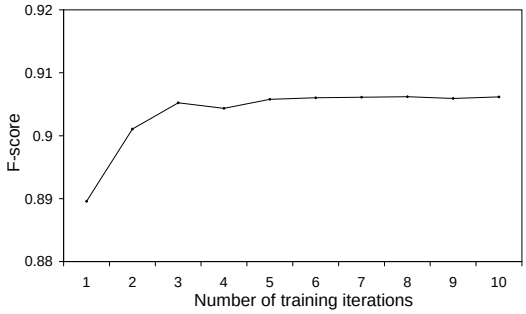


Figure 4.3: The convergence of the learning algorithm for the baseline segmentor

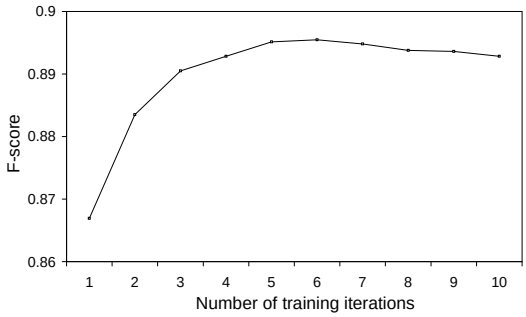


Figure 4.4: The convergence of the learning algorithm for the baseline tagger

the decoder. For direct comparison with Ng and Low (2004), the POS-tagging accuracy was also calculated by the percentage of correct tags on each character.

4.4.1 Development experiments

The accuracy curves for the baseline and joint models according to different numbers of training iterations are shown in Figure 4.3, Figure 4.4 and Figure 4.5, respectively. These curves were used to indicate the convergence of the perceptron and can be used to decide the number of training iterations for the test. It should be noticed that the accuracies from Figure 4.4 and Figure 4.5 are not comparable because gold-standard segmentation is used as the input for the baseline tagger (Figure 4.4). According to the figures, the number of training iterations for the baseline segmentor, POS-tagger, and the joint system are set to 8, 6, and 7, respectively for the remaining experiments.

There are many factors which can influence the accuracy of the joint model. Here we consider the special character category features and the effect of the tag dictionary. The character category

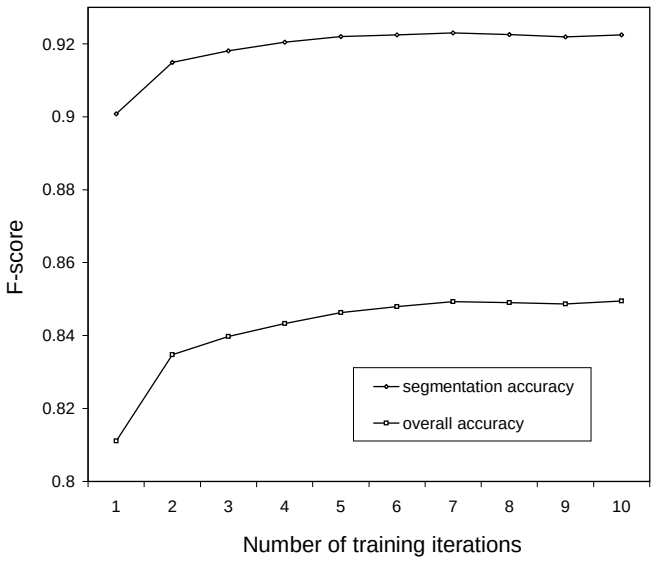


Figure 4.5: The convergence of the learning algorithm for the joint system

Tag	Seg	NN	NR	VV	AD	JJ	CD
NN	20.47	–	0.78	4.80	0.67	2.49	0.04
NR	5.95	3.61	–	0.19	0.04	0.07	0
VV	12.13	6.51	0.11	–	0.93	0.56	0.04
AD	3.24	0.30	0	0.71	–	0.33	0.22
JJ	3.09	0.93	0.15	0.26	0.26	–	0.04
CD	1.08	0.04	0	0	0.07	0	–

Table 4.3: Error analysis for the joint model

features (templates 15 and 16 in Table 4.2) represent a Chinese character by all the tags associated with the character in the training data. They have been shown to improve the accuracy of a Chinese POS-tagger (Tseng et al., 2005). In the joint model, these features also represent segmentation information, since they concern the starting and ending characters of a word. Development tests showed that the overall tagging F-score of the joint model increased from 84.54% to 84.93% using the character category features. In the development test, the use of the tag dictionary improves the decoding speed of the joint model, reducing the decoding time from 416 seconds to 256 seconds. The overall tagging accuracy also increased slightly, consistent with observations from pure POS-taggers in the literature.

The error analysis for the development test is shown in Table 4.3. Here an error is counted

when a POS-tagged word in the standard output is not produced by the decoder, due to incorrect segmentation or tag assignment. Statistics about the six most frequently mistaken tags are shown in the table, where each row presents the analysis of one tag from the standard output, and each column gives a wrongly assigned value. The column "Seg" represents segmentation errors. Each figure in the table shows the percentage of the corresponding error out of all the errors.

It can be seen from the table that NN-VV and VV-NN mistakes were the most commonly made by the decoder, while NR-NN mistakes are also frequent. These three types of errors significantly outnumber the rest, together contributing 14.92% of all the errors. Moreover, the most commonly mistaken tags are NN and VV, while among the most frequent tags in the corpus, PU, DEG and M had comparatively less errors. Lastly, segmentation errors contribute around half (51.47%) of all the errors.

4.4.2 Test results

10-fold cross validation was performed to test the accuracy of the joint word segmentor and POS-tagger, and to make comparisons with existing models in the literature. Following Ng and Low (2004), we partitioned the sentences in CTB3, ordered by sentence ID, into 10 groups evenly. In the nth test, the nth group was used as the testing data.

Table 4.4 shows the results for the cross validation tests, each row representing one test. As can be seen from the table, the joint model outperforms the baseline system in each test.

Table 4.5 shows the overall accuracies of the baseline and joint systems, together with the relevant models in the literature. The accuracy of each model is shown in a row, where "Ng" represents the models from Ng and Low (2004) and "Shi" represents the models from Shi and Wang (2007). Each accuracy measure is shown in a column, including the segmentation F-score (SF), the overall tagging F-score (TF) and the tagging accuracy by characters (TA). As can be seen from the table, our joint model achieved the largest improvement over the baseline, reducing the segmentation error by 14.58% and the overall tagging error by 12.18%.

The overall tagging accuracy of our joint model was comparable to but less than the joint model of Shi and Wang (2007). Despite the higher accuracy improvement from the baseline, the joint system did not give higher overall accuracy. One likely reason is that Shi and Wang (2007) included

	Baseline			Joint		
#	SF	TF	TA	SF	TF	TA
1	96.98	92.91	94.14	97.21	93.46	94.66
2	97.16	93.20	94.34	97.62	93.85	94.79
3	95.02	89.53	91.28	95.94	90.86	92.38
4	95.51	90.84	92.55	95.92	91.60	93.31
5	95.49	90.91	92.57	96.06	91.72	93.25
6	93.50	87.33	89.87	94.56	88.83	91.14
7	94.48	89.44	91.61	95.30	90.51	92.41
8	93.58	88.41	90.93	95.12	90.30	92.32
9	93.92	89.15	91.35	94.79	90.33	92.45
10	96.31	91.58	93.01	96.45	91.96	93.45
Av.	95.20	90.33	92.17	95.90	91.34	93.02

Table 4.4: The accuracies by 10-fold cross validation

SF – segmentation F-score,
TF – overall F-score,
TA – tagging accuracy by character.

Model	SF	TF	TA
Baseline+ (Ng)	95.1	–	91.7
Joint+ (Ng)	95.2	–	91.9
Baseline+* (Shi)	95.85	91.67	–
Joint+* (Shi)	96.05	91.86	–
Baseline (ours)	95.20	90.33	92.17
Joint (ours)	95.90	91.34	93.02

Table 4.5: The comparison of overall accuracies by 10-fold cross validation using CTB

\+ – knowledge about sepcial characters,
* – knowledge from semantic net outside CTB.

knowledge about special characters and semantic knowledge from web corpora (which may explain the higher baseline accuracy), while our system is completely data-driven. However, the comparison is indirect because our partitions of the CTB corpus are different. Shi and Wang (2007) also chunked the sentences before doing 10-fold cross validation, but used an uneven split. We chose to follow Ng and Low (2004) and split the sentences evenly to facilitate further comparison.

Compared with Ng and Low (2004), our baseline model gave slightly better accuracy, consistent with our previous observations about the word segmentors in Chapter 3. Due to the large accuracy gain from the baseline, our joint model performed much better.

In summary, when compared with existing joint word segmentation and POS-tagging systems in the literature, our proposed model achieved the best accuracy boost from the pipelined baseline,

and competitive overall accuracy.

4.5 Comparison with related work

Ng and Low (2004) and Shi and Wang (2007) were important earlier models for joint word segmentation and POS-tagging. Both models reduced the large search space by imposing strong restrictions on the form of search candidates. In particular, Ng and Low (2004) used character-based POS-tagging, which prevents some important POS-tagging features such as word + POS-tag; Shi and Wang (2007) used an N-best reranking approach, which limits the influence of POS-tagging on segmentation to the N-best list. In comparison, our joint model does not impose any hard limitations on the interaction between segmentation and POS information.[4] Fast decoding speed is achieved by using a novel multiple-beam search algorithm.

Nakagawa and Uchimoto (2007) proposed a hybrid model for word segmentation and POS-tagging using an HMM-based approach. Word information is used to process known-words, and character information is used for unknown words in a similar way to Ng and Low (2004). In comparison, our model handles character and word information simultaneously in a single perceptron model.

Different methods have been proposed to reduce error propagation between pipelined tasks, both in general (Sutton et al., 2004; Daume III, 2006; Finkel et al., 2006) and for specific problems such as language modeling and utterance classification (Saraclar and Roark, 2005) and labeling and chunking (Shimizu and Haas, 2006). Though our model is built specifically for Chinese word segmentation and POS-tagging, the idea of using the perceptron model to solve multiple tasks simultaneously can be generalized to other tasks.

4.6 Conclusion and future work

We proposed a joint Chinese word segmentation and POS-tagging model, which achieved a considerable reduction in error rate compared to a baseline two-stage system.

We used a single linear model for combined word segmentation and POS-tagging, and chose the generalized perceptron algorithm for joint training, and beam search for efficient decoding. However,

[4] Apart from the beam search algorithm, we do impose some minor limitations on the search space by methods such as the tag dictionary, but these can be seen as optional pruning methods for optimization.

the application of beam search was complicated by the size of the combined search space. Motivated by the question of what are the comparable partial hypotheses in the space, we developed a novel multiple beam search decoder which effectively explores the large search space. Similar techniques can potentially be applied to other problems involving joint inference in NLP.

Other choices are available for the decoding of a joint linear model, such as exact inference with dynamic programming, provided that the range of features allows efficient processing. The baseline feature templates for Chinese segmentation and POS-tagging, when added together, make exact inference for the proposed joint model prohibitively slow. However, the accuracy loss from the beam decoder, as well as alternative decoding algorithms, are worth further exploration.

The joint system takes features only from the baseline segmentor and the baseline POS-tagger to allow a fair comparison. There may be additional features that are particularly useful to the joint system. Open features, such as knowledge of numbers and European letters, and relationships from semantic networks (Shi and Wang, 2007), have been reported to improve the accuracy of segmentation and POS-tagging. Therefore, given the flexibility of the feature-based linear model, an obvious next step is the study of open features in the joint segmentor and POS-tagger.

Chapter 5

Phrase-Structure Parsing

A parser produces the structure of an input natural language sentence according to a particular formal grammar. Common grammar formalisms used by parsers include context free grammar, dependency grammar, lexical functional grammar, head-driven phrase structure grammar, categorial grammar and tree adjoining grammar. We study parsing with a context free grammar in this chapter, and with dependency grammar in Chapter 6.

Transition-based approaches have shown competitive performance on constituent and dependency parsing of Chinese. State-of-the-art accuracies have been achieved by a deterministic shift-reduce parsing model on parsing the Chinese Treebank 2 data. We propose a global discriminative model based on the shift-reduce constituent parsing process, combined with a beam-search decoder, obtaining competitive accuracies on CTB2. We also report the performance of the parser on CTB5 data, obtaining the highest scores in the literature for a dependency-based evaluation.

5.1 Introduction and background

The syntax of a language can be described in different ways. While traditional grammars are often prescriptive or descriptive, NLP algorithms require *formal grammars* that give a strict definition of syntactic structures. Important grammar formalisms studied in NLP include context free grammar (CFG) (Chomsky, 1957; Collins, 1997), dependency grammar (McDonald et al., 2005a; Nivre et al.,

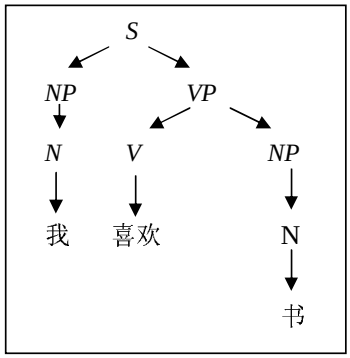


Figure 5.1: A CFG tree

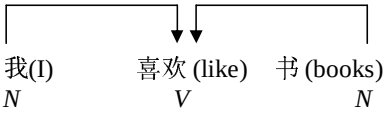


Figure 5.2: A dependency tree

2006), combinatory categorial grammar (CCG) (Steedman, 2000; Clark and Curran, 2007), lexical functional grammar (Kaplan and Bresnan, 1982), link grammar (Sleator and Temperley, 1993) and tree adjoining grammar (Schabes, 1992). Figures 5.1 and 5.2 give an illustration of context free grammar and dependency grammar, respectively. Beause the left hand side of each production rule represents a syntactic constituent in a CFG, CFG parsing is also referred to as constituent parsing.

We study parsing with context free grammar in this chapter, and dependency grammar in the next chapter. We assume that the input for a parser is a segmented and POS-tagged sentence, though a parser can make POS-tagging decisions simultaneously (Bikel, 2004).

5.1.1 Background on CFG parsing

Given an input sentence, a rule-based parser searches for a set of production rules that generates its structure. There are often multiple ways to generate the same sentence, and it is very hard for rule-based parsers to choose the best one. A statistical parser is capable of ambiguity resolution by assigning scores to different possible parses, and choosing the highest scored one to output. According to whether word information is used in the parsing model, statistical parsers can be divided into lexicalized and unlexicalized types. Though the statistical approach has become the

```
procedure Scan(index):
  for each rule R_x → a_index:
    Chart[index, index, R_x] = True

procedure Compose(start, end, middle):
  for each rule R_x → R_y R_z:
    if Chart[start, middle, R_y] == True
       and Chart[middle + 1, end, R_z]:
       Chart[start, end, R_x] = True

function CYK():
  initialize Chart with each entry = False
  for each index ∈ {1..n}:
    Scan(index)
  for each length ∈ {2..n}:
    for each start ∈ {1..n − length + 1}:
       for each middle ∈ {start..start + length − 1}:
          Compose(start, start + length − 1, middle)
  return Chart[1, n, R_start]
```

Figure 5.3: An illustration of the CYK parser

dominant approach in the computational linguistics literature, many ideas and algorithms developed for rule-based parsers are still used.

Rule-based parsing

The CYK (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967) and the Earley (Earley, 1970) algorithms are two important rule-based algorithms. Both use dynamic programming to build charts for parsing.

The CYK (or CKY) algorithm is a bottom-up parser that finds a set of possible parses by building a chart. CYK works with the Chomsky normal form (CNF) of the CFG, which requires each production rule to be either $A \rightarrow B\ C$ or $A \rightarrow \alpha$, where A, B and C are non-terminal symbols and α is a terminal symbol. By restricting the form of production rules, CNF is used to ensure the cubic computational complexity of the CYK algorithm.

Denote the input sentence as $a_1, ..., a_n$, and the non-terminal symbols in a CNF as $R_1, ..., R_r$ (the start symbol being R_{start}). Each CNF production rule can be written either as $R_x \rightarrow a_y$ or as $R_x \rightarrow R_y R_z$. The CYK algorithm is illustrated in Figure 5.3. It finds a possible parse tree by building a compact chart of partial parse trees. The chart is an n by n by r table, where Chart[*start*,

end, *symbol*] records a boolean value indicating whether the subsequence of the input sentence from *start* to *end* can be generated from *symbol*. It is constructed by two bottom-up building actions: "scan", which builds a partial parse tree from a terminal symbol a, and "compose", which builds a partial parse tree from two non-terminals R_y and R_z. Because information about a partial parse tree can be shared by all possible parses that contain it, it is computed only once and cached in the corresponding chart entry for reuse. The chart-parsing idea from the CYK algorithm has wide applications to parsing algorithms.

The Earley algorithm combines bottom-up search with top-down prediction. It processes input sentences from the left to the right, storing different types of partial parses from the beginning of the sentence to the current word in a chart. Three actions are used, including "predict", which lists possible non-terminals for the current word, "scan", which matches the current word to a production rule that generates it, and "complete", which finishes predictions and moves to the next word. Unlike the CYK algorithm, the Earley algorithm does not require the CFG to be in CNF.

Given an input sentence, a rule-based parser finds a parse tree according to the grammar, or NULL when there are no parses. However, there can be often multiple parses for an input sentence, and it is difficult for a rule-based parser to resolve ambiguities. A classical example of ambiguity for English parsing is the prepositional phrase attachment (PP attachment) problem, which is illustrated in Figure 5.4. In this example, it is ambiguous whether "with my hand" is attached to "touch" or "man", for both parse trees are syntactically correct. The ambiguity can only be resolved with knowledge about the relationship between the words in the sentence. It is impossible for manual rules to cover every case exhibiting this kind of ambiguity.

Statistical parsing

An important early model for statistical CFG parsing is the probabilistic context free grammar (PCFG), which associates each production rule with a probability. PCFG is a generative model, treating the production rules in a parse tree as independent of one another. Therefore, the probability of a parse tree can be computed by the product of the probabilities of each production rule in it. Denoting each production rule in a parse tree T for the sentence S with $L_i \rightarrow R_i$, the score of the

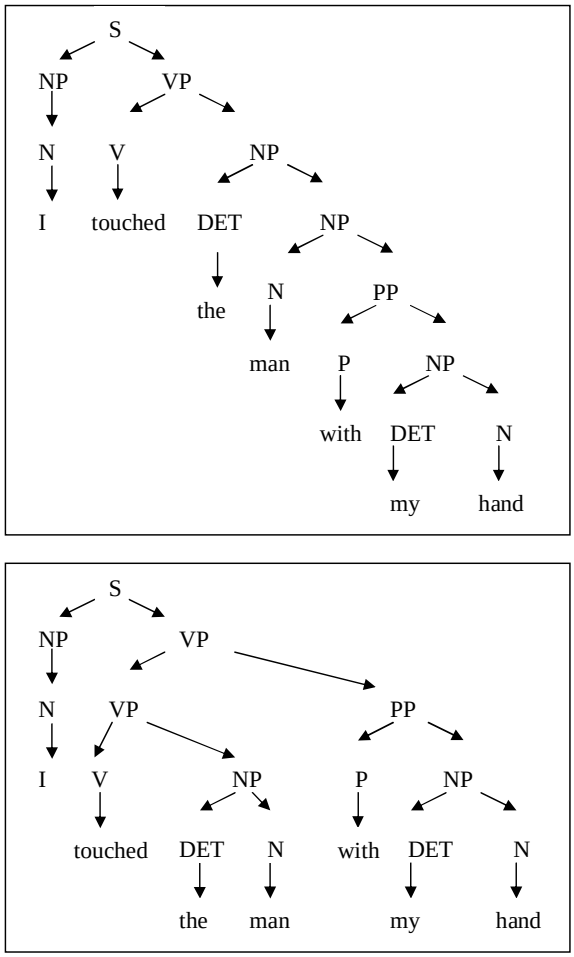


Figure 5.4: An illustration of the PP attachment problem

parse tree T can be written as:

$$Score(T) = P(T|S) = \frac{P(T,S)}{P(S)} \propto P(T,S) = \prod_i P(R_i|L_i)$$

Given the above score definition, a PCFG parser works by examining all possible parse trees for an input sentence, and choosing the one with the highest probability.

A modified version of the CYK algorithm for PCFG parsing is shown in Figure 5.5. In contrast to the CYK algorithm in Figure 5.3, which records in Chart[*start*, *end*, *symbol*] a boolean value indicating whether *symbol* can generate the sub string from *start* to *end*, the CYK algorithm for

```
procedure Scan(index):
  for each rule R_x → a_index:
    Chart[index, index, R_x] = P(a_index|R_x)

procedure Compose(start, end, mid):
  for each rule R_x → R_y R_z:
    Chart[start, end, R_x] = max (
              Chart[start, end, R_x],
              Chart[start, mid, R_y] ×
              Chart[mid+1, end, R_z] × P(R_y R_z|R_x)   )

function CYK():
  initialize Chart with each entry = 0
  for each index ∈ {1..n}:
    Scan(index)
  for each length ∈ {2..n}:
    for each start ∈ {1..n − length + 1}:
      for each mid ∈ {start..start + length − 1}:
          Compose(start, start + length − 1, mid)
  return Chart[1, n, R_start]
```

Figure 5.5: An illustration of the CYK style parser for PCFG

PCFG parsing uses Chart[*start*, *end*, *symbol*] to record the probability score for *symbol* generating the sub string from *start* to *end*. Correspondingly, the function CYK() returns a probability instead of a boolean value. Real valued chart entries allow ambiguity resolution: when there are multiple ways to build a partial parse, the PCFG parser chooses the most probable one in the chart. The chart is built in the bottom-up direction, and the top chart entry corresponds to the most probable parse.

The probabilities associated with production rules are standardly derived from annotated text using maximum likelihood estimation, introduced back in Section 2.1.1. Suppose that there are 1000 production rules starting with symbol VP in the training corpus, among which 600 are $VP \rightarrow V\ NP$. According to the maximum likelihood princple, the probability $P(\,V\ NP\,|\,VP\,)$ is $600/1000 = 0.6$. The manually annotated corpus for parser training is called a *treebank*.

The PCFG parser uses the probability to disambiguate different parse trees for the same sentence. However, its power for ambiguity resolution is limited (Briscoe and Carroll, 1993). One of the reasons is not using word information. For example, a PCFG parser will generate the same parse trees for the sentences “I touched the man with my hand” and “I saw the man with my umbrella”,

because both sentences have the same POS-tag sequence, and are identical to the parser.

Magerman (1995) included word information in a decision-tree based parser by lexicalizing constituents with their head words. The Magerman parser builds a parse tree bottom-up by using a set of actions, and computes the probability of the parse tree by the product of the probabilities of the actions. The production rule, the head words, the POS of head words and position information are extracted from each node and its surrounding nodes, and used as contextual information. The parser uses a two-pass decoding algorithm to balance between time efficiency and search optimality. Magerman's parser was reported to give significant accuracy improvement over PCFG parsing by the use of a lexicalized grammar.

Collins (1996) developed a dependency-based statistical CFG parser, which was much simpler than the Magerman parser, and yet achieved comparable accuracy. Like the Magerman (1995) parser, Collins' parser is based on lexicalized CFG. It treats the probability of a parse tree as the product of the probabilities of a set of baseNPs (a non-recursive noun phrase) and a set of lexical dependencies. The use of baseNPs excludes dependencies within noun phrases from the parse trees, thereby reducing the dependency ambiguity in the model. The parameters are estimated by relative frequencies and the decoder is a CYK style chart parser.

Both Magerman (1995) and Collins (1996) compute the probability of a parse tree by the product of basic components. However, neither gave theoretical justification to the probability computation. Collins (1997) proposed three generative CFG parsing models, which can be seen as instances of the generative models for the structural prediction problem in Section 2.1.1. The computation of parse tree probabilities for these parsers is justified by the probability chain rule and independence assumptions, and the parameter estimation is justified by MLE.

Various probabilistic models have been proposed for statistical parsing since the Collins parser, including the Charniak (2000) parser, which uses another generative model, and the Ratnaparkhi (1999) parser, which is based on the maximum entropy theory. Having been successfully applied to English parsing, probabilistic models were adapted to Chinese. Bikel and Chiang (2000) applied two probabilistic English parsers to Chinese, with minor adjustments to the parameters. Levy and Manning (2003) further studied the linguistic properties of the Chinese Treebank, and proposed a generative probabilistic model for Chinese parsing that makes different independence assumptions

from English. Bikel (2004) reimplemented the Collins parser and applied it to Chinese parsing directly, with only changes to the head-finding rules. While the above work can be seen as adaptations of the Collins parser, Luo (2003) and Fung et al. (2004) applied the maximum entropy parsing method to Chinese parsing.

Discriminative models have been applied to statistical CFG parsing mainly in the form of reranking (Collins, 2000; McClosky et al., 2006; Huang, 2008), which finds the best parse from a list of best outputs of a baseline parser by using discriminative features. The current best accuracies for English Penn Treebank parsing are achieved by discriminative reranking. However, full discriminative parsing models such as Collins and Roark (2004) gave lower accuracies than the best generative models, and competitive accuracies are reached only by incorporating a generative baseline into the discriminative parser.

All the previously mentioned CFG parsers work on parse trees directly, searching for the highest scored parse among possible candidates. An alternative statistical approach is transition-based parsing, which associates scores with each decision in the parsing process, selecting the parse which is built by the highest scoring sequence of decisions (Briscoe and Carroll, 1993). The parsing algorithm is typically some form of bottom-up shift-reduce algorithm, so that scores are associated with actions such as *shift* and *reduce*. One advantage of this approach is that the parsing can be highly efficient, for example by pursuing a greedy strategy in which a single action is chosen at each decision point.

For English constituent-based parsing using the Penn Treebank, the best performing transition-based parser lags behind the current state-of-the-art (Sagae and Lavie, 2005). However, for constituent parsing using the Chinese Treebank, Wang et al. (2006) have shown that a shift-reduce parser can give competitive accuracy scores together with high speeds, by using an SVM to make a single decision at each point in the parsing process.

5.1.2 Introduction to our Chinese constituent parser

There are two deficiencies in Wang et al. (2006)'s approach. One, the greedy nature of the algorithm means that errors can be made by committing to a parse decision too early; and two, the local nature of the classification models means that the parser may make decisions which are not globally

optimal. In this chapter we describe a global discriminative model for Chinese shift-reduce parsing, combined with beam search, thereby addressing the weaknesses of the Wang et al. approach. The parser still operates in linear time, but use of beam search allows the parser to recover from errors made by a greedy search. We use a generalized perceptron model defined over a complete sequence of transition-based actions, so that the perceptron algorithm learns weights in the context of a complete parse.

The contributions of this chapter are as follows. First, we define a global discriminative model for Chinese constituent-based parsing, continuing recent work in this area which has focused on English (Clark and Curran, 2007; Carreras et al., 2008; Finkel et al., 2008). Second, we show how such a model can be applied to shift-reduce parsing and combined with beam search, resulting in an accurate linear-time parser. Using CTB2, our model achieved Parseval F-scores competitive with the current state-of-the-art for Chinese parsing. We also present accuracy scores for the much larger CTB5, using both a constituent-based and dependency-based evaluation. The scores for the dependency-based evaluation were higher than the state-of-the-art dependency parsers for the CTB5 data. Most of the content in the rest of the chapter has been published (Zhang and Clark, 2009).

5.2 The shift-reduce parsing process

The shift-reduce parsing process used by our beam-search decoder is based on the greedy shift-reduce parsers of Sagae and Lavie (2005) and Wang et al. (2006). The process assumes binary-branching trees; section 5.2.1 explains how these are obtained from the arbitrary-branching trees in the Chinese Treebank.

The input is assumed to be segmented and POS-tagged, and the word-POS pairs waiting to be processed are stored in a queue. A stack holds the partial parse trees that are built during the parsing process. A parse *state* is defined as a ⟨stack,queue⟩ pair. Parsing actions, including SHIFT and various kinds of REDUCE, define functions from states to states by shifting word-POS pairs onto the stack and building partial parse trees.

The actions used by the parser are:

- SHIFT, which pushes the next word-POS pair in the queue onto the stack;

- REDUCE–unary–X, which makes a new unary-branching node with label X; the stack is popped and the popped node becomes the child of the new node; the new node is pushed onto the stack;
- REDUCE–binary–{L/R}–X, which makes a new binary-branching node with label X; the stack is popped twice, with the first popped node becoming the right child of the new node and the second popped node becoming the left child; the new node is pushed onto the stack;
- TERMINATE, which pops the root node off the stack and ends parsing.[1][2]

The trees built by the parser are lexicalized. The left (L) and right (R) versions of the REDUCE-binary rules indicate whether the head of the new node is to be taken from the left or right child. Note also that, since the parser is building binary trees, the X label in the REDUCE rules can be one of the temporary constituent labels, such as NP*, which are needed for the binarization process described in Section 5.2.1. Hence the number of left and right binary reduce rules is the number of constituent labels in the binarized grammar.

Wang et al. (2006) give a detailed example showing how a segmented and POS-tagged sentence can be incrementally processed using the shift-reduce actions to produce a binary tree. We show this example in Figure 5.6.

5.2.1 The binarization process

The algorithm in Figure 5.7 is used to map CTB trees into binarized trees, which are required by the shift-reduce parsing process. For any tree node with more than two child nodes, the algorithm works by first finding the head node, and then processing its right-hand-side and left-hand-side, respectively. $Y = X_1..X_m$ represents a tree node Y with child nodes $X_1...X_m (m \geq 1)$.

The label of the newly generated node Y^* is based on the constituent label of the original node Y, but marked with an asterisk. Hence binarization enlarges the set of constituent labels. We call

[1] Sagae and Lavie (2005) and Wang et al. (2006) only used the first three transition actions, setting the final state as all incoming words having been processed, and the stack containing only one node. However, there are a small number of sentences (14 out of 3475 from the training data) that have unary-branching roots. For these sentences, Wang's parser will terminate immediately after a root is found, and therefore fail to produce unary-branching roots. We define a separate action to terminate parsing, allowing unary-reduce actions to be applied to roots.

[2] The effect of the TERMINATE action on the parsing accuracy is empirical. On the one hand, it allows unary-branching roots to be produced, enabling more sentences to be parsed correctly. On the other hand, it enlarges the set of transition actions, and thereby enlarges the search space. In our development experiments, we did not find consistent accuracy improvements by introducing this extra action.

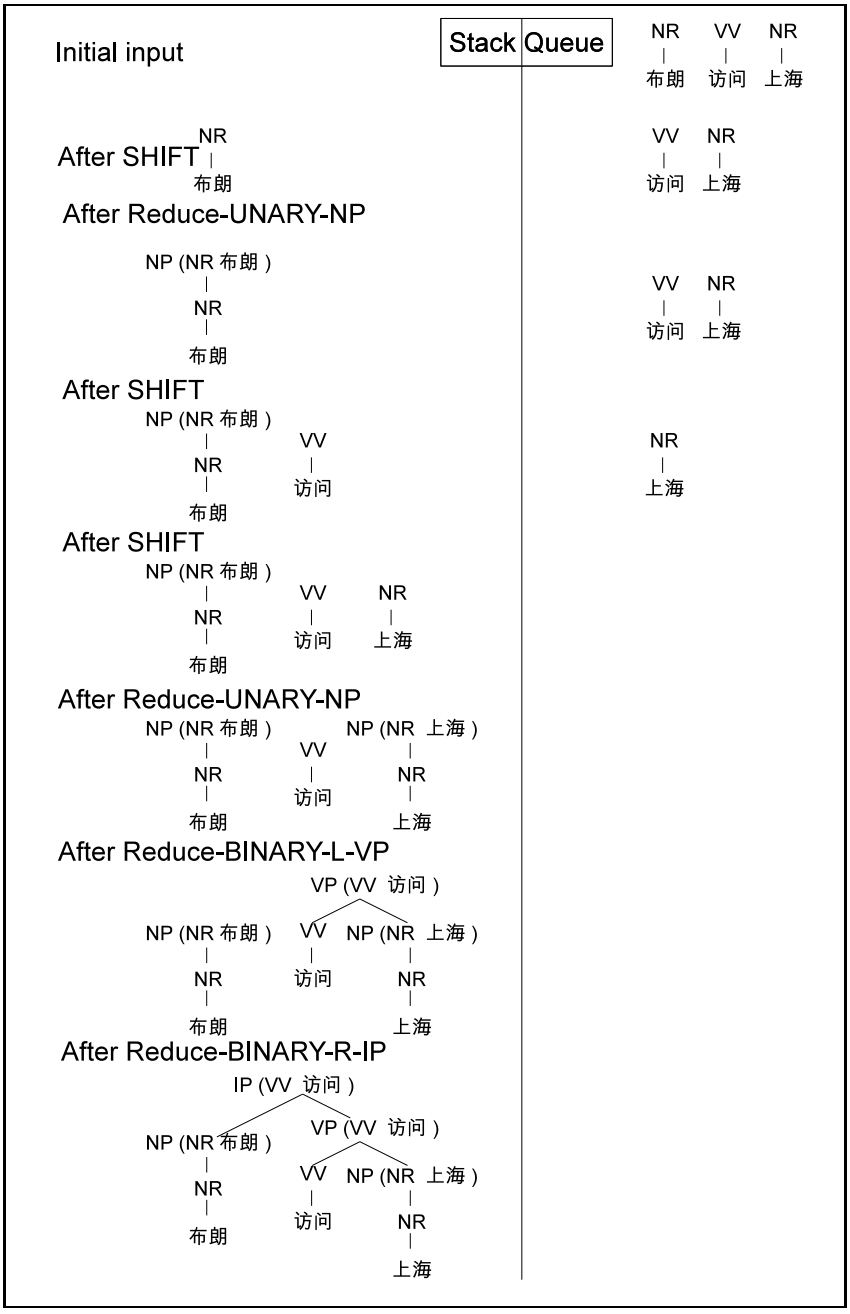


Figure 5.6: An example shift-reduce constituent parsing process

the constituents marked with * *temporary* constituents. The binarization process is reversible, in that output from the shift-reduce parser can be unbinarized into CTB format, which is required for evaluation.

```
for node Y = X_1..X_m ∈ T :
  if m > 2 :
    find the head node X_k(1 ≤ k ≤ m) of Y
    m' = m
    while m' > k and m' > 2 :
      new node Y* = X_1..X_{m'-1}
      Y ← Y* X_{m'}
      m' = m' − 1
    n' = 1
    while n' < k and k − n' > 1 :
      new node Y* = X_{n'}..X_k
      Y ← X_{n'} Y*
      n' = n' + 1
```

Figure 5.7: the binarization algorithm with input T

5.2.2 Restrictions on the sequence of actions

Not all sequences of actions produce valid binarized trees. In the deterministic parser of Wang et al. (2006), the highest scoring action predicted by the classifier may prevent a valid binary tree from being built. In this case, Wang et al. simply return a partial parse consisting of all the subtrees on the stack.

In our parser a set of restrictions is applied which guarantees a valid parse tree. For example, two simple restrictions are that a SHIFT action can only be applied if the queue of incoming words is non-empty, and the binary reduce actions can only be performed if the stack contains at least two nodes. Some of the restrictions are more complex than this; the full set is listed below. The restriction on the number of consecutive unary rule applications is taken from Sagae and Lavie (2005); it prevents infinite running of the parser by repetitive use of unary reduce actions, and ensures linear time complexity in the length of the sentence.

- the shift action can only be performed when the queue of incoming words is not empty;
- when the node on top of the stack is temporary and its head word is from the right child, no shift action can be performed;
- the unary reduce actions can be performed only when the stack is not empty;
- a unary reduce with the same constituent label ($Y \rightarrow Y$) is not allowed;
- no more than three unary reduce actions can be performed consecutively;
- the binary reduce actions can only be performed when the stack contains at least two nodes,

```
Variables: state item item = (S, Q), where
           S is stack and Q is incoming queue;
           the agenda agenda;
           list of state items next;
Algorithm:
  for item ∈ agenda:
    if item.score = agenda.bestScore and
     item.isFinished:
      rval = item
      break
    next = []
    for move ∈ item.legalMoves:
      next.push(item.TakeAction(move))
    agenda = next.getBBest()
Outputs: rval
```

Figure 5.8: the beam-search decoding algorithm for the constituent parser

with at least one of the two nodes on top of stack (with R being the topmost and L being the second) being non-temporary;

- if L is temporary with label X^*, the resulting node must be labeled X or X^* and left-headed (i.e. to take the head word from L); similar restrictions apply when R is temporary;
- when the incoming queue is empty and the stack contains only two nodes, binary reduce can be applied only if the resulting node is non-temporary;
- when the stack contains only two nodes, temporary resulting nodes from binary reduce must be left-headed;
- when the queue is empty and the stack contains more than two nodes, with the third node from the top being temporary, binary reduce can be applied only if the resulting node is non-temporary;
- when the stack contains more than two nodes, with the third node from the top being temporary, temporary resulting nodes from binary reduce must be left-headed;
- the terminate action can be performed when the queue is empty, and the stack size is one.

5.3 Decoding with beam search

Our incremental decoder is based on the shift-reduce parsing process described in Section 5.2. We apply beam-search, keeping the B highest scoring state items in an agenda during the parsing

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$:
    $z_i = parse(x_i, \vec{w})$
    if $z_i \neq y_i$:
      $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

Figure 5.9: the perceptron learning algorithm for the constituent parser

process. The agenda is initialized with a state item containing the starting state, i.e. an empty stack and a queue consisting of all word-POS pairs from the sentence.

At each stage in the decoding process, existing items from the agenda are progressed by applying legal parsing actions. From all newly generated state items, the B highest scoring are put back on the agenda. The decoding process is terminated when the highest scored state item in the agenda reaches the final state. If multiple state items have the same highest score, parsing terminates if any of them are finished. The algorithm is shown in Figure 5.8.

5.4 Model and learning algorithm

We use a linear model to score state items. Recall that a parser state is a ⟨stack,queue⟩ pair, with the stack holding subtrees and the queue holding incoming words waiting to be processed. The score for state item Y is defined by:

$$Score(Y) = \vec{w} \cdot \Phi(Y) = \sum_i \lambda_i f_i(Y)$$

where $\Phi(Y)$ is the global feature vector from Y, and $\vec{w}$ is the weight vector defined by the model. Each element from $\Phi(Y)$ represents the global count of a particular feature from Y. The feature set consists of a large number of features which pick out various configurations from the stack and queue, based on the words and subtrees in the state item. The features are described in Section 5.4.1. The weight values are set using the generalized perceptron algorithm, described in Chapter 2.

The perceptron algorithm is shown in Figure 5.9. As before, it is slightly different from the perceptron algorithm in Figure 2.5 in Chapter 2 in that z_i is computed from the *parse* function, instead

Description	Feature templates
Unigrams	S_0tc, S_0wc, S_1tc, S_1wc,
	S_2tc, S_2wc, S_3tc, S_3wc,
	N_0wt, N_1wt, N_2wt, N_3wt,
	S_0lwc, S_0rwc, S_0uwc,
	S_1lwc, S_1rwc, S_1uwc,
Bigrams	S_0wS_1w, S_0wS_1c, S_0cS_1w, S_0cS_1c,
	S_0wN_0w, S_0wN_0t, S_0cN_0w, S_0cN_0t,
	N_0wN_1w, N_0wN_1t, N_0tN_1w, N_0tN_1t
	S_1wN_0w, S_1wN_0t, S_1cN_0w, S_1cN_0t,
Trigrams	$S_0cS_1cS_2c$, $S_0wS_1cS_2c$,
	$S_0cS_1wS_2c$, $S_0cS_1cS_2w$,
	$S_0cS_1cN_0t$, $S_0wS_1cN_0t$,
	$S_0cS_1wN_0t$, $S_0cS_1cN_0w$
Bracket	S_0wb, S_0cb
	S_0wS_1cb, S_0cS_1wb, S_0cS_1cb
	S_0wN_0tb, S_0cN_0wb, S_0cN_0tb
Separator	S_0wp, S_0wcp, S_0wq, S_0wcq,
	S_1wp, S_1wcp, S_1wq, S_1wcq
	S_0cS_1cp, S_0cS_1cq

Table 5.1: Feature templates concerning actions

of being defined as the highest scored candidate. The parse function in our parser is implemented using beam-search, which is approximate. In order to avoid overfitting we used the averaged version of this algorithm.

We also apply the *early update* modification from Collins and Roark (2004). If the agenda, at any point during the decoding process, does not contain the correct partial parse, it is not possible for the decoder to produce the correct output. In this case, decoding is stopped early and the weight values are updated using the highest scoring partial parse on the agenda.

5.4.1 Feature set

Table 5.1 shows the set of feature templates for the model. Individual features are generated from these templates by first instantiating a template with particular labels, words and tags, and then pairing the instantiated template with a particular action. In the table, the symbols S_0, S_1, S_2, and S_3 represent the top four nodes on the stack, and the symbols N_0, N_1, N_2 and N_3 represent the first four words in the incoming queue. S_0L, S_0R and S_0U represent the left and right child for binary branching S_0, and the single child for unary branching S_0, respectively; w represents the

lexical head token for a node; c represents the label for a node. When the corresponding node is a terminal, c represents its POS-tag, while when the corresponding node is non-terminal, c represents its constituent label; t represents the POS-tag for a word.

The context S_0, S_1, S_2, S_3 and N_0, N_1, N_2, N_3 for the feature templates is taken from Wang et al. (2006). However, Wang et al. (2006) used a polynomial kernel function with an SVM and did not manually create feature combinations. Since we used the linear perceptron algorithm we manually combined Unigram features into Bigram and Trigram features.

The "Bracket" row shows bracket-related features, which were inspired by Wang et al. (2006). Here brackets refer to left brackets including "（", "「" and "《" and right brackets including "）", "」" and "》". In the table, b represents the matching status of the last left bracket (if any) on the stack. It takes three different values: 1 (no matching right bracket has been pushed onto stack), 2 (a matching right bracket has been pushed onto stack) and 3 (a matching right bracket has been pushed onto stack, but then popped off).

The "Separator" row shows features that include one of the separator punctuations (i.e. "，", "。", "、" and "；") between the head words of S_0 and S_1. These templates apply only when the stack contains at least two nodes; p represents a separator punctuation symbol. Each unique separator punctuation between S_0 and S_1 is only counted once when generating the global feature vector. q represents the count of any separator punctuation between S_0 and S_1.

Whenever an action is being considered at each point in the beam-search process, templates from Table 5.1 are matched with the context defined by the parser state and combined with the action to generate features.

Wang et al. (2006) used a range of other features, including rhythmic features of S_0 and S_1 (Sun and Jurafsky, 2003), features from the most recently found node that is to the left or right of S_0 and S_1, the number of words and the number of punctuations in S_0 and S_1, the distance between S_0 and S_1 and so on. We did not include these features in our parser, because they did not lead to improved performance during development experiments.

Constituent	Rules
ADJP	r ADJP JJ AD; r
ADVP	r ADVP AD CS JJ NP PP P VA VV; r
CLP	r CLP M NN NP; r
CP	r CP IP VP; r
DNP	r DEG DNP DEC QP; r
DP	r M; l DP DT OD; l
DVP	r DEV AD VP; r
FRAG	r VV NR NN NT; r
IP	r VP IP NP; r
LCP	r LCP LC; r
LST	r CD NP QP; r
NP	r NP NN IP NR NT; r
NN	r NP NN IP NR NT; r
PP	l P PP; l
PRN	l PU; l
QP	r QP CLP CD; r
UCP	l IP NP VP; l
VCD	l VV VA VE; l
VP	l VE VC VV VNV VPT VRD VSB VCD VP; l
VPT	l VA VV; l
VRD	l VV VA; l
VSB	r VV VE; r
default	r

Table 5.2: Head-finding rules to extract dependency data from CTB
l – search from left to right; r – from right to left.

5.5 Experiments

The experiments were performed using the Chinese Treebank 5 data. Standard data preparation was performed before the experiments: empty terminal nodes were removed; any non-terminal nodes with no children were removed; any unary $X \rightarrow X$ nodes resulting from the previous steps were collapsed into one X node.

The standard Chinese Treebank is not lexicalized, and the head-finding rules from Table 5.2 are used to assign lexical heads to the Treebank data. Most of the head-finding rules are taken from the work of Sun and Jurafsky (2004), while we added rules to handle NN and FRAG, and a default rule to use the rightmost node as the head for constituents that are not listed.

For all experiments, we used the EVALB tool[3] for evaluation, and used labeled recall (LR),

[3]http://nlp.cs.nyu.edu/evalb/

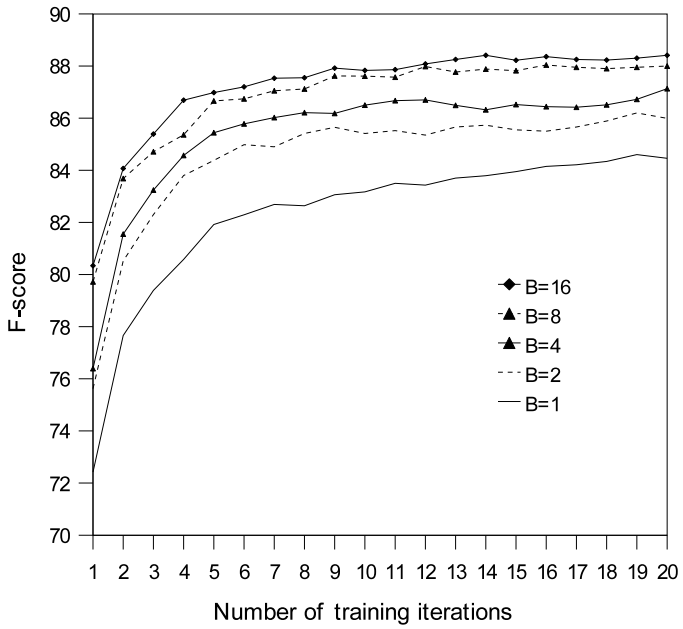


Figure 5.10: The influence of beam-size

labeled precision (LP) and $F1$ score (which is the harmonic mean of LR and LP) to measure parsing accuracy.

5.5.1 The influence of beam-size

Figure 5.10 shows the accuracy curves using different beam-sizes for the decoder. The number of training iterations is on the x-axis with F-score on the y-axis. The tests were performed using the development test data and gold-standard POS-tags. The figure shows the benefit of using a beam as opposed to a beam size of 1 corresponding to deterministic parsing, with comparatively little accuracy gain being obtained beyond a beam size of 8. Hence we set the beam size to 16 for the rest of the experiments.

5.5.2 Test results on CTB2

The experiments in this section were performed using CTB2 to allow comparison with previous work.[4] The data was split into training, development test and test sets, as shown in Table 5.3,

[4]We extracted this data from Chinese Treebank 5.

	Sections	Sentences	Words
Training	001–270	3484	84,873
Development	301–325	353	6,776
Test	271–300	348	7,980

Table 5.3: The standard split of CTB2 data

Model	LR	LP	$F1$
Bikel Thesis	80.9%	84.5%	82.7%
Wang 2006 SVM	87.2%	88.3%	87.8%
Wang 2006 Stacked	88.3%	88.1%	88.2%
Our parser	89.4%	90.1%	89.8%

Table 5.4: Accuracies on CTB2 with gold-standard POS-tags

which is consistent with Wang et al. (2006) and earlier work. The tests were performed using both gold-standard POS-tags and POS-tags automatically assigned by a POS-tagger. We used our own implementation of the perceptron-based tagger of Collins (2002).

The results of various models measured using sentences with less than 40 words and using gold-standard POS-tags are shown in Table 5.4. The rows represent the model of Bikel and Chiang (2000), Bikel (2004), the SVM and ensemble models of Wang et al. (2006), and our parser, respectively. The accuracy of our parser is competitive using this test set.

The results of various models using automatically assigned POS-tags are shown in Table 5.5. The rows in the table represent the models of Bikel and Chiang (2000), Levy and Manning (2003), Xiong et al. (2005), Bikel (2004), Chiang and Bikel (2002), the SVM model from Wang et al. (2006) and the ensemble system from Wang et al. (2006), and our parser, respectively. Our parser gave comparable accuracies to the SVM and ensemble models from Wang et al. (2006). However, comparison with Table 5.4 shows that our parser is more sensitive to POS-tagging errors than some of the other models. One possible reason is that some of the other parsers, e.g. Bikel (2004), use the parser model itself to resolve tagging ambiguities, whereas we rely on a POS tagger to accurately assign a single tag to each word. In fact, for the Chinese data, POS tagging accuracy is not very high, with the perceptron-based tagger achieving an accuracy of only 93%. The beam-search decoding framework we use could accommodate joint parsing and tagging, although the use of features based on the tags of incoming words complicates matters somewhat, since these features rely on tags having been

	≤ 40 words				≤ 100 words				Unlimited			
	LR	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*
Bikel 2000	76.8%	77.8%	77.3%	-	73.3%	74.6%	74.0%	-	-	-	-	-
Levy 2003	79.2%	78.4%	78.8%	-	-	-	-	-	-	-	-	-
Xiong 2005	78.7%	80.1%	79.4%	-	-	-	-	-	-	-	-	-
Bikel Thesis	78.0%	81.2%	79.6%	-	74.4%	78.5%	76.4%	-	-	-	-	-
Chiang 2002	78.8%	81.1%	79.9%	-	75.2%	78.0%	76.6%	-	-	-	-	-
Wang 2006 SVM	78.1%	81.1%	79.6%	92.5%	75.5%	78.5%	77.0%	92.2%	75.0%	78.0%	76.5%	92.1%
Wang 2006 Stacked	79.2%	81.1%	80.1%	92.5%	76.7%	78.4%	77.5%	92.2%	76.2%	78.0%	77.1%	92.1%
Our parser	80.2%	80.5%	80.4%	93.5%	76.5%	77.7%	77.1%	93.1%	76.1%	77.4%	76.7%	93.0%

Table 5.5: Accuracies on CTB2 with automatically assigned tags

	Sections	Sentences	Words
Set A	001–270	3,484	84,873
Set B	Set A; 400–699	6,567	161,893
Set C	Set B; 700–931	9,707	236,051

Table 5.6: Training sets with different sizes

assigned to all words in a pre-processing step. We leave this problem for future work.

In a recent paper, Petrov and Klein (2007) reported *LR* and *LP* of 85.7% and 86.9% for sentences with less than 40 words and 81.9% and 84.8% for all sentences on the CTB2 test set, respectively. These results are significantly better than any model from Table 5.5. However, they used a training set from CTB5 that is much larger than the standard CTB2 training set. Therefore we did not include their scores in the table.

5.5.3 The effect of training data size

CTB2 is a relatively small corpus, and so we investigated the effect of adding more training data from CTB5. Intuitively, more training data leads to higher parsing accuracy. By using an increased number of training sentences (Table 5.6) from CTB5 with the same development test data (Table 5.3), we draw the accuracy curves with different number of training iterations (Figure 5.11). This experiment confirmed that the accuracy increases with the amount of training data.

Another motivation for us to use more training data is to reduce overfitting. We invested considerable effort into feature engineering using CTB2, and found that a small variation of feature templates (e.g. changing the feature template S_0cS_1c from Table 5.1 to S_0tcS_1tc) can lead to a

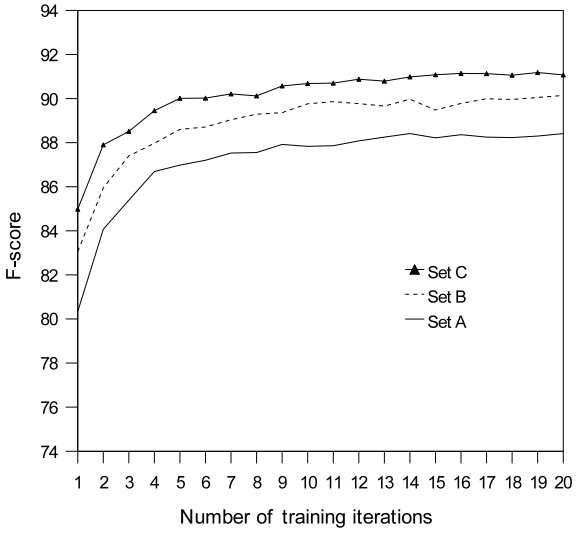


Figure 5.11: The influence of the size of training data

	Sections	Sentences	Words
Training	001–815; 1001–1136	16,118	437,859
Dev	886–931; 1148–1151	804	20,453
Test	816–885; 1137–1147	1,915	50,319

Table 5.7: Standard split of CTB5 data

$\leq$ 40 words				Unlimited			
LR	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*
87.9%	87.5%	87.7%	100%	86.9%	86.7%	86.8%	100%
80.2%	79.1%	79.6%	94.1%	78.6%	78.0%	78.3%	93.9%

Table 5.8: Accuracies on CTB5 using gold-standard and automatically assigned POS-tags

comparatively large change (up to 1%) in the accuracy. One possible reason for this variation is the small size of the CTB2 training data. When performing experiments using the larger set B from Table 5.6, we observed improved stability relative to small feature changes.

5.5.4 Test accuracy using CTB5

Table 5.8 presents the performance of the parser on CTB5. We adopt the data split from Duan et al. (2007), as shown in Table 5.7. We used the same parser configurations as Section 5.5.2.

	Non-root	Root	Complete
Dependency parser	86.21%	76.26%	34.41%
Constituent parser	86.95%	79.19%	36.08%

Table 5.9: Comparison with state-of-the-art dependency parsing using CTB5 data

As an additional evaluation we also produced dependency output from the phrase-structure trees, using the head-finding rules, so that we can also compare with dependency parsers. We compare the dependencies read off our constituent parser using CTB5 data with our dependency parser. The same measures are taken and the accuracies with gold-standard POS-tags are shown in Table 5.9. Our constituent parser gave higher accuracy than the dependency parser described in Chapter 6. It is interesting that our constituent parser uses many fewer feature templates than our dependency parser, but the features include constituent information, which is unavailable to dependency parsers.

5.6 Comparison with related work

The incremental shift-reduce process used by our parser is based on the deterministic shift-reduce parsing process of Wang et al. (2006). An important difference between the Wang et al. (2006) parser and our parser is that our parser is based on a discriminative learning model with global features, whilst the parser of Wang et al. (2006) is based on a local classifier that optimizes each individual choice. Instead of greedy local decoding, we used beam search in the decoder.

Both our parser and and the parser of Collins and Roark (2004) use a global discriminative model and an incremental parsing process. The major difference is the use of different incremental parsing processes. To achieve better performance for Chinese parsing, our parser is based on the shift-reduce parsing process. In addition, we did not include a generative baseline model in the discriminative model, as did Collins and Roark (2004).

5.7 Conclusion

We developed an incremental shift-reduce parser that uses a global discriminative model and a beam-search decoder, which achieved accuracy that is comparable or exceeds that previously reported by others using the CTB2 data. We observed that more training data improves accuracy and

stability, and reported performance of the parser using CTB5 data, obtaining the highest scores for a dependency evaluation on that data in the literature.

Due to the comparatively low accuracy for Chinese POS-tagging, the parsing accuracy dropped significantly using automatically assigned POS-tags than using gold-standard POS-tags. An obvious area for future work is to investigate possible methods of joint POS-tagging and parsing under the discriminative model and beam-search framework.

Chapter 6

Dependency Parsing

Dependency grammar is a syntactic framework which focuses on word-to-word dependencies, which are useful for many NLP applications.

The two main approaches to statistical dependency parsing are graph-based and transition-based, which adopt very different views of the problem, each having its own strengths and limitations. Given an input sentence, graph-based parsers generate all possible candidates, scoring each of them, and choose the highest scored one as the output. Since the search space is typically exponential in the length of the sentence, graph-based parsers often apply dynamic programming to decode in polynomial time. In contrast, transition-based parsers build the output by using a stack and a set of transition-actions, such as *shift* and *reduce*. They are often deterministic and much faster. While giving comparable accuracy to graph-based parsers for English, transition-based parsers have been shown to be more accurate than graph-based parsers with Chinese data.

In this chapter we study both graph-based and transition-based approaches under the framework of beam-search. By developing a graph-based and a transition-based dependency parser, we show that a beam-search decoder is a competitive choice for both methods. More importantly, we propose a beam-search-based parser that combines both graph-based and transition-based parsing into a single system for training and decoding, showing that it outperforms both the pure graph-based and the pure transition-based parsers. The combined system gave state-of-the-art accuracy of 86.2% on the CTB data. When applied to the English Treebank, it also gave an accuracy of 92.1%, which is

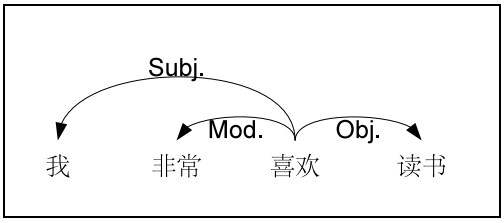


Figure 6.1: An example Chinese dependency tree

close to the best reported for purely supervised models in the literature.

6.1 Introduction and background

A *dependency graph* represents the syntactic structure of a sentence according to a *dependency grammar*. As can be seen from Figure 6.1, a dependency graph consists of a set of vertices and directed arcs. Each arc represents the relationship between a pair of words in the sentence; it points from the head word to its modifier. For example, in the link between the word "我 (I)" and the word "喜欢 (like)", "喜欢 (like)" is the head word and "我 (I)" is the subject that modifies "喜欢 (like)"; in the link between the word "喜欢 (like)" and the word "读书 (reading)", "喜欢 (like)" is the head word and "读书 (reading)" is the object that modifies "喜欢 (like)". In a dependency graph, there is only one word that does not modify any other word, and it is called the *root* of the sentence. The other words each modify exactly one word. No word can modify itself. According to this definition, a dependency graph can also be viewed as a tree. It is also called the *dependency tree* or parse tree.

A dependency tree is called *projective* if there are no crossing links when the sentence is represented linearly, in word order, with the links either all above or all below the word string. Though almost all languages are non-projective to some degree, the majority of sentences in most languages are projective. In the CoNLL shared tasks of dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007), for example, most datasets contain only 1% to 2% non-projective dependency links, and projective dependency parsing models can give reasonable accuracy in these tasks (Carreras et al., 2006). Eisner (1996), McDonald et al. (2005a) and Nivre et al. (2006) discribe important examples of projective parsing models. Non-projective dependency trees can be transformed into projective dependency trees by using a reversible procedure (Nivre et al., 2006). Direct non-projective parsing models have also been proposed (McDonald et al., 2005b).

Left span	Right span	Combined span	Action
left-free	left-free	left-free	N/A
left-free	both-free	left-free	right-left cross link
left-free	both-free	right-free	left-right cross link
left-free	both-free	both-free	N/A
both-free	right-free	left-free	right-left cross link
both-free	right-free	right-free	left-right cross link
both-free	right-free	both-free	N/A
right-free	right-free	right-free	N/A

Table 6.1: Combination of different spans

Input: *sent* – tagged input with a special EOS(root)
Variables: spans *left*, *right* and *combined*
span start index *start*
span type *type*
span length *length*
the chart *chart*
Initialization:
length=2
for *start* = 1 .. *sent*.length:
for *type* in {*left_free*, *both_free*, *right_free*}:
initialize(*chart*[*start*][*length*][*type*])
Algorithm:
for *length* = 3 .. *sent*.length:
for *start* = 0 .. *sent*.length − *length*:
for each possible *left*, *right* sub spans
compute the best *chart*[*start*][*length*][*type*]
according to the span combination rules
Outputs: *chart*[0][*sent*.length][*right_free*]

Figure 6.2: An illustration of the Eisner decoder for unlabeled dependency parsing

A dependency tree without dependency labels such as "Subj" and "Obj" is *unlabeled*. The same techniques used by unlabeled dependency parsers can be applied to labeled dependency parsing.

In an early paper on statistical parsing, Eisner (1996) gave three probabilistic models for dependency parsing. Given an input sentence, the models find the most probable output by examining all possible dependency trees. The probability of a candidate dependency tree is computed as the product of smaller, independent probabilities, according to the three different models. An important contribution of Eisner (1996) is an efficient dynamic programming decoding algorithm. As a projective dependency parser, the Eisner decoder has $O(n^3)$ time complexity, in contrast to the $O(n^5)$

complexity for the lexicalized CYK parsing algorithm. This is achieved by introducing a special structure: the *span*. A span consists of more than one word and the corresponding POS tags and dependency links. Moreover, all the words inside a span (i.e. not the left or right boundary) must have a head word in the span. According to whether the boundary words are dependent on a head within the span, spans can be classified into three types: left-free (only the right boundary word being dependent on a head word), right-free (only the left boundary word being dependent on a head word) and both-free (neither boundary words being dependent on head words). Two spans can be combined into a larger span, according to the rules shown in Table 6.1, provided that (1) they are adjacent and share the same boundary word; (2) the left span is minimal (i.e. not the simple combination of two spans). According to these rules, a dependency tree can be built by recursive span combinations. This process is illustrated by Figure 6.2. By recording the best for all spans with the same type and words, the decoding algorithm performs dynamic programming in cubic time.

MSTParser (McDonald et al., 2005a) used a discriminative model with the Eisner (1996) decoding algorithm, computing the score of a parse tree by the weighted sum of global features. The MIRA and the generalized perceptron learning algorithms introduced back in Chapter 2 were used to train the feature weights.

MaltParser (Nivre et al., 2006) is a deterministic parser that builds the output parse by using a stack and four transition actions. It adopts a very different view of the parsing problem. Instead of searching for the output parse directly, it works by finding a sequence of transition actions to build it. Greedy local search is used to build a parse tree in linear time.

McDonald and Nivre (2007) used the terms "graph-based" and "transition-based" to describe the difference between MSTParser (McDonald and Pereira, 2006) and MaltParser (Nivre et al., 2006). We use the same terms in this thesis, but do not differentiate graph-based and transition-based parsers by whether dynamic programming or greedy search is applied to the decoder. A graph-based parser can use an approximate decoder, and a transition-based parser is not necessarily deterministic. We classify the two types of parser by the following two criteria:

1. whether or not the outputs are built by explicit transition-actions, such as SHIFT and REDUCE;
2. whether it is dependency graphs or transition-actions that the parsing model assigns scores

to.

MSTParser and MaltParser gave comparable accuracies in the CoNLL-X shared task (Buchholz and Marsi, 2006). However, they make different types of errors, which can be seen as a reflection of their theoretical differences (McDonald and Nivre, 2007). MSTParser has the strength of exact inference, but its choice of features is constrained by the requirement of efficient dynamic programming. MaltParser is deterministic, yet its comparatively larger feature range is an advantage. By comparing the two, three interesting research questions arise: (1) how to increase the flexibility in defining features for graph-based parsing; (2) how to add search to transition-based parsing; and (3) how to combine the two parsing approaches so that the strengths of each are utilized.

We study these questions under the beam-search framework, building three parsers. First, using the same features as MSTParser, we develop a graph-based parser to examine the accuracy loss from beam-search compared to exact-search, and the accuracy gain from extra features that are hard to encode for exact inference. Our conclusion is that beam-search is a competitive choice for graph-based parsing. Second, using the transition actions from MaltParser, we build a transition-based parser and show that search has a positive effect on its accuracy compared to deterministic parsing. Finally, we show that by using a beam-search decoder, we are able to combine graph-based and transition-based parsing into a single system, which significantly outperforms each baseline system. In experiments with the CTB, the combined parser gave 86.2% accuracy, outperforming the previous best reported by 1.8%.

In line with previous work on dependency parsing using the Penn Treebank, we focus on projective dependency parsing. The main content in the remainder of this chapter has been published (Zhang and Clark, 2008b).

6.2 The graph-based parser

Like MSTParser (McDonald et al., 2005a; McDonald and Pereira, 2006), we define the graph-based parsing problem as finding the highest scoring tree y from all possible outputs given an input x:

$$y_{out}(x) = \arg\max_{y \in \mathrm{GEN}(x)} \mathrm{Score}(y)$$

```
Variables: agenda – the beam for state items
           item – partial parse tree
           output – a set of output items
           index, prev – word indexes
Input: x – POS-tagged input sentence.
Initialization: agenda = [“”]
Algorithm:
for index in 1..x.length():
    clear output
    for item in agenda:
        // for all prev words that can be linked with
        // the current word at index
        prev = index − 1
        while prev ≠ 0: // while prev is valid
            // add link making prev parent of index
            newitem = item // duplicate item
            newitem.link(prev, index)  // modify
            output.append(newitem)   // record
            // if prev does not have a parent word,
            // add link making index parent of prev
            if item.parent(prev) == 0:
                item.link(index, prev)  // modify
                output.append(item)   // record
            prev = the index of the first word before
                   prev whose parent does not exist
                   or is on its left; 0 if no match
    clear agenda
    put the best items from output to agenda
Output: the best item in agenda
```

Figure 6.3: A beam-search decoder for graph-based dependency parsing, developed from the deterministic Covington (2001) algorithm for projective parsing

where GEN(x) denotes the set of possible parses for the input x. We do not consider the method of finding the arg max to be part of the definition of graph-based parsing, only the fact that the dependency graph itself is being scored, and factored into scores attached to the dependency links.

The score of an output parse y is given by a linear model:

$$\text{Score}(y) = \Phi(y) \cdot \vec{w}$$

where $\Phi(y)$ is the global feature vector from y and $\vec{w}$ is the weight vector of the model.

McDonald et al. (2005a) used both the generalized perceptron and MIRA learning algorithms,

while we chose the perceptron learning algorithm to train the values of $\vec{w}$. Like the previous chapters, the perceptron algorithm for the parser is similar to the algorithm in Figure 2.5 in chapter 2, except that the decoder output is used in place of the exact highest scored output, because we use approximate beam-search. Again, we use the averaged weight values for the perceptron, which reduces overfitting.

The biggest difference between MSTParser and our graph-based parser is the decoding algorithm. While MSTParser used the Eisner (1996) decoder, which performed exact-inference, we use beam-search. We built the beam-search decoder by extending the deterministic Covington algorithm for projective dependency parsing (Covington, 2001). As shown in Figure 6.3, our decoder works incrementally, building a state item (i.e. partial parse tree) word by word. Before decoding starts, the agenda contains an empty sentence. At each processing stage, existing partial candidates from the agenda are extended in all possible ways by adding dependency links between the current words and its predecessors according to the Covington algorithm. After all items in the agenda have been processed, the agenda is emptied and the best B among the newly generated candidates are put into it. The same process is repeated until all input words are processed, and the best candidate output from the agenda is taken as the final output. At each processing stage, partial candidates are scored by the graph-based model according to information up to the current word.

The projectivity of the output dependency trees is guaranteed by the incremental Covington process. Because when each word is processed, possible links are added between the current word and all its predecessors, the time complexity of this algorithm is $O(n^2)$, where n is the length of the input sentence.

During training, the "early update" strategy of Collins and Roark (2004) is used as in Chapter 5: when the correct state item falls out of the beam at any stage, parsing is stopped immediately, and the model is updated using the current best partial item. The intuition is to improve learning by avoiding irrelevant information: when all the items in the current agenda are incorrect, further parsing steps will be irrelevant because the correct partial output no longer exists in the candidate ranking.

Table 6.2 shows the feature templates from the MSTParser (McDonald and Pereira, 2006), which are defined in terms of the context of a word, its parent and its sibling. To give more templates,

1	Parent word (P)	Pw; Pt; Pwt
2	Child word (C)	Cw; Ct; Cwt
3	P and C	PwtCwt; PwtCw; PwCwt; PwtCt; PtCwt; PwCw; PtCt
4	A tag Bt between P, C	PtBtCt
5	Neighbour words of P, C, left (PL/CL) and right (PR/CR)	PtPLtCtCLt; PtPLtCtCRt; PtPRtCtCLt; PtPRtCtCRt; PtPLtCLt; PtPLtCRt; PtPRtCLt; PtPRtCRt; PLtCtCLt; PLtCtCRt; PRtCtCLt; PRtCtCRt; PtCtCLt; PtCtCRt; PtPLtCt; PtPRtCt
6	sibling (S) of C	CwSw; CtSt; CwSt; CtSw; PtCtSt;

Table 6.2: Feature templates from MSTParser
w – word; t – POS-tag.

1	leftmost (CLC) and rightmost (CRC) children of C	PtCtCLCt; PtCtCRCt
2	left (la) and right (ra) arity of P	Ptla; Ptra; Pwtla; Pwtra

Table 6.3: Additional feature templates for the graph-based dependency parser

features from templates 1 – 5 are also conjoined with the link direction and distance, while features from template 6 are also conjoined with the direction and distance between the child and its sibling. Here "distance" refers to the difference between word indices. We apply all these feature templates to the graph-based parser. In addition, we define two extra feature templates (Table 6.3) that capture information about grandchildren and arity (i.e. the number of children to the left or right). These features are not conjoined with information about direction and distance. They are difficult to include in an efficient dynamic programming decoder, but easy to include in a beam-search decoder.

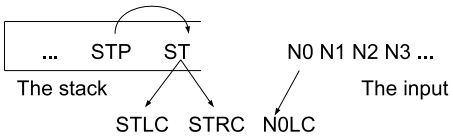


Figure 6.4: Feature context for the transition-based dependency parsing algorithm

6.3 The transition-based parser

We develop our transition-based parser using the transition model of the MaltParser (Nivre et al., 2006), which is characterized by the use of a stack and four transition actions: SHIFT, ARCRIGHT, ARCLEFT and REDUCE. An input sentence is processed from left to right, with an index maintained for the current word. Initially empty, the stack is used throughout the parsing process to store unfinished words, which are the words before the current word that may still be linked with the current or a future word.

The SHIFT action pushes the current word to the stack and moves the current index to the next word. The ARCRIGHT action adds a dependency link from the stack top to the current word (i.e. the stack top becomes the parent of the current word), pushes the current word on to the stack, and moves the current index to the next word. The ARCLEFT action adds a dependency link from the current word to the stack top, and pops the stack. The REDUCE action pops the stack. Among the four transition actions, SHIFT and ARCRIGHT push a word on to the stack while ARCLEFT and REDUCE pop the stack; SHIFT and ARCRIGHT read the next input word while ARCLEFT and ARCRIGHT add a link to the output. By repeated application of these actions, the parser reads through the input and builds a parse tree.

The MaltParser works deterministically. At each step, it makes a single decision and chooses one of the four transition actions according to the current context, including the next input words, the stack and the existing links. As illustrated in Figure 6.4, the contextual information consists of the top of stack (ST), the parent (STP) of ST, the leftmost (STLC) and rightmost child (STRC) of ST, the current word (N0), the next three words from the input (N1, N2, N3) and the leftmost child of N0 (N0LC). Given the context s, the next action T is decided as follows:

$$T(s) = \underset{T \in \text{ACTION}}{\arg\max}\ \text{Score}(T, s)$$

where ACTION = {SHIFT, ARCRIGHT, ARCLEFT, REDUCE}.

One drawback of deterministic parsing is error propagation, since once an incorrect action is made, the output parse will be incorrect regardless of the subsequent actions. To reduce such error propagation, a parser can keep track of multiple candidate outputs and avoid making irrevocable decisions too early. Suppose that the parser builds a set of candidates GEN(x) for the input x, the best output $y_{out}(x)$ can be decided by considering all actions:

$$y_{out}(x) = \arg\max_{y \in \mathrm{GEN}(x)} \sum_{T' \in \mathrm{act}(y)} \mathrm{Score}(T', s_{T'})$$

Here T' represents one action in the sequence (act(y)) by which y is built, and $s_{T'}$ represents the corresponding context when T' is taken.

Our transition-based algorithm keeps B different sequences of actions in the agenda, and chooses the one having the overall best score as the final parse. Pseudo code for the decoding algorithm is shown in Figure 6.5. Here each state item contains a partial parse tree as well as a stack configuration, and state items are built incrementally by transition actions. Initially the stack is empty, and the agenda contains an empty sentence. At each processing stage, one transition action is applied to existing state items as a step to build the final parse. Unlike the MaltParser, which makes a decision at each stage, our transition-based parser applies all possible actions to each existing state item in the agenda to generate new items; then from all the newly generated items, it takes the B with the highest overall score and puts them onto the agenda. In this way, some ambiguity is retained.

Note that the number of transition actions needed to build different parse trees can vary. For example, the three-word sentence "A B C" can be parsed by the sequence of three actions "SHIFT ARCRIGHT ARCRIGHT" (B modifies A; C modifies B) or the sequence of four actions "SHIFT ARCLEFT SHIFT ARCRIGHT" (both A and C modifies B). To ensure that all final state items are built by the same number of transition actions, we require that the final state items must 1) have fully-built parse trees; and 2) have only one root word left on the stack. In this way, stack-popping actions should be made even after a complete parse tree is built, if the stack still contains more than one word.

Now because each word excluding the root must be pushed to the stack once and popped off once during the parsing process, the number of actions needed to parse a sentence is always $2n - 1$,

```
Variables: agenda – the beam for state items
           item – (partial tree, stack config)
           output – a set of output items
           index – iteration index
Input: x – POS-tagged input sentence.
Initialization: agenda = [("", [])]
Algorithm:
for index in 1 .. 2 × x.length() −1:
    clear output
    for item in agenda:
        // when all input words have been read, the
        // parse tree has been built; only pop.
        if item.length() == x.length():
            if item.stacksize() > 1:
                item.Reduce()
                output.append(item)
        // when some input words have not been read
        else:
            if item.lastaction() ≠ Reduce:
                newitem = item
                newitem.Shift()
                output.append(newitem)
            if item.stacksize() > 0:
                newitem = item
                newitem.ArcRight()
                output.append(newitem)
                if (item.parent(item.stacktop())==0):
                    newitem = item
                    newitem.ArcLeft()
                    output.append(newitem)
                else:
                    newitem = item
                    newitem.Reduce()
                    output.append(newitem)
    clear agenda
    transfer the best items from output to agenda
Output: the best item in agenda
```

Figure 6.5: A beam-search decoding algorithm for transition-based dependency parsing

where n is the length of the sentence. Therefore, the decoder has linear time complexity, given a fixed beam size. Because the same transition actions as the MaltParser are used to build each item, the projectivity of the output dependency tree is ensured.

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
// R training iterations; N examples
for $t = 1..R$, $i = 1..N$:
$z_i = \arg\max_{y \in \text{GEN}(x_i)} \sum_{T' \in \text{act}(y_i)} \Phi(T', c') \cdot \vec{w}$
if $z_i \neq y_i$:
$\vec{w} = \vec{w} + \sum_{T' \in \text{act}(y_i)} \Phi(T', c_{T'})$
$- \sum_{T' \in \text{act}(z_i)} \Phi(T', c_{T'})$
Outputs: $\vec{w}$

Figure 6.6: the perceptron learning algorithm for the transition-based dependency parser

1	stack top	STwt; STw; STt
2	current word	N0wt; N0w; N0t
3	next word	N1wt; N1w; N1t
4	ST and N0	STwtN0wt; STwtN0w; STwN0wt; STwtN0t; STtN0wt; STwN0w; STtN0t
5	POS bigram	N0tN1t
6	POS trigrams	N0tN1tN2t; STtN0tN1t; STPtSTtN0t; STtSTLCtN0t; STtSTRCtN0t; STtN0tN0LCt
7	N0 word	N0wN1tN2t; STtN0wN1t; STPtSTtN0w; STtSTLCtN0w; STtSTRCtN0w; STtN0wN0LCt

Table 6.4: Feature templates for the transition-based dependency parser
w – word; t – POS-tag.

We use a linear model to score each transition action, given a context:

$$Score(T, s) = \Phi(T, s) \cdot \vec{w}$$

$\Phi(T, s)$ is the feature vector extracted from the action T and the context s, and $\vec{w}$ is the weight vector. Features are extracted according to the templates shown in Table 6.4, which are based on the context in Figure 6.4. Note that our feature definitions are similar to those used by MaltParser, but rather than using a kernel function with simple features (e.g. STw, N0t, but not STwt or STwN0w), we combine features manually.

As with the graph-based parser, we use the generalized perceptron to train the transition-based model. The perceptron algorithm for the transition-based parser is shown in Figure 6.6. It is worth

noticing that, in contrast to MaltParser, which trains each action decision individually, our training algorithm globally optimizes all action decisions for a parse. Again, "early update" and averaging parameters are applied to the training process.

6.4 The combined parser

The graph-based and transition-based approaches adopt very different views of dependency parsing. McDonald and Nivre (2007) showed that the MSTParser and MaltParser produce different errors. This observation suggests a combined approach: by using both graph-based information and transition-based information, parsing accuracy might be improved.

The beam-search framework we have developed facilitates such a combination. Our graph-based and transition-based parsers share many similarities. Both build a parse tree incrementally, keeping an agenda of comparable state items. Both rank state items by their current scores, and use the averaged perceptron with early update for training. The key differences are the scoring models and incremental parsing processes they use, which must be addressed when combining the parsers.

Firstly, we combine the graph-based and the transition-based score models simply by summation. This is possible because both models are global and linear. In particular, the transition-based model can be written as:

$$\begin{aligned}\text{Score}_{\text{T}}(y) &= \textstyle\sum_{T' \in \text{act}(y)} \text{Score}(T', s_{T'}) \\ &= \textstyle\sum_{T' \in \text{act}(y)} \Phi(T', s_{T'}) \cdot \vec{w_{\text{T}}} \\ &= \vec{w_{\text{T}}} \cdot \textstyle\sum_{T' \in \text{act}(y)} \Phi(T', s_{T'})\end{aligned}$$

If we take $\sum_{T' \in \text{act}(y)} \Phi(T', s_{T'})$ as the global feature vector $\Phi_{\text{T}}(y)$, we have:

$$Score_{\text{T}}(y) = \Phi_{\text{T}}(y) \cdot \vec{w_{\text{T}}}$$

which has the same form as the graph-based model:

$$Score_{\text{G}}(y) = \Phi_{\text{G}}(y) \cdot \vec{w_{\text{G}}}$$

We therefore combine the two models to give:

$$\begin{aligned} Score_{\mathrm{C}}(y) &= Score_{\mathrm{G}}(y) + Score_{\mathrm{T}}(y) \\ &= \Phi_{\mathrm{G}}(y) \cdot \vec{w_{\mathrm{G}}} + \Phi_{\mathrm{T}}(y) \cdot \vec{w_{\mathrm{T}}} \end{aligned}$$

Concatenating the feature vectors $\Phi_{\mathrm{G}}(y)$ and $\Phi_{\mathrm{T}}(y)$ to give a global feature vector $\Phi_{\mathrm{C}}(y)$, and the weight vectors $\vec{w_{\mathrm{G}}}$ and $\vec{w_{\mathrm{T}}}$ to give a weight vector $\vec{w}_{\mathrm{C}}$, the combined model can be written as:

$$Score_{\mathrm{C}}(y) = \Phi_{\mathrm{C}}(y) \cdot \vec{w}_{\mathrm{C}}$$

which is a linear model with exactly the same form as both sub-models, and can be trained with the perceptron algorithm in Figure 2.5. Because the global feature vectors from the sub models are concatenated, the feature set for the combined model is the union of the sub model feature sets.

Second, the transition-based decoder can be used for the combined system. Both the graph-based decoder in Figure 6.3 and the transition-based decoder in Figure 6.5 construct a parse tree incrementally. However, the graph-based decoder works on a per-word basis, adding links without using transition actions, and so is not appropriate for the combined model. The transition-based algorithm, on the other hand, builds state items which contain partial parse trees, and so provides all the information needed by the graph-based parser (i.e. dependency graphs), and hence the combined system. The beam search algorithm does not impose any limitations on the features, allowing features from the graph-based model to be used.

In summary, we built the combined parser by using a global linear model, the union of feature templates and the decoder from the transition-based parser.

6.5 Experiments

Because the parsers that we built are language-independent, and both MSTParser and MaltParser were developed with English data, we used English data for development. The Penn Treebank 3 data was separated into the training, development test and test sets in the same way as McDonald et al. (2005a), as shown in Table 6.5. Bracketed sentences from the Treebank were translated into

	Sections	Sentences	Words
Training	2–21	39,832	950,028
Development	22	1,700	40,117
Test	23	2,416	56,684

Table 6.5: The training, development and test data for English

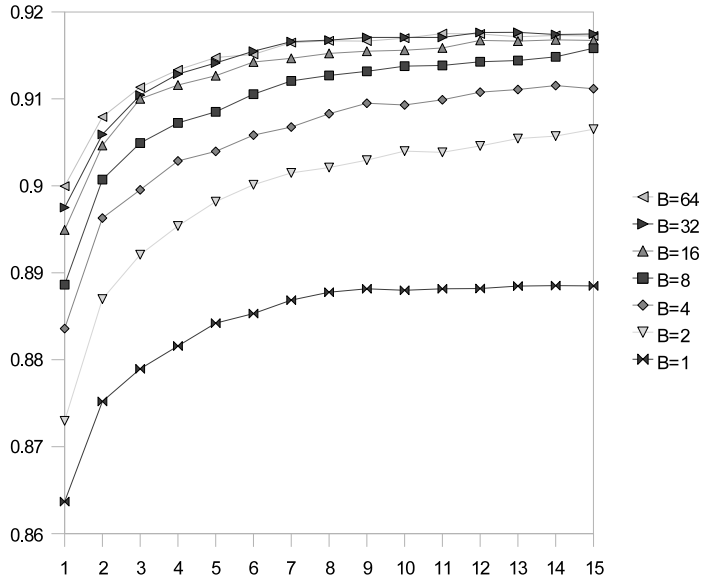


Figure 6.7: The influence of beam size on the transition-based parser, using the development data
X-axis: number of training iterations
Y-axis: word precision

dependency structures using the head-finding rules of Yamada and Matsumoto (2003).

Before parsing, POS-tags are assigned to the input sentence using our reimplementation of the POS-tagger of Collins (2002). Like McDonald et al. (2005a), we evaluated the parsing accuracy by the precision of lexical heads (the percentage of input words, excluding punctuation, that have been assigned the correct parent) and by the percentage of complete matches, in which all words excluding punctuation have been assigned the correct parent.

6.5.1 The influence of the beam-size

The beam size affects all three parsers that we built, and we studied its influence on the transition-based parser. Figure 6.7 shows different accuracy curves using the development data, each with a different beam size B. The X-axis represents the number of training iterations, and the Y-axis the

	Word	Complete
Yamada 2003	90.3	38.4
MSTParser 1	90.7	36.7
Graph [M]	91.2	40.8
Transition	91.4	41.8
Graph [MA]	91.4	42.5
MSTParser 2	91.5	42.1
Combined [TM]	92.0	45.0
Combined [TMA]	92.1	45.4

Table 6.6: Accuracy comparisons on English data

precision of lexical heads.

The parsing accuracy generally increased as the beam size increased, while the quantity of increase became very small when B was large enough. The decoding times after the first training iteration were 10.2s, 27.3s, 45.5s, 79.0s, 145.4s, 261.3s and 469.5s, respectively, when $B = 1, 2, 4, 8, 16, 32, 64$. In the rest of the experiments, we set $B = 64$ in order to obtain the highest possible accuracy.

When $B = 1$, the transition-based parser becomes a deterministic parser. Comparison of the curves when $B = 1$ and $B = 2$ shows that while the use of search reduced the parsing speed, it dramatically improved the quality of the output parses. Therefore, beam-search is a reasonable choice for transition-based parsing.

6.5.2 Comparison between the graph-based, the transition-based and the combined parsers

The accuracies of our graph-based, transition-based and combined parsers on English data are shown together with other systems in Table 6.6. In the table, each row represents a parsing model. Row "Yamada 2003" represents Yamada and Matsumoto (2003); rows "MSTParser 1/2" show the first-order (using feature templates 1 – 5 from Table 6.2) (McDonald et al., 2005a) and second-order (using all feature templates from Table 6.2) (McDonald and Pereira, 2006) MSTParsers, as reported by the corresponding papers. Rows "Graph [M]" and "Graph [MA]" represent our graph-based parser using the features from MSTParser 2 (shown in Table 6.2) and the features from MSTParser 2 plus our additional features (shown in Table 6.3), respectively; row "Transition" represents our transition-based parser; rows "Combined [TM]" and "Combined [TMA]" represent our combined

parser using the transition-based features (shown in Table 6.4) plus the graph-based features from MSTParser 2 (shown in Table 6.2), and the above features plus our additional graph-based features (shown in Table 6.3), respectively. Columns "Word" and "Complete" show the precision of lexical heads and complete matches, respectively.

As can be seen from the table, beam-search reduced the accuracies from 91.5%/42.1% ("MST-Parser 2") to 91.2%/40.8% ("Graph [M]") with the same features as exact-inference. However, with only two extra feature templates from Table 6.3, which were not conjoined with direction or distance information, the accuracies were improved to 91.4%/42.5% ("Graph [MA]"). This improvement is a benefit of beam-search, which facilitates the use of global features.

When only graph-based features (from MSTParser 2) were used, the combined parser gave 88.6% accuracy, which is much lower than 91.2% from the graph-based parser using the same features ("Graph [M]"). This can be explained by the difference between the decoders. In particular, the graph-based model is unable to score the actions REDUCE and SHIFT, since they do not modify the parse tree. This comparison gives a reference for the effect of additional features in the combined parser.

Using both transition-based features and graph-based features from the MSTParser 2 ("Combined [TM]"), the combined parser achieved 92.0% per-word accuracy, which was significantly higher than the pure graph-based and transition-based parsers. Additional graph-based features further improved the accuracy to 92.1%/45.5%, which was the best among all the parsers compared.[1]

6.5.3 Final tests using the Chinese Treebank

We used the Penn Chinese Treebank 5 as experimental data. Following Duan et al. (2007), we split the corpus into training, development and test data as shown in Table 6.7, and use the head-finding rules in Table 5.2 in the last chapter to turn the bracketed sentences into dependency structures.

Like Duan et al. (2007), we used gold-standard POS-tags for the input. The parsing accuracy was evaluated by the percentage of non-root words that were assigned the correct head, the percentage of correctly identified root words, and the percentage of complete matches, all excluding punctuation.

[1]Recently, Koo et al. (2008) reported parent-prediction accuracy of 92.0% using a graph-based parser with a different (larger) set of features (Carreras, 2007). By applying separate word cluster information, Koo et al. (2008) improved the accuracy to 93.2%, which is the best known accuracy on the Penn Treebank data. We excluded these from Table 6.6 because our work is not concerned with the use of such additional knowledge.

	Sections	Sentences	Words
Training	001–815; 1001–1136	16,118	437,859
Dev	886–931; 1148–1151	804	20,453
Test	816–885; 1137–1147	1,915	50,319

Table 6.7: Training, development and test data from CTB

	Non-root	Root	Comp.
Graph [MA]	83.86	71.38	29.82
Duan 2007	84.36	73.70	32.70
Transition	84.69	76.73	32.79
Combined [TM]	86.13	77.04	35.25
Combined [TMA]	86.21	76.26	34.41

Table 6.8: Test accuracies with CTB5 data

The accuracies are shown in Table 6.8. Rows "Graph [MA]", "Transition", "Combined [TM]" and "Combined [TMA]" show our models in the same way as for the English experiments from Section 6.5.2. Row "Duan 2007" represents the transition-based model of Duan et al. (2007), which applied beam-search to the deterministic model of Yamada and Matsumoto (2003), and achieved the previous best accuracy on the data.

The observations were similar to the English tests. Our combined parser outperformed both the graph-based and the transition-based parsers. It gave the best accuracy we are aware of for dependency parsing using the CTB.

6.6 Comparison with related work

Our graph-based parser is derived from the work of McDonald and Pereira (2006). Instead of performing exact inference by dynamic programming, we incorporated the linear model and feature templates from McDonald and Pereira (2006) into our beam-search framework, while adding new global features. Nakagawa and Uchimoto (2007) and Hall (2007) also showed the effectiveness of global features in improving the accuracy of graph-based parsing, using the approximate Gibbs sampling method and a reranking approach, respectively.

Our transition-based parser is derived from the deterministic parser of Nivre et al. (2006). We

incorporated the transition process into our beam-search framework, in order to study the influence of search on this algorithm. Existing efforts to add search to deterministic parsing include Sagae and Lavie (2006a), which applied best-first search to constituent parsing, and Johansson and Nugues (2006) and Duan et al. (2007), which applied beam-search to dependency parsing. All three methods estimate the probability of each transition action, and score a state item by the product of the probabilities of all its corresponding actions. But in contrast to our transition-based parser, which trains all transitions for a parse globally, these models train the probability of each action separately. Based on the work of Johansson and Nugues (2006), Johansson and Nugues (2007) studied global training with an approximated large-margin algorithm. This model is the most similar to our transition-based model, while the differences include the choice of learning and decoding algorithms, the definition of feature templates and our application of the "early update" strategy.

Our combined parser is the main contribution of this chapter. In contrast to the models above, it includes both graph-based and transition-based components. An existing method to combine multiple parsing algorithms is the ensemble approach (Sagae and Lavie, 2006b), which was reported to be useful in improving dependency parsing (Hall et al., 2007). A more recent approach (Nivre and McDonald, 2008) combined MSTParser and MaltParser by using the output of one parser for features in the other. The methods of both Hall et al. (2007) and Nivre and McDonald (2008) can be seen as ways of combining separately defined models. In contrast, our parser combines two components in a single model, in which all parameters are trained consistently.

6.7 Conclusion and future work

We developed a graph-based and a transition-based dependency parser using beam-search, demonstrating that beam-search is a competitive choice for both parsing approaches. We then combined the two parsers into a single system, using discriminative perceptron training and beam-search decoding. The appealing aspect of the combined parser is the incorporation of two largely different views of the parsing problem, thus increasing the information available to a single statistical parser, and thereby significantly increasing the accuracy. When tested using both English and Chinese dependency data, the combined parser was highly competitive compared to the best systems in the literature. The idea of combining different approaches to the same problem using a global discriminative model

could be applied to other NLP tasks.

An interesting conclusion from our experiments is that transition-based parsing gave higher accuracies than graph-based parsing for Chinese. This may be because transition-based parsing is more suitable for Chinese syntax, but this hypothesis remains to be justified in future experiments.

Chapter 7

Conclusion

We explored discriminative approaches to the statistical processing of Chinese syntax, including word segmentation, POS-tagging, and parsing using both phrase-structure and dependency grammars. One of the main advantages of the discriminative approach is the freedom in using arbitrary features to capture global statistical patterns. This freedom was exploited in this thesis by our effort to incorporate more sources of statistical information for the improvement of accuracy. For example, our word-based segmentor extends the character-based approach by including word information; our joint word segmentor and POS-tagger utilizes POS information for word segmentation. We also built a dependency parser that combines statistical information from two different methods into a single, consistently trained model. Our models gave state-of-the-art accuracies in these problems, demonstrating the advantage of using a global discriminative approach.

We built our discriminative models using the perceptron learning and beam-search decoding algorithms. Compared to alternative discriminative training algorithms, the perceptron has a simple parameter update process, which is often efficient in both memory usage and running time, depending on the decoding algorithm. As an online algorithm, perceptron learning is based on the decoding process. Beam-search can have linear time complexity and enables perceptron training to be performed efficiently for complex search spaces. For performing joint word segmentation and POS-tagging, practical running speed can be achieved by beam-search, but not by dynamic programming in our experiments. Though for particular tasks, there may be better choices for attaining a

particular level of accuracy, the combination of perceptron and beam-search was shown to perform consistently well for the tasks we considered. We found that when used with the perceptron learning algorithm, beam-search gave no less accuracy than exact inference for word segmentation, possibly because the parameter adjustment of the perceptron is based on the errors made by the decoder. Beam-search also has the advantage of allowing arbitrary features when compared with dynamic programming. For the combined dependency parser, it is difficult for a dynamic programming decoder to achieve reasonable performance using both graph and transition features.

The thesis made the following contributions to Chinese NLP specifically. For statistical methods applied to Chinese POS-tagging, we advocated joint processing with word segmentation. Because statistical POS-tagging corpora contain segmentation information, no extra effort is needed for manual annotation. A joint system can give a significant accuracy boost over the traditional pipelined approach. For statistical Chinese parsing, we found that the transition-based approach is a competitive choice with both constituent and dependency grammar. A possible explanation is that transition-based parsing is more effective for the syntactic structure in Chinese, and further investigation may lead to improved parsing accuracy.

We conclude that the discriminative approach is a competitive choice for the statistical analysis of Chinese syntax, and that the incorporation of a wider range of relevant information into the features of a discriminative model helps to improve the prediction accuracy in general. The statistical NLP systems that we studied are purely data-driven: the models make predictions according to statistical information from the training data. Data-driven models shift the manual work of building linguistic knowledge into the work of data annotation, and are currently the dominant approach in the computational linguistics literature. However, total reliance on the training data brings disadvantages, one problem being the comparative difficulty in domain adaptation. The accuracy of a statistical NLP system drops significantly when the testing data is in a different domain or genre from the training data. We experimented with the use of linguistic rules to simplify the prediction problem for word segmentation, and observed moderate improvement on accuracy. Future work needs to be done in the study of methods to incorporate more linguistic knowledge into a statistical system and help reduce the reliance on manually created training data.

References

Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Daniel M. Bikel and David Chiang. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of SIGHAN Workshop*, pages 1–6, Hong Kong, China.

Daniel M. Bikel. 2004. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models.* Ph.D. thesis, University of Pennsylvania.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Computational Learning Theory*, pages 144–152.

Ted Briscoe and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Sabine Buchholz and Erwin Marsi. 2006. Conll-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL*, pages 149–164, New York City, June.

Xavier Carreras, Mihai Surdeanu, and Lluis Marquez. 2006. Projective dependency parsing with perceptron. In *Proceedings of CoNLL*, New York City, USA, June.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, pages 9–16, Manchester, England, August.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL*, pages 957–961, Prague, Czech Republic, June.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, pages 132–139, Seattle, Washington, USA, April.

David Chiang and Daniel M. Bikel. 2002. Recovering latent information in treebanks. In *Proceedings of COLING*, Taipei, Taiwan, August.

Noam Chomsky. 1957. *Syntactic Structures.* Mouton, The Hague.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

John Cocke and Jacob T. Schwartz. 1970. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, New York, NY.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of ACL*, pages 184–191, Santa Cruz, California, USA, June.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the ACL*, pages 16–23, Madrid, Spain.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing.* Ph.D. thesis, University of Pennsylvania.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182, Stanford, CA, USA, June.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA, July.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the ACM Southeast Conference*, Athens, Georgia, March.

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

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

James Curran and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the EACL*, pages 91–98, Budapest, Hungary.

J. N. Darroch and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.

Hal Daume III. 2006. *Practical Structured Learning for Natural Language Processing.* Ph.D. thesis, USC.

Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393.

Xiangyu Duan, Jun Zhao, and Bo Xu. 2007. Probabilistic models for action-based Chinese dependency parsing. In *Proceedings of ECML/ECPPKDD*, Warsaw, Poland, September.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING*, pages 340–345, Copenhagen, August.

Thomas Emerson. 2005. The second international Chinese word segmentation bakeoff. In *Proceedings of SIGHAN Workshop*, Jeju, Korea, October.

Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines. In *Proceedings of EMNLP*, pages 618–626, Sydney, Australia, July.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL/HLT*, pages 959–967, Columbus, Ohio, June.

Y. Freund and R. Schapire. 1999. Large margin classification using the perceptron algorithm. In *Machine Learning*, pages 277–296.

Pascale Fung, Grace Ngai, Yongsheng Yang, and Benfeng Chen. 2004. A maximum-entropy Chinese parser augmented by transformation-based learning. *ACM Transactions on Asian Language Information Processing*, 3(2):159–168.

Jesús Giménez and Lluís Màrquez. 2003. Fast and accurate part-of-speech tagging: The SVM approach revisited. In *Proceedings of RANLP*, pages 153–163.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL*, pages 933–939.

Keith Hall. 2007. K-best spanning tree parsing. In *Proceedings of ACL*, Prague, Czech Republic, June.

Borong Huang and Xudong Liao. 2002. *Modern Chinese (third edition)*. Higher education press, China.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL/HLT*, pages 586–594, Columbus, Ohio, June.

Wenbin Jiang, Liang Huang, Qun Liu, and Yajuan Lü. 2008. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL/HLT*, pages 897–904, Columbus, Ohio, June.

Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of CoNLL*, pages 206–210, New York City, USA, June.

Richard Johansson and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *Proceedings of the CoNLL/EMNLP*, pages 1134–1138, Prague, Czech Republic, June.

R. M. Kaplan and J. Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, Bedford, MA.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL/HLT*, pages 595–603, Columbus, Ohio, June.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289, Massachusetts, USA, June.

Roger Levy and Christopher D. Manning. 2003. Is it harder to parse Chinese, or the Chinese treebank? In *Proceedings of ACL*, pages 439–446, Sapporo, Japan, July.

Percy Liang. 2005. Semi-supervised learning for natural language. Master's thesis, MIT.

Xiaoqiang Luo. 2003. A maximum entropy Chinese character-based parser. In *Proceedings of EMNLP*, pages 192–199, Morristown, NJ, USA.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL*, Cambridge, Massachusetts, USA, June.

R. Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL*, pages 49–55, Taipei, Taiwan, August.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML*, pages 591–598, Stanford, California, June.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and self-training for parser adaptation. In *Proceedings of ACL*, pages 337–344, Sydney, Australia, July.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP/CoNLL*, pages 122–131.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88, Trento, Italy, April.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, Ann Arbor, Michigan, June.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP*, pages 523–530, Vancouver, British Columbia, Canada, October.

Tetsuji Nakagawa and Kiyotaka Uchimoto. 2007. A hybrid approach to word segmentation and POS tagging. In *Proceedings of ACL Demo and Poster Session*, Prague, Czech Republic, June.

Hwee Tou Ng and Jin Kiat Low. 2004. Chinese part-of-speech tagging: One-at-a-time or all-at-once? word-based or character-based? In *Proceedings of EMNLP*, Barcelona, Spain.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL/HLT*, pages 950–958, Columbus, Ohio, June.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*, pages 221–225, New York City, June.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL*, pages 915–932.

F. Peng, F. Feng, , and A. McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of COLING*, Geneva, Switzerland.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT/NAACL*, pages 404–411, Rochester, New York, April.

J. Platt. 1998. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report 98-14, Microsoft, Redmond, Washington, April.

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

Nathan Ratliff, J. Andrew Bagnell, and Martin Zinkevich. 2007. Subgradient methods for structured predition. In *Proceedings of AISTATS*, San Juan, Puerto Rico, March.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of EMNLP*, pages 133–142, Somerset, New Jersey.

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

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of IWPT*, pages 125–132, Vancouver, British Columbia, October.

Kenji Sagae and Alon Lavie. 2006a. A best-first probabilistic shift-reduce parser. In *Proceedings of COLING/ACL poster session*, pages 691–698, Sydney, Australia, July.

Kenji Sagae and Alon Lavie. 2006b. Parser combination by reparsing. In *Proceedings of HLT/NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June.

Murat Saraclar and Brian Roark. 2005. Joint discriminative language modeling and utterance classification. In *Proceedings of ICASSP*, volume 1, pages 561–564, Philadelphia, USA, March.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of COLING*, pages 426–432, Nantes, France, August.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of NAACL/HLT*, pages 134–141, Morristown, NJ, USA.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*, pages 760–767, Prague, Czech Republic, June.

Yanxin Shi and Mengqiu Wang. 2007. A dual-layer CRF based joint decoding method for cascade segmentation and labelling tasks. In *Proceedings of IJCAI*, Hyderabad, India.

Nobuyuki Shimizu and Andrew Haas. 2006. Exact decoding for jointly labeling and chunking sequences. In *Proceedings of COLING/ACL, Poster Sessions*, pages 763–770, Sydney, Australia, July.

D. D. Sleator and D. Temperley. 1993. Parsing English with a link grammar. In *Proceedings of IWPT*, Tilburg, The Netherlands, May.

Richard Sproat and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *Proceedings of The Second SIGHAN Workshop*, pages 282–289, Sapporo, Japan, July.

R. Sproat, C. Shih, W. Gail, and N. Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. In *Computational Linguistics*, volume 22(3), pages 377–404.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, Mass.

Honglin Sun and Daniel Jurafsky. 2003. The effect of rhythm on structural disambiguation in Chinese. In *Proceedings of SIGHAN Workshop*.

Honglin Sun and Daniel Jurafsky. 2004. Shallow semantic parsing of Chinese. In *Proceedings of NAACL/HLT*, Boston, USA, May.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. 2004. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of ICML*, Banff, Canada, July.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max margin Markov networks. In *Proceedings of ICML*, Vancouver, Canada, December.

W. J. Teahan, Yingying Wen, Rodger J. McNab, and Ian H. Witten. 2000. A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3):375–393.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT/NAACL*, Edmonton, Canada, May.

Huihsin Tseng, Daniel Jurafsky, and Christopher Manning. 2005. Morphological features help POS tagging of unknown words across language varieties. In *Proceedings of SIGHAN Workshop*, Jeju, Korea, October.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of ICML*, Banff, Canada, July.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of HLT/EMNLP*, Vancouver, Canada, October.

Hanna Wallach. 2002. Efficient training of conditional random fields. Master's thesis, University of Edinburgh.

Xinhao Wang, Xiaojun Lin, Dianhai Yu, Hao Tian, and Xihong Wu. 2006. Chinese word segmentation with maximum entropy and n-gram language model. In *Proceedings of SIGHAN Workshop*, pages 138–141, Sydney, Australia, July.

Fei Xia, 2000. *The part-of-speech tagging guidelines for the Chinese Treebank (3.0).*

Deyi Xiong, Shuanglong Li, Qun Liu, Shouxun Lin, and Yueliang Qian. 2005. Parsing the Penn Chinese Treebank with semantic knowledge. In *Proceedings of IJCNLP*, Jeju, Korea, October.

N. Xue. 2003. Chinese word segmentation as character tagging. In *International Journal of Computational Linguistics and Chinese Language Processing*, volume 8(1).

H Yamada and Y Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proceedings of IWPT*, Nancy, France, April.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189–208.

Yue Zhang and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of ACL*, pages 840–847, Prague, Czech Republic, June.

Yue Zhang and Stephen Clark. 2008a. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL/HLT*, pages 888–896, Columbus, Ohio, June.

Yue Zhang and Stephen Clark. 2008b. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, Hawaii, USA, October.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese Treebank using a global discriminative models. In *Proceedings of IWPT*, Paris, France, October.

Ruiqiang Zhang, Genichiro Kikui, and Eiichiro Sumita. 2006. Subword-based tagging by conditional random fields for Chinese word segmentation. In *Proceedings of the HLT/NAACL, Companion*, volume Short Papers, pages 193–196, New York City, USA, June.

Hai Zhao, Chang-Ning Huang, and Mu Li. 2006. An improved Chinese word segmentation system with conditional random field. In *Proceedings of SIGHAN Workshop*, pages 162–165, Sydney, Australia, July.