Learning Probabilistic Lexicalized Grammars for Natural Language Processing

A thesis presented

by

Rebecca Hwa

to

The Division of Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Harvard University

Cambridge, Massachusetts

March 2001

Abstract

A good representation of language is essential to building natural language processing (NLP) applications. In recent years, the growing availability of machine-readable text corpora has popularized the use of corpus-trained probabilistic grammars to represent languages in NLP systems. Although automatically inducing grammars from large corpora is an appealing idea, it faces several challenges. First, the trained grammar must capture the complexity and ambiguities inherent in human languages. Second, to be useful in practical applications, the grammar must be computationally tractable. Third, although there exists an abundance of raw text, the induction of high-quality grammars depends on manually annotated training data, which are scarce; therefore, the learning algorithm must be able to generalize well. Finally, there are inherent trade-offs in attempting to meet all three challenges; a meta-level challenge is to find a good compromise between the competing factors.

To address these challenges, this thesis presents a partially-supervised induction algorithm based on the Expectation-Maximization principle for the Probabilistic Lexicalized Tree Insertion Grammar (PLTIG) formalism. Using the lexical properties of the PLTIG formalism in the learning algorithm, we show that it is possible to automatically induce a grammar for a natural language that adequately resolves ambiguities and manages domain complexity at a tractable computational cost. By augmenting the basic learning algorithm with training techniques such as grammar adaptation and sample-selection, we show that the induction's dependency on annotated training data can be significantly reduced. Our empirical studies indicate that even with a 36% reduction in annotated training data, the learning algorithm can nonetheless induce grammars without degrading their quality.

Acknowledgements

I would like to thank my advisor, Stuart Shieber. He has taught me what it means to think independently, to conduct research studies judiciously, to communicate clearly, and to make a charlotte russe. I am grateful to Barbara Grosz, whose advice and suggestions throughout the years have helped to make my work as strong as possible. Many thanks to Leslie Valiant for serving on my thesis committee and for making invaluable comments on the dissertation; and to Yves Schabes for introducing me to the Probabilistic Lexicalized Tree Insertion Grammar formalism, which formed the basis of this work.

My years at Harvard have been enriched by the people I met. My fellow graduate students are knowledgeable colleagues as well as good friends: Ellie Baker, Michael Bender, Stan Chen, Josh Goodman (for encouraging me to apply to graduate school), Xienfeng Gu (for letting me play in his band), Raquel Hill, Luke Hunsberger (for the much needed morning coffee), Andy Kehler, Anna Litvinova, Karen Lochbaum, Christine Nakatani, Jill Nickerson, Tim Rauenbusch, Wheeler Ruml (for having good taste in all things), Kathy Ryall (for her first-rate pep talks), Pedro Sander, Rocco Servedio (for his impromptu lectures on machine learning), Keith Smith (for the opera tickets), Dave Sullivan, and Catherine Zheng (for dinners at Zoe's). Other members of the Harvard community have helped me along in many ways. I thank Peg Schafer and Aaron Mandel, not only for system support, but also for hosting the fabulous Friday ice-cream breaks; Ronda Scott and Peter Arvidson, who always managed to get me reinbursed despite my various misadventures with filing expense reports; and Susan Wieczore, who keeps me on track with the academic schedule.

Outside of school, I rely on Phil Bangayan, Josephina Fernandez, Mark Hessman, Buffie Kao, Jon Kleinberg, Karen McArthur, Paul and Ann Martin, James and Amanda Mathiesen, Ben Scarlet, and Sheila Tejada to keep me sane with their friendship.

My deepest thanks go to Lillian Lee, who has been influential in pretty much every aspect of my graduate life since the first time we spoke.

Finally, I could not have completed this thesis without the love and support from my family, who have tried very hard to stifle their question: "So, when are you finishing?" every Thanksgiving, and Ric Crabbe, who never wanders far from my mind.

The work described in this thesis is supported in part by Grant IRI-9350192, Grant IRI-9712068, and an NSF Graduate Student Fellowship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we investigate two central problems for learning a computationally tractable model of natural languages: defining the properties that the model should capture to support language processing applications, and specifying the methods of training a model with these properties automatically and efficiently. We argue that one particular model, called Probabilistic Lexicalized Tree-Insertion Grammars (PLTIG), exhibits many attractive properties for representing natural languages. We provide an efficient learning algorithm for inducing PLTIGs from large corpora, and in addition, we present two novel training techniques that minimize the model's reliance on annotated training data.

1.1 Overview

The field of Natural Language Processing (NLP) is concerned with developing algorithms that enable humans to communicate with machines in human languages (commonly referred to as *natural languages* to distinguish these from artificial languages such as mathematical and programming languages). Research in this area has contributed to advances in many practical applications, such as speech recognition and information extraction.

For the machine to communicate in a natural language, it must have a *model* of the language. By model, we mean some internal representation that guides the system to differentiate those strings that are part of the language from those that are not. How does a system explicitly represent the abstract concept of a language? A naive model might be a database of all the sentences that make up the language. Using this model, the system can accurately determine whether a string is a part of the language by checking its database; however, constructing such a database would be infeasible. To represent the language more cogently, one might compile a set of rules that describe the common linguistic structures in the sentences. For example, one rule about English sentences might be:

> A sentence is made up of a subject noun phrase followed by a predicate verb phrase.

The collection of such rules forms what linguists call a *grammar*. A shorthand of the above rule is often notated as:

$$\mathrm{S} \rightarrow \mathrm{NP}\ \mathrm{VP},$$

where S symbolizes *sentence*; NP symbolizes *noun phrase*; and VP symbolizes *verb phrase*. Rules about the noun phrases and verb phrases can be further specified recursively.

While a set of rules about the language is a more concise model than a database of sentences, it is also a less precise model because it is susceptible to under-generate (unable to identify sentences in the language) as well as over-generate (include spurious sentences not in the language). One of the central challenges in constructing a grammar is to determine the set of rules that can express *all and only* the valid sentences in the language. Formulating general rules that are applicable for all syntactic phenomena is difficult because language usage often depends on the lexical properties of the words. For instance, in some sentences, the verbs *hope* and *believe* may both serve as the main verb:

> He hopes that he will win the race.
>
> He believes that he will win the race.

However, there are also sentences in which *hope* can be used, but *believe* cannot:[1]

> He hopes to win the race.
>
> *He believes to win the race.

Although a rule writer could propose specific rules to cover the special cases, modifying a grammar in an ad hoc fashion may result in inconsistency (i.e., conflicting rules). This highlights another major challenge, the need for a systematic method of selecting the appropriate rules for analyzing sentences. Even without conflicting rules, the problem still exists due to the inherent ambiguities in natural languages. For some expressions, analyses using different rules may yield multiple equally-possible interpretations; while for others, one interpretation may be strongly preferred. For example, in the sentence *She wrote a book on the Titanic*, the prepositional phrase may modify either the verb, *wrote*, or the object noun, *book*; however, in the sentence *She wrote a book on her sabbatical*, the interpretation in which the prepositional phrase modifies the verb is preferred. This example also shows that lexical relationships between words can have a strong influence over selecting rules to resolve ambiguities. To reflect these relationship in a grammar would require a large set of lexicon-specific rules, too complex and time-consuming to be manually designed.

One approach to meeting the challenges is to to embed lexical information within each grammar rule and to parameterize the rules with probability values that specify the likeli-

---

[1] This example is taken from Perlmutter and Soames (1979)

hoods of applying the rules. By anchoring the rules with lexical items, the grammar can express lexicon-specific rules and reduce over-generation. By associating probability values to the rules, the grammar can compute the likelihoods of applying different sets of rules to a sentence and use them to resolve ambiguities. Framing the grammar as a probabilistic model also allows it to be automatically induced in a systematic fashion, thus avoiding the problems associated with manual grammar construction. To combine both the lexical and the statistical features in one representation, we work with the *Probabilistic Lexicalized Tree Insertion Grammar* (PLTIG) formalism. In the first half of this thesis, we present a learning algorithm to induce a PLTIG automatically, and we show that the trained grammar performs well on two applications of NLP through empirical studies.

1.2 Applications of Natural Language Processing

This work focuses on inducing grammars to perform two NLP tasks in particular: language modeling and parsing. Language modeling provides error correction on the word level, whereas parsing provides structural analysis on the sentence level. Because the two tasks process languages at different levels, many grammar models tend to perform well on one task at the cost of the other. In contrast, we will demonstrate that PLTIGs are well-suited to both tasks.

1.2.1 Language Modeling

A *language model* describes the likelihood distribution of all the strings in a language. The term *language model* has a technical meaning, referring only to the computation of string probabilities, not to be confused with *a model of a language*, which refers to the actual grammar. We use probabilistic grammars (models of languages) to compute the probabilities of the strings (the language model). One application for a language model is speech recognition, the task of transcribing an acoustic utterance into text. Because many words have similar acoustic attributes, the system needs to choose between multiple candidates to transcribe an utterance. For example, consider the following pair of candidates:

> They slept in the cheap hostel beds
>
> ?They slept in the cheap hostile bids.

Intuitively, we would think that the first sentence is much more likely to occur than the second. A good language model would reflect our intuition by assigning a higher probability to the first sentence than the second. Thus, the system can use the language model to help find the correct transcription for the utterance.

1.2.2 Parsing

A *parser* analyzes the structure of a sentence, constructing the syntactic tree of the sentence out of relevant grammar rules. Parsing is an important component in applications such as information extraction and machine translation, which benefit from the knowledge about sentence structures. A probabilistic grammar-based parser computes the likelihood distribution of all the parse trees for a sentence, and typically, the parse tree with the highest probability is chosen to be the intended syntactic structure for the sentence.

1.3 Grammar Induction

To automatically learn a grammar for use in NLP tasks discussed above, the system needs to analyze samples taken from the language (*training examples*) to learn how the language is used. Ideally, we would like the learning process to be *unsupervised*, in which the only source of training data provided to the learning system would be a collection of raw text sampled from the language. However, empirical studies suggest that raw text alone is not sufficient for inducing high quality grammars for natural language processing applications (Pereira and Schabes, 1992). Typically, unsupervised training takes the form of a heuristic search that induces locally optimal grammars through maximum likelihood estimations. Since the search process is a greedy algorithm, convergence tends to fall on local optima that make poor grammar models. To induce better grammar models requires encoding more information in training examples. Previous research work has shown that structural information is helpful. The improvement comes at a cost: pairing raw texts with structural information requires human linguistic knowledge. Thus, the laborious process of hand-crafting grammar rules is replaced by a different laborious process of hand-annotating a large collection of sentences. Moreover, grammars trained from such labeled corpora tend to over-fit to the style of the training corpora and may not accurately represent sentences from arbitrary domains. Therefore, even the success of inducing a grammar from one large corpus does not obviate the need to annotate more corpora.

The second half of this thesis concentrates on minimizing the amount of supervision required in the training process. First, we explore the possibility of adapting a trained grammar (induced from existing corpora) to process sentences from a new domain. Our goal is to minimize the amount of *additional* training annotation in the new corpus. Next, we investigate reducing the size of the labeled training corpus through *sample selection*, a method of detecting potentially informative training examples from raw text. Instead of parsing a large number of randomly chosen sentences, the annotators' effort is reduced to focus on only the smaller subset of sentences that are most helpful for the induction process.

1.4 Summary of Contributions

- We have developed an algorithm that can induce PLTIGs, a probabilistic and lexicalized grammar formalism. The algorithm can operate either unsupervised or partially supervised.
- We have shown empirically that a trained PLTIG is well-suited for two NLP tasks: language modeling and parsing, whereas other representations perform well on one task at the cost of the other.
- We have shown that adapting a trained grammar to a new corpus results in higher quality grammars compared to directly inducing a grammar when the corpus contains sparsely annotated training examples.
- We have shown that augmenting the base grammar induction algorithm with sample selection techniques can significantly reduce the amount of training annotations needed for grammar induction.
- We have also applied sample selection to learning a prepositional phrase attachment model, showing that it significantly reduces the number of training examples for this learning task as well.

1.5 Road Map

In Chapter 2 we define the PLTIG formalism and place it in the context of other formalisms. In Chapter 3, we present the induction algorithm for PLTIGs and report an empirical study that evaluates the induced PLTIGs against induced grammars of other formalisms. The next two chapters focus on training techniques that minimize supervised training. In Chapter 4, we investigate *grammar adaptation*, the process of adapting a grammar trained from one corpus to reflect accurately the grammar of another corpus. In Chapter 5, we show that the number of training examples can be reduced by *sample selection*, a method of filtering out uninformative training examples. Finally, we point to areas of future research and offer our concluding remarks in Chapter 6.

Chapter 2

Probabilistic Lexicalized Tree Insertion Grammars

2.1 Introduction

The choice of a grammar formalism for natural language processing depends on several factors. What degree of expressivity is necessary for modeling natural languages? Is it practical, in terms of computational resources, to process sentences when the language is described by this type of grammars? Can a grammar in this form be automatically learned to describe a natural language? In this chapter, we first review several grammar formalisms that have been applied to natural language tasks in the past. We discuss the advantages and the disadvantages each formalism offers. Finally, we describe the Probabilistic Lexicalized Tree Insertion Grammar (PLTIG) formalism, the grammar representation we use for our work presented in this thesis. A comparison between the properties of the different formalisms suggests that the PLTIG formalism is poised to strike a balanced compromise between the competing factors of modeling natural languages. We will validate this hy-

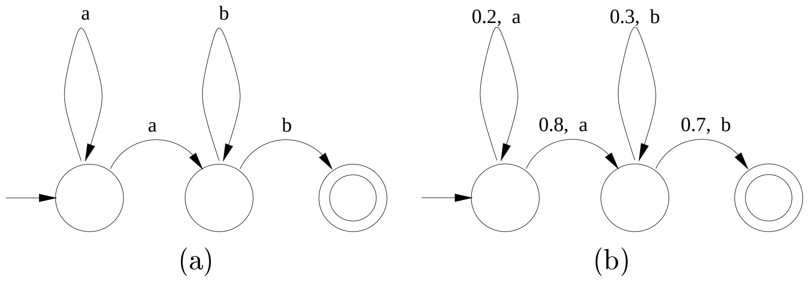


Figure 2.1: (a)An FSA that models the simple language: a^+b^+. The arrow points at the initial state, and the final state is shown as two concentric circles. Each transition arc is labeled with a symbol to be matched with the input token. (b)A probabilistic FSA for the same language. Each transition arc is labeled with an additional probability value.

pothesis with empirical studies in Chapter 3.

2.2 Related Work

Researchers have worked with many different formalisms for NLP tasks, making a trade-off between expressive power, computational efficiency, and linguistic plausibility. In this section, we review several related probabilistic grammar formalisms. They are presented in increasing order of expressive power. A formalism with more expressive power can model more classes of languages; however, it also require more computational time.

2.2.1 Probabilistic Finite State Automata

The weakest type of formalism is the finite state automata (FSA). They describe the most restrictive class of languages, the regular languages. Although regular languages do not adequately characterize natural languages, it has been argued that an FSA is adequate for modeling sentences that *humans* can parse in practice (Pulman, 1986).

An FSA consists of a set of *internal states* and a set of *transition arcs* that connect the states into a network. Each arc is labeled with a symbol that specifies the condition under which a transition could take place. Two states within the set of states are distinguished from the rest: the *initial state* and the *final state*. The FSA keeps track of its current state with a control pointer, originally set to the initial state. As it receives input tokens, the FSA changes states by following the outgoing transition arc from the current state whose symbol matches the current input token. The FSA is said to *accept* the input string if all tokens have been processed and the last current state is the final state. An FSA is a model of a language if it accepts all and only strings that are valid sentences of that language. Figure 2.1(a) shows an example of an FSA that generates the language: a^+b^+.

In a probabilistic FSA, the string matching process is conducted stochastically. Two types of distributions are introduced to describe this process: transitional distributions and symbol distributions. A transitional distribution specifies the likelihood of taking transition arcs out of the current state. Let $p(s_j \mid s_i)$ denote the probability of making a transition from state s_i to state s_j. For a set of states, $s_0, \ldots, s_M$, $\sum_{j=0}^{M} p(s_j \mid s_i) = 1$. A symbol distribution specifies the likelihood of matching the input symbol for a transition arc. Let $q(w_k \mid s_i, s_j)$ denote the probability that the symbol w_k matches the current input token for transition arc between state s_i and state s_j. For a set of symbols, $w_0, \ldots, w_W$, $\sum_{k=0}^{W} q(w_k \mid s_i, s_j) = 1$. Figure 2.1(b) shows the probabilistic counter-part to the previous FSA example.

A probabilistic FSA expresses the acceptance of a string as a probability.[1] It is the sum of the probabilities of all paths through the network that led to the final state for the input string, and the probability of a path is the product of the probabilities of the transition arcs on the path and the probabilities of matching the input tokens. That is, for an input

[1]The Hidden Markov Model (HMM), which has been widely applied to speech recognition, is a probabilistic FSA. Rabiner (1989) provides an excellent overview of HMM's and their roles in speech recognition systems.

string with T tokens, $\mathbf{w} = w_1, \ldots, w_T$:

$$P(\mathbf{w}) = \sum_{\mathbf{s} \in \mathcal{S}} \prod_{i=1}^{T} p(s_i \mid s_{i-1}) q(w_i \mid s_{i-1}, s_i),$$

where $\mathcal{S}$ represents the set of paths through the network and $\mathbf{s}$ specifies one path through the network.

N-grams

One particular type of probabilistic FSA commonly used for language modeling is the N-gram model. An N-gram model assumes that the occurrence of a word in the sentence only depends on the previous $N-1$ words. Typically, N is taken to be 2 (bigrams) or 3 (trigrams). It computes the probability of the sentence by taking the product of the probabilities of each word occurring in the sentence:

$$P(\mathbf{w}) \approx \prod_{i=1}^{T} P(w_i \mid w_{i-N+1}, \ldots, w_{i-1}).$$

An N-gram model is a special degenerative form of a standard probabilistic FSA. Because each string has a unique path through the network, each transition arc matches only one input symbol. To model a language with W symbols, we construct a network of W^{N-1} states, each representing a context of $N-1$ words. There are W out-going transition arcs from every state. Each arc connects a pair of states s_i and s_j such that s_i represents the string $w_{k-N+1}, \ldots, w_{k-1}$ and s_j represents the string $w_{k-N+2}, \ldots, w_{k-1}, w_k$. The symbol probability $q(w_k \mid s_i, s_j)$ is 1 because each transition arc matches only one symbol. The transition probability $p(s_j \mid s_i)$ specifies the likelihood of seeing w_k as the N^{th} word given $w_{i-N+1}, \ldots, w_{i-1}$.

The probabilities in the network can be seen as a set of parameters for the model. To construct an N-gram model, we need to set values to these parameters so that the set of string probabilities computed by the model is an accurate reflection of the true distribution of sentences in the language. Typically these values are determined empirically because we rarely know the true distribution of the language we wish to model. An approximation of the true distribution can be made by taking a large sample of sentences drawn from the language. Using the set of samples as *training* sentences, we choose values for the parameters so that the string probabilities computed by the model reflects the distribution of the training data. Therefore, the estimated transition probability of an arc, denoted $\hat{p}$, is the number of times that this arc is used to leave a state, out of all the times that the state is reached in processing the training data. In other words, it is the co-occurrence frequency of $w_{i-N+1}, \ldots, w_i$ normalized by the co-occurrence frequency of $w_{i-N+1}, \ldots, w_{i-1}$:

$$\hat{p}(w_i \mid w_{i-N+1}, \ldots, w_{i-1}) = \frac{Count(w_{i-N+1}, \ldots, w_i)}{Count(w_{i-N+1}, \ldots, w_{i-1})}.$$

Although the model may seem simplistic, N-grams make surprisingly effective language models. Their strength is that by representing words as states, the network directly captures the dependency relationships between the lexical items in a sentence. Moreover, computationally they are very efficient. Since the path through the network is uniquely determined by the words, we can compute the probability of a string in linear time.

However, an N-gram model has many deficiencies. First, only lexical dependencies between *neighboring* words are modeled, but long-distance dependencies of arbitrary lengths between two words are quite common. For example, in the sentence "I *rowed* the cute little fishing *boat* toward the shore," the verb and its object are separated by 4 words, and the distance can grow arbitrarily wider with additional modifiers. Second, N-grams do not scale well. A network that models a language with W vocabulary items requires W^N parameters; thus, as N increases, the number of parameters grows exponentially. This issue of not having enough data to train all the parameters is known as the *data sparsity problem*. Third, because it is a history-based flat network, an N-gram model cannot provide hierarchical structures (i.e., parse trees).

2.2.2 Probabilistic Context-Free Grammars

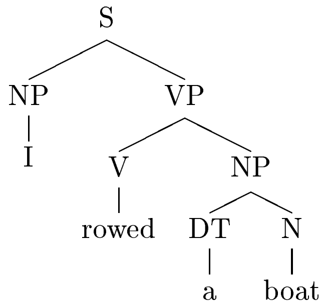
To compensate for the deficiencies of the probabilistic FSA, formalisms with more expressive power have been considered. The Context-Free Grammars formalism (CFG) (Chomsky, 1964) is a proper superset of FSA.

As we have briefly alluded to in the introduction, a CFG consists of a a set of *nonterminal symbols*, usually denoted by upper-case letters; a set of *terminal symbols* (i.e., the vocabularies), usually denoted by lower-case letters; and a set of *production rules*, specifying the allowable expansion of one nonterminal symbol to a string of nonterminal and terminal symbols. A special nonterminal symbol is designated to be the *start symbol*, often denoted as S. It spawns off all subsequent expansions. Figure 2.2(a) illustrates an example of a CFG.

The CFG determines that an input string belongs to the language that it models if there exists a sequence of rule expansions, beginning with the start symbol, that eventually produces a string that exactly matches the input string. A grammar may support multiple expansion sequences for the same string. In this case, the grammar is said to be *ambiguous*. The process of finding the sequences is called *parsing* the string; each sequence is called a *derivation* of the string; and the structure formed by following the steps in a

$$
\begin{array}{lcl}
\mathrm{S} & \rightarrow & \mathrm{NP\ VP} \\
\mathrm{VP} & \rightarrow & \mathrm{VD\ NP} \\
\mathrm{NP} & \rightarrow & \mathrm{DT\ NN} \\
\mathrm{NP} & \rightarrow & I \\
\mathrm{DT} & \rightarrow & a \mid the \\
\mathrm{NN} & \rightarrow & boat \mid car \\
\mathrm{VD} & \rightarrow & drove \mid rowed
\end{array}
$$

(a)



(b)

Figure 2.2: (a)A CFG for a simple language. (b)The derivation and parse tree for the sentence "I rowed a boat" under this grammar.

sequence is a *parse tree* of the sentence. Figure 2.2(b) shows the derivation tree for an example sentence.[2] Finding efficient parsing algorithms for CFGs is a well-studied topic. Kasami (1965),Younger (1967), and Earley (1970) have shown that parsing a sentence requires $O(n^3)$ time for a sentence with n words.

Note that an FSA can be expressed as a CFG. Each state of an FSA network is represented by a nonterminal symbol, and each transition arc is represented by a production rule of the form $S_i \rightarrow w_{ij}\ S_j$, where S_i is the state the arc leaves from, S_j is the state the arc points to, and w_{ij} is the symbol associated with the arc. For this kind of restricted CFG, a nonterminal symbol can only be replaced by at most one nonterminal symbol, so the derivation path is always linear, whereas in a general CFG, a nonterminal symbol can be expanded into multiple nonterminal symbols, resulting in a hierarchical derivation tree.

A probabilistic Context-Free Grammar (PCFG) randomizes the process of expanding the nonterminal symbols. Figure 2.3 shows a probabilistic variant of the CFG in Figure 2.2. A probability is associated with each rule, specifying its likelihood of expanding the left-

[2] In the case of CFG's, a derivation tree also serves as a parse tree for the string. This is not always the case, as we will see in the next section.

(1.0)	S	$\rightarrow$	NP VP
(1.0)	VP	$\rightarrow$	VD NP
(0.8)	NP	$\rightarrow$	DT NN
(0.2)	NP	$\rightarrow$	*I*
(0.6)	DT	$\rightarrow$	*the*
(0.4)	DT	$\rightarrow$	*a*
(0.7)	NN	$\rightarrow$	*car*
(0.3)	NN	$\rightarrow$	*boat*
(0.7)	VD	$\rightarrow$	*drove*
(0.3)	VD	$\rightarrow$	*rowed*

Figure 2.3: The probabilistic counter-part to the previous CFG example.

hand side nonterminal symbol. Let X be a nonterminal symbol, appearing on the left-hand side of r rules; the probabilities of these rules form a distribution:

$$\sum_{i=1}^{r} P(X \rightarrow \alpha_i) = 1.$$

Because nonterminals are expanded independently of other nonterminals, the probability that a derivation v is the true parse for string $\mathbf{w}$ is the product of the probabilities of all the rules used in the expansion. Summing over the set of all possible derivations for the string (denoted as $\mathcal{V}$) results in the probability that the input string belongs to the language:

$$P(\mathbf{w}) = \sum_{v \in \mathcal{V}} P(v).$$

PCFG in Chomsky Normal Form

Building a PCFG for natural language processing is much more difficult than building an N-gram model. Unlike the N-gram model, in which each element of the network directly depends on some lexical words, the form of a PCFG is under specified. How many nonterminal symbols should the grammar have? How many production rules should be allotted to expand each nonterminal symbol, and what form should they be?

One way to place some restrictions on the grammars is to express them in *Chomsky normal form* (CNF), which requires the production rules to follow one of two forms: either a nonterminal symbol is replaced by exactly two nonterminals (i.e., $X \rightarrow YZ$), or a nonterminal symbol is replaced by exactly one terminal symbol (i.e.., $X \rightarrow w$). The main attractive feature of CNF is that any CFG can be transformed into an equivalent CFG in CNF. Therefore, while expressing grammars in CNF restricts the form of the production rule, it does not diminish the expressive power of the grammar. Expressing grammars in CNF also places an upper bound on the number of rules the grammar could possibly have for a given number N of nonterminals and W of terminals. Each nonterminal can expand into at most N^2 nonterminal rules and at most W terminal rules. In total, the grammar can have at most $N^3 + NW$ rules. Thus, the form of a grammar in CNF depends only on the number of nonterminals (which can be determined through trial and error). From this point on, unless specifically noted, we take a PCFG to refer to a probabilistic context free grammar whose production rules are in CNF.

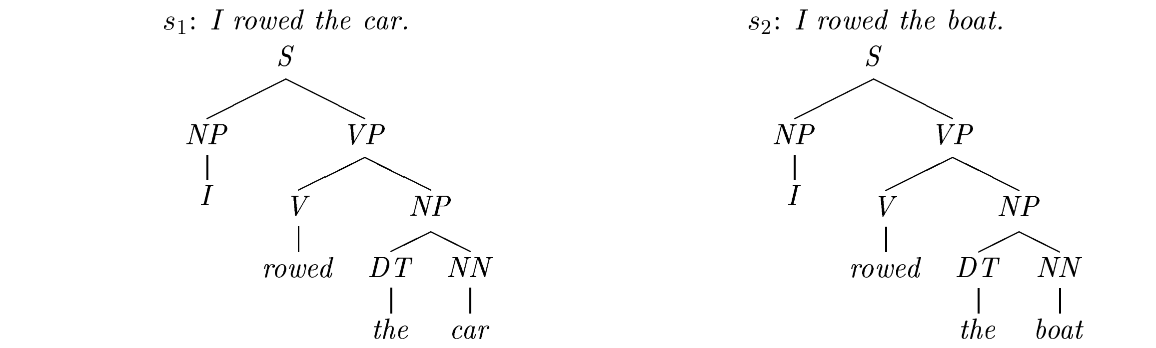
Once the form of the PCFG is specified, we still need to determine the values for the grammar's parameters (i.e., the rule probabilities). Similar to estimating the parameters of an N-gram model, we empirically determine a PCFG's probabilities from a large set of training data sampled from the language. However, the estimation process is not as straightforward. While in an N-gram network the derivation for a sentence is always a unique path, a PCFG may generate many parse trees with different likelihood values. In other words, merely observing a training sentence does not tell us which set of rules should be credited for deriving the sentence. This difficult task of automatically learning the parameters of a grammar is called *grammar induction.* We shall discuss inducing a PCFG in greater detail in Chapter 3.

Assuming that such a PCFG is trainable, there are several advantages to this grammar representation over N-grams. First, PCFGs can describe more complex languages and model linguistic constructs such as center-embedded clauses. For example, in the sentence

> The boat *the sailor rowed* flipped over.

the sentential clause "*the sailor rowed*" is embedded within the sentential clause "the boat flipped over." Second, a PCFG can model hierarchical structures and capture dependencies between linguistic units that are far apart. For example, to relate the main noun phrase to the main verb phrase in the above sentence, a PCFG could use a rule that says "S $\rightarrow$ NP VP." The relationship holds regardless of all further expansions of NP and VP. Third, a PCFG can compute not only the string probability (i.e., the likelihood that a string belongs to the language), but also the *parse tree probabilities*, the likelihood that the string is derived by a parse tree.

Although the PCFG formalism is a more powerful grammar construct and can provide parse trees for sentences, it is, surprisingly, inferior for language modeling and typically performs much worse than the N-gram models in practice. The main flaw of the PCFG representation is that most rules of the grammar do not directly relate lexical items to each other because the nonterminals expand independently of other nonterminals. For example, consider using the grammar in Figure 2.3 to model two strings: *I rowed the car* and *I rowed the boat.* The probabilities assigned to these two strings are 0.02016 and 0.00864, respectively (see Figure 2.4). Although in reality the noun *boat* is the more likely object of the verb rowed, the grammar gives the verb phrase *rowed the car* a higher probability



$P(s_1) = 1x0.2x1x0.3x0.8x0.6x0.7 = 0.02016$ $\quad P(s_2) = 1x0.2x1x0.3x0.8x0.6x0.3 = 0.00864$

Figure 2.4: A PCFG may assign a higher probability to a less likely sentence.

because the noun *car* is deemed more likely to occur overall.

2.2.3 Probabilistic Context-Sensitive Grammars

Are natural languages a proper subset of context-free languages? Some linguistic evidence suggest that the answer is no (Culy, 1985; Shieber, 1985). Thus, there have been some attempts to model natural languages with still more powerful formalisms such as context-sensitive grammars (CSG). As the name suggests, in a CSG, the application of the grammar rules depends on the surrounding *context*. More specifically, whereas a CFG rule has the form:

$$A \rightarrow \beta,$$

where β is a string of nonterminal and terminal symbols, a CSG rule has the form:

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2,$$

where string β is not empty. Therefore, the expansion of nonterminal symbol A depends on the context surrounding it, α_1 and α_2.

Tree Adjoining Grammars

A formalism that can represent some context-sensitive languages[3] is the *tree adjoining grammar* (TAG) formalism, first introduced by Joshi *et al.* (1975). Computational linguists have long been interested in the TAG formalism because it has many attractive linguistic properties that may make it suitable for natural language processing. After presenting the main formalism and several variants, we shall discuss the advantages and disadvantages of using TAGs for NLP. Probabilistic Lexicalized Tree Insertion Grammars (PLTIGs), the grammar representation that forms the foundation of this thesis, is a variant of TAGs. In Section 2.3, we shall discuss how PLTIGs retain most of the advantages of TAGs without suffering the disadvantages.

Unlike the CFGs, sometimes called a *string rewriting system* because an application of the production rule changes the derivation string (replacing a nonterminal with a string of symbols), TAGs are a *tree rewriting system* because its "rules" are expressed as tree fragments and an application of its rules modifies the parse *tree*.[4] Thus, instead of a collection of production rules, a TAG has a collection of *elementary trees*. There are two types of elementary trees: the *initial trees* and the *auxiliary trees*. Figure 2.5 shows some examples of elementary trees. Unlike full parse trees but reminiscent of the productions of a CFG, both types of trees may have nonterminal leaf nodes. Auxiliary trees have, in addition, a distinguished nonterminal leaf node, labeled with the same nonterminal as the root node of the tree, called the *foot* node.[5]

[3]Strictly speaking, TAGs are only "mildly context-sensitive" since they cannot generate some context sensitive languages such as $a^n b^n c^n d^n e^n$, for $n \leq 1$ (Joshi, 1985).

[4]Unlike the derivations of CFGs, the derivation trees of a TAG are *not* the same as the derived (parse) trees. A concrete example illustrates the difference in Section 2.3.3.

[5]The labels of the foot nodes are subscripted with a special symbol, $*$, as shown in Figure 2.5 (b) and (c).

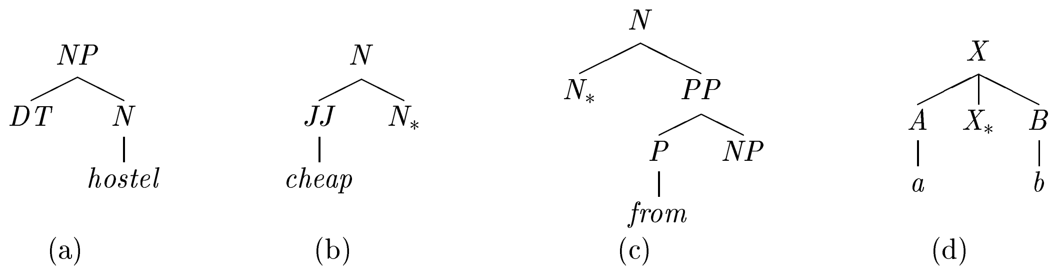


Figure 2.5: Some elementary trees of a TAG. (a) An elementary initial tree. (b)-(d) show some possible auxiliary trees.

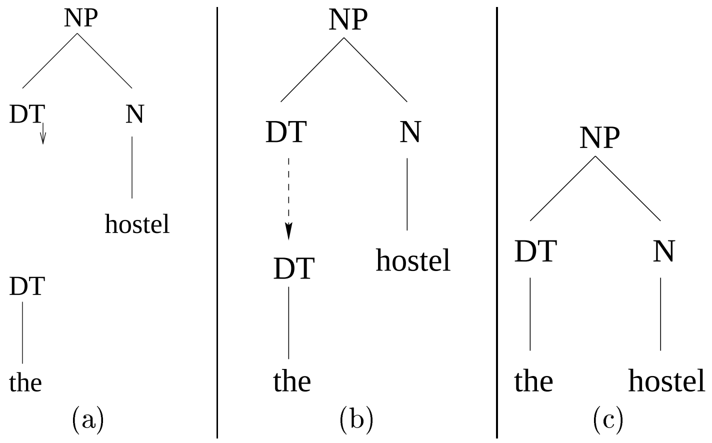


Figure 2.6: The substitution operator substitutes an initial tree into a frontier nonterminal node of another tree. (a) shows the two trees before substitution. (b) shows the initial tree being substituted into the frontier node. (c) shows the resulting derived tree after the substitution has completed.

Two types of operations on the elementary trees are used to construct the parse trees: substitution and adjunction. An initial tree can be *substituted* into the nonterminal leaf node of another tree in a way similar to the substitution of nonterminals in the production rules of CFGs. Figure 2.6 illustrates the steps of the substitution operation. An auxiliary tree is inserted into another tree through *adjunction*, which splices the auxiliary tree into the target tree at an interior node labeled with the same nonterminal as the root (and foot) of the auxiliary tree. Figure 2.7 illustrates the steps of an adjunction operation. Linguistically, substitution operations specify mandatory components, whereas adjunction

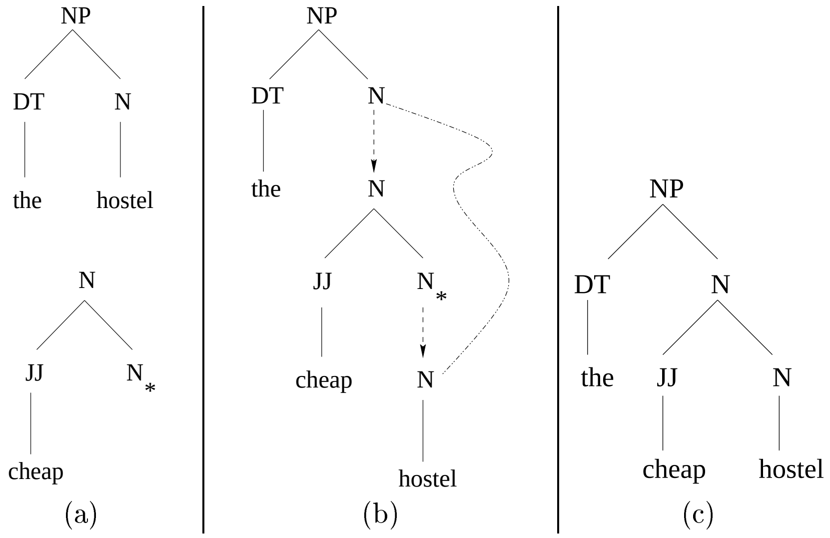


Figure 2.7: The adjunction operator inserts an auxiliary tree into an interior nonterminal node of another tree. (a) shows the trees before adjunction (b) shows the auxiliary tree being inserted into the interior node. (c) shows the resulting derived tree after the adjunction.

operations specify optional arguments. For example, in order for the NP tree in Figure 2.5(a) to be used in the derivation, a DT tree must substitute into its leaf DT node; however, it is optional that the auxiliary tree in Figure 2.5(b) adjoin into the NP tree.

The basic TAG formalism has been extended in many ways to model different aspects of natural language processing. For example, the Synchronous TAG formalism (Shieber and Schabes, 1990), which consists of two TAGs and a set of links that relate the two TAGs, has been applied to machine translation. Another example is the Feature TAG formalism (Vijay-Shanker and Joshi, 1988), in which the TAG tree nodes are augmented with a set of features such that adjunction is allowed only for nodes with compatible features. Feature TAGs may have applications to modeling lexical semantics. Two variants are particularly related to the tasks of parsing and language modeling: Lexicalized TAGs and Probabilistic TAGs.

Variation 1: Lexicalized TAGs

The elementary trees of a *lexicalized TAG* (LTAG) always contains at least one non-empty terminal symbol as a leaf node. The terminal symbol, or a lexical item in the language, is called the *anchor* of the tree (Schabes *et al.*, 1988; Schabes, 1990). LTAGs allow the derivation process to take account both the tree structures and the lexical words. This property should be helpful for discerning ambiguities on both the structural level and the word level.

Variation 2: Probabilistic TAGs

Just as probabilistic models have been proposed for FSA and CFGs, a TAG can also be parameterized by probabilities over the nodes and possible operations to form a probabilistic TAG (Resnik, 1992; Schabes, 1992) (or a probabilistic lexicalized TAG, if the trees are also lexicalized). A probabilistic representation of TAGs makes it possible to compute string probabilities for language modeling and to automatically induce a grammar.

The advantages and disadvantages of TAGs

By using a tree representation, TAGs extend the domain of locality of a grammatical primitive, so that they capture both lexical features and hierarchical structure. Moreover, the adjunction operation elegantly models intuitive linguistic concepts such as long distance dependencies between words. Unlike the N-gram model, which only offers dependencies between neighboring words, these trees can model the interaction of structurally related words that occur far apart.

The major disadvantage of the TAG formalism (and the variants discussed above) is computational inefficiency. Just as the ability to recursively center-embed moves the required parsing time from $O(n)$ for regular grammars to $O(n^3)$ for context-free grammars, so the ability to provide context moves the required parsing time further, to $O(n^6)$ for tree-adjoining grammars.[6] To make the computational time more manageable for parsing, Bangalore (1997) proposed to use local lexical information to generate partial parses for the sentences in a pre-processing stage. However, this technique assumes that a TAG for the natural language of interest already exists. To automatically induce a TAG would be far too computationally expensive for the current technologies. The alternative of hand designing a grammar from scratch, which is the direction pursued by the XTAG project (XTAG-Group, 1999), has the disadvantages of being time-consuming and cannot be easily adapted to new languages.

2.2.4 Lexicalized Context-Free Grammars

A desirable property in the grammar representation for natural language applications is to have lexicalized rules. While the LTAG formalism has this property, its mildly context-sensitive expressive power incurs too high a computational cost to be practical. A natural alternative is to consider lexicalizing the less powerful context-free grammars. Although certain linguistic phenomena are known to be problematic for CFGs (such as arbitrarily large Dutch verb clusters), the recent success of broad-coverage CFG-based parsers (Charniak, 1997),(Chiang, 2000) suggest that CFGs are adequate for modeling most common usages of English.

There are three ways in which one might lexicalize grammars. First, one can modify the tree structures so that all context-free productions contain words. Greibach normal form

[6]The best theoretical upper bound on time complexity for the recognition of Tree Adjoining Languages is $O(M(n^2))$, where $M(k)$ is the time needed to multiply two $k \times k$ boolean matrices.(Rajasekaran and Yooseph, 1995)

(GNF) (Greibach, 1965) provides a well-known example of such a lexicalized context-free formalism. To express a grammar in GNF, each production rule contains a lexical item, and that word is the first symbol on the right hand side. That is, every rule has the form:

$$\mathrm{A} \rightarrow \mathrm{a}\ \alpha,$$

where α is a (possibly empty) string of nonterminal symbols. This method is not practical because altering the structures of the grammar damages the linguistic information stored in the original grammar (Schabes and Waters, 1995). Second, one might propagate lexical information upward through the productions. Examples of formalisms using this approach include the work of Magerman (1995), Charniak (1997), Collins (1997), and Goodman (1997). Finally, a more linguistically motivated approach is to expand the domain of productions downward to incorporate more tree structures. As we have seen earlier, LTAGs, although not context-free, is an instance of this category. The *probabilistic lexicalized tree insertion grammar* (PLTIG) formalism, the main topic of this thesis, belongs to this third category and generates only context-free languages.

2.3 Definition of the PLTIG formalism

The tree insertion grammar (TIG) formalism is a proper subset of TAGs that is context-free equivalent. Similar to TAGs, the TIG formalism also supports lexicalization and probabilistic parameterization. The resulting variant is the probabilistic lexicalized tree insertion grammar (PLTIG) formalism.

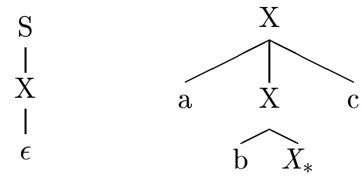
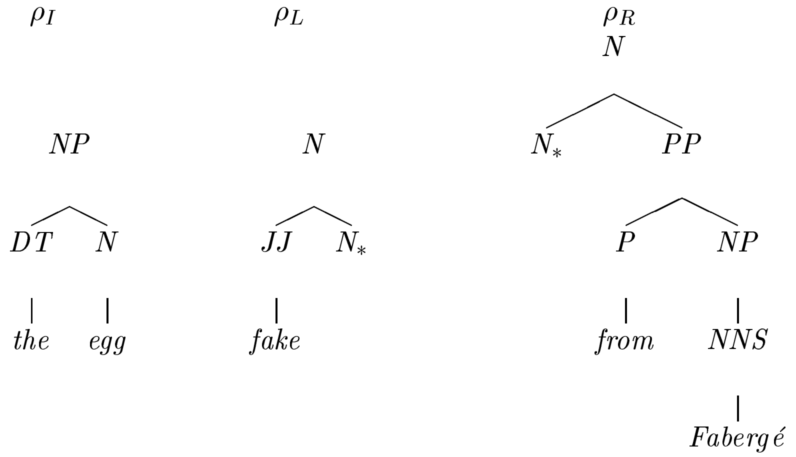


Figure 2.8: A set of TAG elementary trees that can generate the non-context-free language: $a^n b^n c^n$. This is possible through repeatedly adjoining the wrapping auxiliary tree into itself at the interior X node.

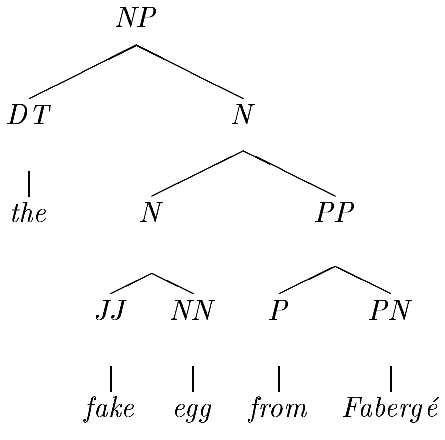
2.3.1 Differences between TIGs and TAGs

TIGs have less expressive power than TAGs. TAGs can generate some strictly context-sensitive languages. They do so by using *wrapping* auxiliary trees, which allow non-empty frontier nodes (i.e., leaf nodes whose labels are not the empty terminal symbol) on both sides of the foot node. A wrapping auxiliary tree makes the formalism context-sensitive because it coordinates the string to the left of its foot with the string to the right of its foot while allowing a third string to be inserted into the foot (see Figure 2.8 for a simple example). The complexity of TAGs can be moderated by eliminating just the wrapping auxiliary trees. TIGs prevent wrapping by restricting auxiliary tree structures to be in one of two forms: the *left auxiliary tree*, whose non-empty frontier nodes are all to the left of the foot node; or the *right auxiliary tree*, whose non-empty frontier nodes are all to the right of the foot node. Auxiliary trees of different types cannot adjoin into each other if the adjunction would result in a wrapping auxiliary tree. The resulting system is strongly equivalent to CFGs, yet is fully lexicalized and still $O(n^3)$ parsable, as shown by Schabes and Waters (1995).

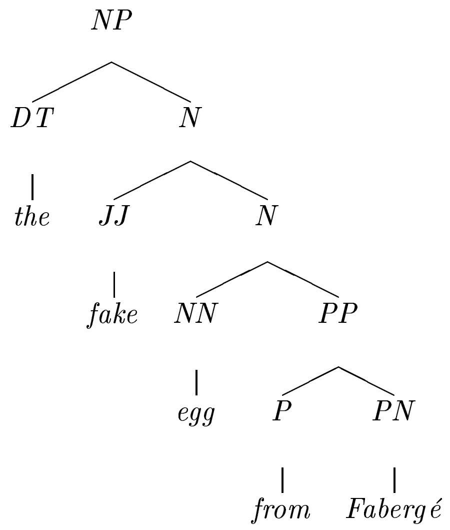
To avoid generating infinitely many ambiguous derivations, empty auxiliary trees (i.e., those anchored by the empty symbol) are disallowed. Moreover, at most one left and one right auxiliary tree may be simultaneously inserted into an adjunction site. Allowing simultaneous adjunction of multiple left and right auxiliary trees would increase the number of derivations exponentially. Simultaneously adjoining one left and one right auxiliary tree into the same site produces two parses with slightly different connotations. For example, consider a set of elementary trees with initial tree, ρ_I, left auxiliary tree, ρ_L, and right auxiliary tree, ρ_R:



Suppose we wish to simultaneously adjoin the auxiliary trees into the initial tree. If the left adjunction took place first, the resulting parse tree is:



This interpretation binds the adjective *fake* closer to the bare noun *egg* to emphasize that a Fabergé egg is not natural. We may wish to apply the left adjunction to the phrase first for a sentence such as: *It may be pretty to look at, but you can't eat the fake egg from Fabergé.* If the right adjunction took place first, the resulting parse tree is:



This interpretation binds the adjective *fake* to the noun phrase *egg from Fabergé* to emphasize that the egg is not from Fabergé. We may wish to apply the right adjunction to the phrase first for a sentence such as: *The craftsmanship of the fake egg from Fabergé pales in comparison to the real thing.*

2.3.2 PLTIG parameters

Just as the basic TAG formalism can be extended to become a lexicalized and stochastic model, similar transformation is possible for the basic TIG formalism. In a Probabilistic Lexicalized TIG, each elementary tree is anchored by a word in the language, and the grammar is parameterized by a set of probabilities that describes the likelihood of applying an operation in each step of deriving a string. The probabilities of a PLTIG make up five types of distributions: start tree distribution, substitution distribution, left adjunction distribution, right adjunction distribution, and simultaneous adjunction precedence distribution.

First, the start tree distribution is needed to determine each initial tree's likelihood of starting the derivation process. The probability of an initial tree ρ being the start tree is denoted as $p_I(\rho)$, and

$$\sum_{\rho \in InitTrees} p_I(\rho) = 1.$$

Second, each substitution node (i.e, a leaf nonterminal) of an elementary tree has a substitution distribution, made up of a set of probabilities between the node and the initial trees that may substitute into it. We denote the probability between a substitution node η_S and an initial tree ρ as $p_S(\eta_S, \rho)$. Note that if tree ρ cannot substitute into node η_S (for example, because their labels do not match), $p_S(\eta_S, \rho) = 0$. Thus, for each η_S,

$$\sum_{\rho \in InitTrees} p_S(\eta_S, \rho) = 1.$$

Third, a left adjunction distribution is associated with each internal nonterminal node of an elementary tree. The distribution specifies the probabilities of each auxiliary tree adjoining into the node as well as the probability that no left adjunction takes place at this node. Let η be an internal nonterminal node, and ρ be a left auxiliary tree. We denote the probability of ρ adjoining into η as $p_L(\eta, \rho)$, and the probability of η taking no left adjunction as $p_{NL}(\eta)$. Thus, for each η,

$$p_{NL}(\eta) + \sum_{\rho \in LAuxTrees} p_L(\eta, \rho) = 1.$$

Similarly, a right adjunction distribution is associated with each internal nonterminal node of an elementary tree and describes the likelihoods of some right auxiliary tree adjoining into the node. In other words,

$$p_{NR}(\eta) + \sum_{\rho \in RAuxTrees} p_R(\eta, \rho) = 1.$$

Finally, the simultaneous adjunction precedence distribution is associated with each internal node, specifying, in case of a simultaneous adjunction at this node, whether to adjoin the left auxiliary tree first or the right auxiliary tree first. We denote the probability of adjoining a left auxiliary tree into a node η before the right as $p_{LR}(\eta)$ and the probability of adjoining a right tree before the left as $p_{RL}(\eta)$. And,

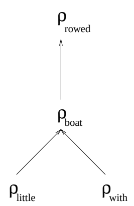
$$p_{LR}(\eta) + p_{RL}(\eta) = 1$$

2.3.3 A PLTIG example

Figure 2.9 depicts a small PLTIG for a simple English-like artificial language similar to the earlier CFG example. It has 4 initial trees, but only two of them (those whose roots are labeled S) can act as the start tree of a sentence; it also has a left and a right auxiliary tree that can modify nouns. Using this grammar to parse the sentence "*I rowed a little boat with holes.*" would result in two possible derivations, depending on the ordering of the left and right auxiliary modifiers.[7] The grammar assigns the parse in which the adjectival left auxiliary tree binds to the noun before the prepositional right auxiliary tree a likelihood value of 0.016632:

$$p_I(\rho_{rowed}) \times p_S(\eta_{(1)}, \rho_{boat}) \times p_L R(\eta_{(3)}) \times p_L(\eta_{(3)}, \rho_{little}) \times p_R(\eta_{(3)}, \rho_{with})$$

[7]Note, however, that even though there are two derived trees (i.e. parse trees), only one derivation tree is needed:



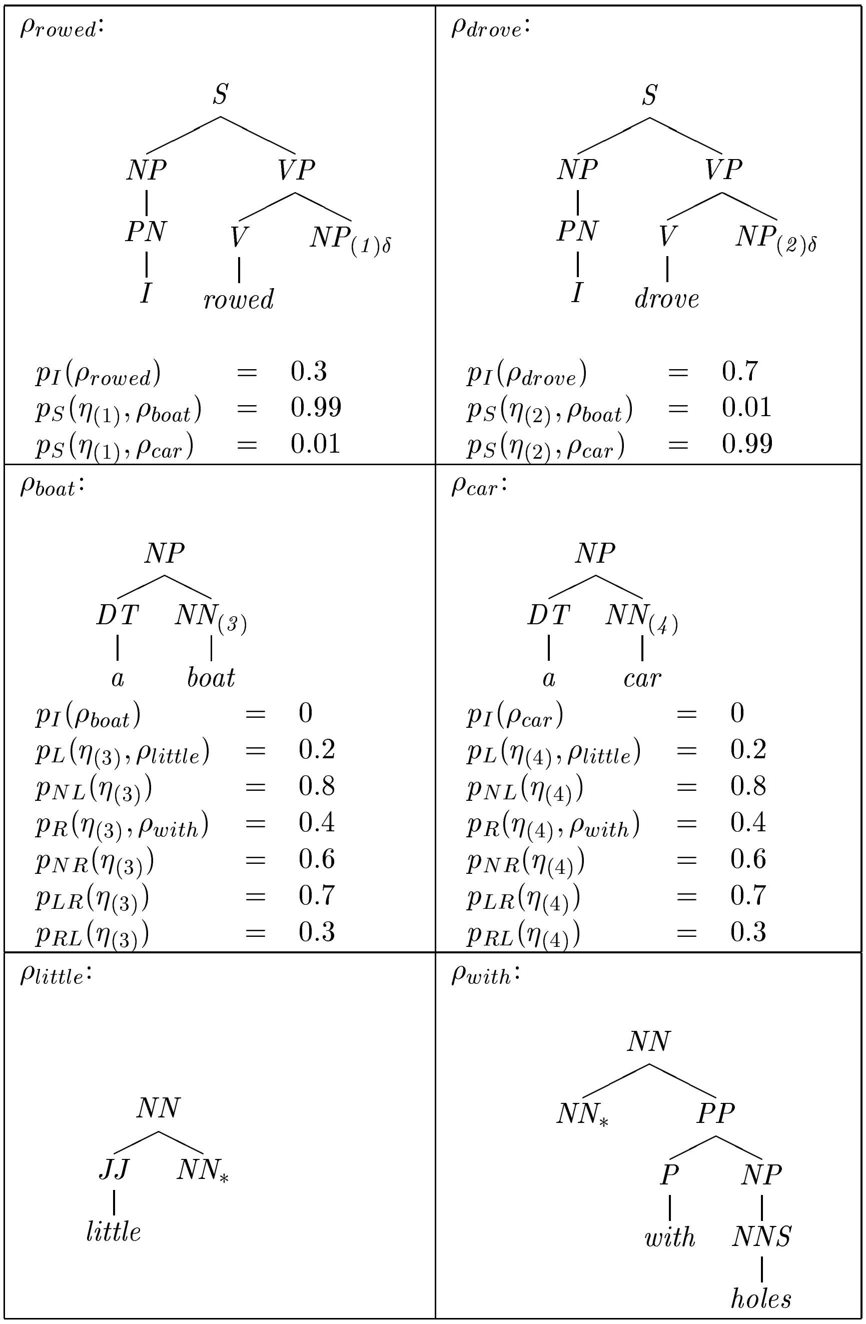


Figure 2.9: A simple PLTIG with 4 initial trees, 1 left auxiliary tree and 1 right auxiliary tree. Nodes on which adjunction or substitution is permitted have subscripts to facilitate referencing (e.g., the node $\eta_{(1)}$ would have the label $NP_{(1)}$).

$$= \quad 0.3 \times 0.99 \times 0.7 \times 0.2 \times 0.4$$

$$= \quad 0.016632,$$

and it assigns the parse in which the right auxiliary tree binds to the noun before the left with a likelihood value of 0.007128:

$$p_I(\rho_{rowed}) \times p_S(\eta_{(1)}, \rho_{boat}) \times p_R L(\eta_{(3)}) \times p_L(\eta_{(3)}, \rho_{little}) \times p_R(\eta_{(3)}, \rho_{with})$$

$$= \quad 0.3 \times 0.99 \times 0.3 \times 0.2 \times 0.4$$

$$= \quad 0.007128.$$

The likelihood that the grammar would generate the sentence is the sum of the two parses, 0.02376.

The parameters of a PLTIG represent the relationship between two lexical items so that the problem we observed earlier in PCFGs (see Figure 2.4) does not arise as frequently. For example, the probability that a sentence such as "*I rowed a little car with holes.*" would be modeled by the above PLTIG is low (0.00024) because the parameter $p_S(\eta_{(1)}, \rho_{car})$ has a small value. More importantly, unlike the lexical relationships modeled by a bi-gram, the relationship (or lack thereof) between *rowed* and *car* does not change even as other modifiers (such as *little*) are inserted between them.

2.4 Chapter Summary

In this chapter, we have reviewed several kinds of formalisms with different strengths and weaknesses in terms of the expressive power, the computational complexity, the applicability to NLP tasks, and the ease of training these models. The PLTIG formalism provides an attractive compromise between linguistic plausibility and computational tractability offered by the following formalisms:

N-gram models can generate regular expressions with a time complexity of $O(n)$ for processing a sentence of length n. The model captures some lexical dependencies between words (with the constraint that the words are within a window size of N), therefore it can be used for language modeling. However, its low expressive power restricts its ability to produce hierarchical parse trees. It can be trained directly from raw text.

PCFGs can generate context-free languages with a time complexity of $O(n^3)$. A trained PCFG can perform both language modeling and parsing. However, because its production rules are not necessarily lexicalized, it typically performs poorly for language modeling. While there exists a heuristic that induces grammars from raw text, it typically converges on poor locally optimal solutions. When the training data also contains some parsing information, better grammars are induced.

PLTAG can generate mildly context-sensitive languages with a time complexity of $O(n^6)$. Its tree representation models both structural and lexical relationships in one formalism, suggesting that it may be good for both language modeling and parsing; however, the high time complexity makes the automatic induction of a PLTAG an impracticality for now.

Because the PLTIG formalism is context-free equivalent, it can, like PCFGs, model the structural relationship between constituents, which is necessary for the task of parsing. By requiring each elementary tree to be anchored by a word, it can, like N-gram networks, model the dependency relationship between words, which is important for the task of language modeling. Moreover, using the adjunction operator as a primitive for building the composite parse trees, like LTAGs, it can model dependencies between words *even when they are far apart.* Finally, the restrictions placed on the adjunction operator ensures the time complexity of parsing a sentence with n words to be in $O(n^3)$ rather than $O(n^6)$ needed for LTAG.

In the next chapter, we will present our learning algorithm for inducing a PLTIG. We will also empirically compare the induced PLTIGs to those induced by other formalisms to validate our claim that PLTIGs can be trained within a reasonable amount of time and that they perform well on both tasks of parsing and language modeling.

Chapter 3

Inducing PLTIGs

3.1 Introduction

A primary motivation for working with probabilistic grammars is to automate the grammar induction process. Ideally, we would like the induction procedure to be *unsupervised*, in which the learning algorithm does not require a *teacher* (usually a human) to guide the learning process. For an unsupervised grammar inducer the training data are simply large collections of raw text. Unfortunately, unsupervised grammar induction in the natural language processing domain typically does not result in high-quality grammars. For example, in the study by Pereira and Schabes (1992), they found that unsupervised grammar induction converges on poor grammars even for the relatively simple domain of the Air Traffic Information System corpus (Hemphill *et al.*, 1990), in which the sentences are short and have regular structural patterns.

To make the task of learning a grammar easier, the raw text training sentences are often augmented with additional linguistic information, typically provided by humans. This can be seen as a form of supervised learning, in which the human teacher demonstrates the correct ways of analyzing sentences to the learning system through the annotated training data. To some degree, the difficulty of grammar induction depends on the quality of the annotated information added to the training data. One source of information that has proved to be useful is the structural analyses of the training sentences. When the training corpus consists of a large reservoir of fully annotated parse trees, it is possible to directly extract a grammar based on these parse trees. The success of recent high-quality parsers (Charniak, 1997; Collins, 1997; Chiang, 2000) and language models (Chelba and Jelinek, 1998) relies on the availability of such detailed treebank corpora. Other types of information may also be helpful, but require much more human involvement. For instance, Hermjakob and Mooney (1997) have described a learning system that can build a deterministic shift-reduce parser from a small set of training examples with the aid of detailed morphological, syntactical, and semantic knowledge databases and step-by-step guidance from human experts.

The induction task becomes more challenging as the amount of supervision in the training data decreases. To compensate for the missing information, the learning process requires heuristic search to find locally optimal grammars. One form of partially supervised data might specify the phrasal boundaries without specifying their labels by bracketing each constituent unit with a pair of parentheses (McNaughton, 1967). For example, the parse tree for the sentence "*Several fund managers expect a rough market this morning before prices stabilize.*" is labeled as "((Several fund managers) (expect ((a rough market) (this morning)) (before (prices stabilize))).)" As shown in Pereira and Schabes (1992), an unsupervised learner based on the Expectation-Maximization (EM) algorithm (Dempster *et al.*, 1977) can be modified to take advantage of these bracketing constraints. In this thesis, we are interested in furthering this line of research because it holds the promise of automating the learning process with the least amount of additional information.

Previously, researchers have applied the EM algorithm to inducing HMMs (Baum, 1972) and PCFGs (Baker, 1979; Lari and Young, 1990). In this chapter, we show that the EM principle can also be applied to induce PLTIGs. In Section 3.2, we discuss the EM principle and how it relates to grammar induction. In Section 3.3, we derive the relevant equations for inducing PLTIGs, and address implementational issues such as computational efficiency and data sparsity to make inducing PLTIGs a practical and efficient solution for NLP tasks. In Section 3.4, we report the experimental results of inducing PLTIGs from large corpora. Our findings show that the induced PLTIGs perform well on both language modeling and parsing tasks, capturing both lexical dependency and structural expressivity in one formalism.

3.2 The EM Algorithm

The EM learning algorithm operates under the maximum-likelihood estimation (MLE) principle. It iteratively updates the parameters of the hypothesis (in our case, the adjunction probabilities of a PLTIG; in the case of a PCFG, the probabilities associated with the production rules) to maximize the expected likelihood of the observed data (i.e., the training sentences). A significant characteristic of the EM-styled learning algorithms is that the induction process is unsupervised. The lack of supervision can be seen as a form of *missing data*, which corresponds to the derivations of the sentences in the grammar induction domain. Moreover, there is a many-to-one mapping between the missing data and the observed data. For example, a sentence may have many parses. With proper conditioning, the observed data is guaranteed to be more likely under the updated parameters. More formally the basic EM theorem says:

Theorem 1 *Let θ_0 denote the old values of the model's parameters, and θ_1 denote the new values. Suppose that v is a set of possible hidden data corresponding to the observed data, w, and let V and W be two random variables; if*

$$E_v[\log Pr_{\theta_1}(V, W = w)] > E_v[\log Pr_{\theta_0}(V, W = w)]$$

then

$$Pr_{\theta_1}(W = w) > Pr_{\theta_0}(W = w).$$

That is, if the expected log likelihood of the complete data (i.e., both the observed data and all possible missing data) under the new parameters is bigger than that under the old parameters, then the probability of the observed data is guaranteed to be higher under the updated parameters. An easy to follow proof of this theorem can be found in Jelinek (1997).

Taking a more algorithmic interpretation of the theorem, we see that in order to improve the current parameter setting (the consequence clause of the theorem), we need to maximize the expectation of the log joint likelihood of the observed and missing data (the antecedent of the theorem) with respect to the parameter variable θ. Expressing the density function in terms of the current setting of the parameters, θ_0, the expectation is expanded to:

$$E[\log Pr_\theta(V, W = w)] = \sum_{\{v\}} Pr_{\theta_0}(v \mid w) \log Pr_\theta(v, w)$$

In the maximization step, we differentiate the above expression to find a new setting for θ that maximizes the expectation. An algorithmic overview of the process (in terms of inducing PLTIGs) is shown in Figure 3.1.

Applying the EM principle to the domain of grammar induction, we denote an observed sentence as $\mathbf{w} = w_1, \ldots, w_T$. The sentence has a set of derivations, denoted as $\mathcal{V} = \{\mathbf{v} : \mathbf{v}$ *is one derivation of* $\mathbf{w}\}$. Since we cannot distinguish the true parse of the sentence, the set of derivations is the missing data. Each derivation of the sentence further decomposes into a set of parsing actions chosen at each step of the derivation (e.g., the expansion of a production rule for PCFGs; the adjunction of an auxiliary tree into an adjunction site for PLTIGs). We denote one derivation path as $\mathbf{v} = \{v_{(s,t)} : v$ *specifies the step in which the substring* $w_{s+1}, \ldots, w_t$ *forms a constituent in the parse tree*$\}$. Moreover, $act(v_{(s,t)}) = \alpha_{ij}$ represents the particular rule chosen at that step, denoting that the j^{th} possible action of the i^{th} distribution is applied at step $v_{(s,t)}$. For example, in a PCFG, α_{ij} may specify the expansion of the j^{th} production rule of nonterminal X_i; in a PLTIG, it may specify the adjunction of the j^{th} auxiliary tree into adjunction site η_i. The parameters of the grammar model the likelihoods of all possible parsing actions, denoted as $\theta = \{p_{ij} : p_{ij} = Pr(\alpha_{ij})\}$. Thus, to maximize $E[\log Pr_\theta(V, W = w)]$, we expand and differentiate the expression with respect to each parameter p_{ij} and set it to 0[1]:

$$\frac{\partial}{\partial p_{ij}} \left[\sum_{\mathbf{v} \in \mathcal{V}} Pr_{\theta_0}(\mathbf{v} \mid \mathbf{w}) \log Pr_\theta(\mathbf{v}, \mathbf{w}) - \sum_m \lambda_m \sum_r p_{mr} \right] \stackrel{set}{=} 0 \tag{3.1}$$

$$\Rightarrow \sum_{\mathbf{v} \in \mathcal{V}} Pr_{\theta_0}(\mathbf{v} \mid \mathbf{w}) \frac{(\frac{\partial}{\partial p_{ij}}) Pr_\theta(\mathbf{v}, \mathbf{w})}{Pr_\theta(\mathbf{v}, \mathbf{w})} = \lambda_i \tag{3.2}$$

$$\Rightarrow \sum_{\mathbf{v} \in \mathcal{V}} \frac{Pr_{\theta_0}(\mathbf{v}, \mathbf{w})}{Pr_{\theta_0}(\mathbf{w})} \frac{(\frac{\partial}{\partial p_{ij}}) Pr_\theta(\mathbf{v}, \mathbf{w})}{Pr_\theta(\mathbf{v}, \mathbf{w})} = \lambda_i \tag{3.3}$$

[1]In Equation (3.1), the numbers λ_m are Lagrangian multipliers to enforce the condition that $\sum_r p_{mr} = 1$, where m is an index over all the distributions (number of nonterminals for PCFG or adjunction sites for PLTIGs) and r is an index over the possible events (rules for PCFG or auxiliary trees for PLTIGs).

Since $\mathbf{v}$ determines the derivation of $\mathbf{w}$,

$$\begin{aligned} Pr_\theta(\mathbf{v},\mathbf{w}) &= Pr_\theta(\mathbf{v}) = \prod_{v\in\mathbf{v}} Pr(act(v)) \\ &= \prod_{v\in\mathbf{v},act(v)\neq\alpha_{ij}} Pr(act(v)) \times p_{ij}^{c_{ij}(\mathbf{v})} \end{aligned} \tag{3.4}$$

where $c_{ij}(\mathbf{v})$ denote the number of times that action α_{ij}, is used in the derivation $\mathbf{v}$. Then,

$$\frac{\frac{\partial}{\partial p_{ij}} Pr_\theta(\mathbf{v},\mathbf{w})}{Pr_\theta(\mathbf{v},\mathbf{w})} = \frac{c_{ij}(\mathbf{v})}{p_{ij}}.$$

Substituting it back to Equation (3.3), we get

$$\sum_{\mathbf{v}\in\mathcal{V}} \frac{Pr_{\theta_0}(\mathbf{v},\mathbf{w})}{Pr_{\theta_0}(\mathbf{w})} \frac{c_{ij}(\mathbf{v})}{p_{ij}} = \lambda_i.$$

Solving for the parameter of interest, we update the value of p_{ij} to be:

$$\hat{p}_{ij} = \frac{\sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w}) c_{ij}(\mathbf{v})}{\lambda_i Pr_{\theta_0}(\mathbf{w})}. \tag{3.5}$$

The denominator $\lambda_i Pr_{\theta_0}(\mathbf{w})$ acts as a normalization factor to ensure that the i^{th} distribution sums to 1. In other words,

$$\begin{aligned} &\sum_j \hat{p}_{ij} = \sum_j \frac{\sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w}) c_{ij}(\mathbf{v})}{\lambda_i Pr_{\theta_0}(\mathbf{w})} = 1 \\ &\Rightarrow \lambda_i = \sum_j \frac{\sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w}) c_{ij}(\mathbf{v})}{Pr_{\theta_0}(\mathbf{w})} \end{aligned}$$

Initialize the parameters of a PLTIG in *canonical form*.
Repeat
  E-Step
    Compute the *expected* log likelihood of deriving the training sentences with the current grammar.
  M-Step
    Update the parameters of the grammar to *maximize* the likelihood deriving the training sentences.
Until Convergence

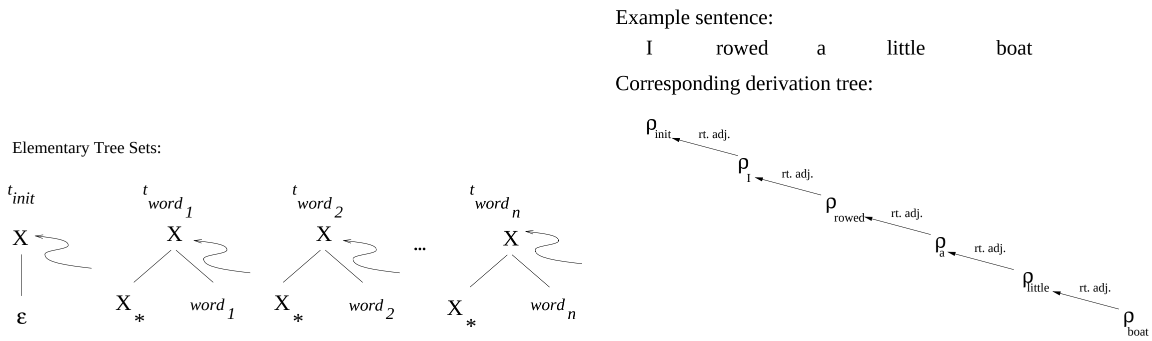
Figure 3.1: A high-level description of the PLTIG learning algorithm

3.3 The PLTIG Induction Algorithm

In this section, we show how to modify the general EM algorithm for the specific task of inducing PLTIGs. For the initialization step, we need to determine a canonical PLTIG tree representation. For the E-Step, we derive the necessary equations for computing the likelihoods of the derivations, and show that they can be computed efficiently through dynamic programming. In the M-Step, we discuss dealing with the real-world constraints of over-fitting to the training data and data sparsity.

3.3.1 Initialization

Because the EM process iteratively refines the parameters, the initial grammars must over-generalize rather than under-generalize. Therefore, the initial grammar must be able to generate any string. In the case of a PCFG, an initial grammar may assign non-zero probabilities to all possible production rules expressed in Chomsky Normal Form. For a PLTIG, we need to specify a *canonical tree representation* for the elementary trees that allows any word to modify another word. A simple PLTIG that fits the requirement is one that simulates a bigram model. It is represented by a tree set that contains a right auxiliary



(a) (b)

Figure 3.2: (a) A set of elementary PLTIG trees that represent a bigram grammar. The arrows indicate adjunction sites. To simplify the diagram, we have omitted the parameters in the figure. (b) An example sentence. Because each tree is right adjoined to the tree anchored with the neighboring word in the sentence, the only structure is right branching

tree for each lexical item as depicted in Figure 3.2(a). Each tree has one adjunction site into which other right auxiliary trees can adjoin. The tree set has only one initial tree, which is anchored by an empty lexical item. The initial tree represents the start of the sentence. Any string can be constructed by right adjoining the words together in order. Training the parameters of this grammar yields the same result as a bigram model: the parameters reflect close correlations between words that are frequently seen together, but the model cannot provide any high-level linguistic structure. (See example in Figure 3.2(b).)

To generate non-linear structures, we need to allow adjunction in both left and right directions. The expanded PLTIG tree set includes a left auxiliary tree representation as well as right for each lexical item. Moreover, we must modify the topology of the auxiliary trees so that adjunction in both directions can occur. We insert an intermediary node between the root and the lexical anchor. At this internal node, at most one adjunction of each direction may take place. The introduction of this node is necessary because the definition of the formalism disallows right adjunction into the root node of a left auxiliary tree and vice versa. For the sake of uniformity, we shall disallow adjunction into the root

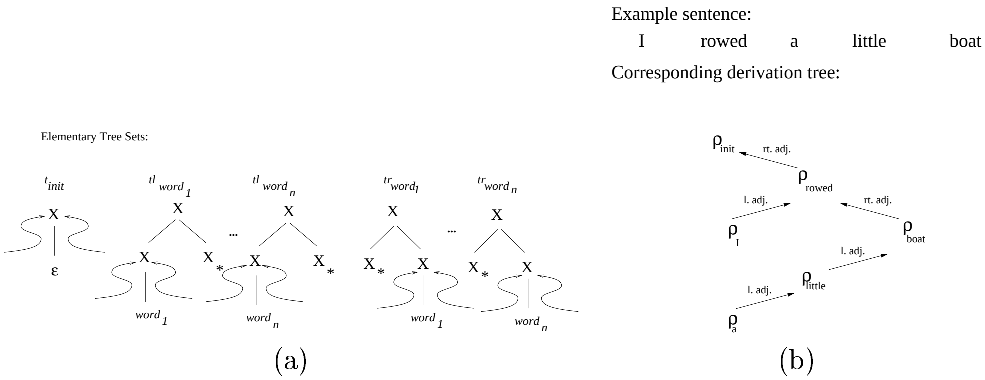


Figure 3.3: (a) A PLTIG elementary tree set that allow both left and right adjunctions. To simplify the diagram, we have omitted the parameters in the figure. (b) With both left and right adjunctions possible, the sentences can be parsed in a more hierarchical manner.

nodes of the auxiliary trees from now on. Figure 3.3 shows a PLTIG that allows at most one left and one right adjunction for each elementary tree. This enhanced PLTIG can produce hierarchical structures that the bigram model could not (See Figure 3.3.) It is, however, still too limiting to allow only one modifier from each direction. Many words often require more than one modifier. In the above example, the PLTIG in Figure 3.3 fails to generate the derivation in which both the determiner *a* and the adjective *little* modify the object noun *boat*.

To create more adjunction sites for each word, we introduce more intermediary nodes between the root and the lexical anchor. Our empirical studies show that each lexicalized auxiliary tree requires at least 3 adjunction sites to parse all the sentences in the corpora. Figure 3.4(a) and (b) show two examples of auxiliary trees with 3 adjunction sites. The number of parameters in a PLTIG is dependent on the number of adjunction sites just as the size of a PCFG is dependent on the number of nonterminals. For a language with W

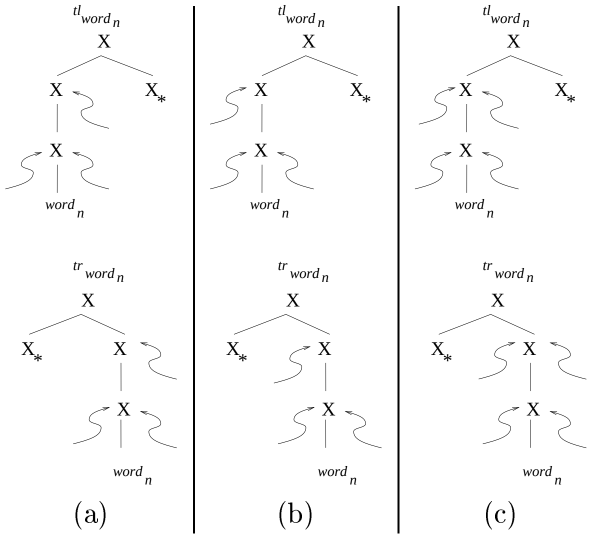


Figure 3.4: Prototypical auxiliary trees for three PLTIGs: (a) L1R2, (b) L2R1, and (c) L2R2.

vocabulary items, the number of parameters for the type of PLTIGs used in this paper is

$$2(W+1) + 2W(K)(W+1), \tag{3.6}$$

where K is the number of adjunction sites per tree. The first term of the equation is the number of parameters contributed by the initial tree, which always has two adjunction sites in our experiments. The second term is the contribution from the auxiliary trees. There are $2W$ auxiliary trees, each tree has K adjunction sites; and $W+1$ parameters describe the distribution of adjunction at each site. The number of parameters of a PCFG with M nonterminals is $M^3 + MW$. For the experiments, we try to choose values of K and M for the PLTIGs and PCFGs such that

$$2(W+1) + 2W(K)(W+1) \approx M^3 + MW$$

3.3.2 E-Step

As described in Section 3.2, the E-Step is responsible for computing two quantities for all parameters p_{ij}:

$$Pr_{\theta_0}(\mathbf{w}) = \sum_{\mathbf{v}\in\mathcal{V}} Pr(\mathbf{v}) \tag{3.7}$$

$$\sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w})c_{ij}(\mathbf{v}) \tag{3.8}$$

Since there may be exponentially many parses for each sentence, it is not practical to compute the likelihood of each parse individually. Using dynamic programming techniques, one can compute Expression (3.7) and Expression (3.8) within polynomial time. The key idea is to avoid repeating the same computations by factoring out common terms and storing them as intermediary probabilities in a table. For Context-Free formalisms, these intermediary probabilities are called the *inside* and *outside* probabilities (Lari and Young, 1990)[2]. Expression (3.8) can be rewritten in terms of these probabilities. Later in this section, we will derive the equations for the inside and outside probabilities in PLTIGs; and we will describe how to use them to compute Expression (3.8) for updating PLTIG parameters in Section 3.3.3. For introductory purposes, we present the concept of inside and outside probabilities in terms of the PCFG formalism.

Every possible pairing of a nonterminal symbol and a substring of the sentence has both an inside and an outside probability associated with it. The inside probability for a pair of a nonterminal X and a substring between positions s and t, denoted as $e(X, s, t)$, represents

[2]Intermediary probabilities are called the *backward* and *forward* probabilities for HMMs (Baum, 1972)

the likelihood of X deriving the substring. In other words,

$$e(X, s, t) = Pr_{\theta_0}(X \stackrel{*}{\Rightarrow} w_{s+1}, \ldots, w_t).$$

Note that Expression (3.7) can be rephrased as the inside probability of the start symbol over the entire string, $e(S, 0, T)$. The outside probability of symbol X for the substring $w_{s+1} \ldots, w_t$, denoted as $f(X, s, t)$, represents the likelihood of the partially completed derivation in which all the words outside of the substring have been derived with X being the only unexpanded nonterminal remaining. That is,

$$f(X, s, t) = Pr_{\theta_0}(S \stackrel{*}{\Rightarrow} w_1, \ldots, w_s \; X \; w_{t+1}, \ldots, w_T).$$

The outside probability complements the inside probability. Their product represents the likelihood of X deriving the substring $w_{s+1}, \ldots, w_t$ for the derivation of the sentence.

To rewrite Expression (3.8) in terms of inside and outside probabilities, we begin by observing that $c_{ij}(\mathbf{v})$ can be rewritten as $\sum_{v \in \mathbf{v}} \delta(act(v), \alpha_{ij})$.[3] Note that a substring (specified by its boundary positions s and t) is implicitly associated with each derivation step $v \in bfv$. Thus, substituting back into Expression (3.8), we get

$$\begin{aligned} \sum_{\mathbf{v} \in \mathcal{V}} Pr_{\theta_0}(\mathbf{v}, \mathbf{w}) c_{ij}(\mathbf{v}) &= \sum_{\mathbf{v} \in \mathcal{V}} (Pr_{\theta_0}(\mathbf{v}, \mathbf{w}) \sum_{v \in \mathbf{v}} \delta(act(v), \alpha_{ij})) \\ &= \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} \sum_{\mathbf{v} \in \mathcal{V}_{(s,t)}} Pr_{\theta_0}(\mathbf{v}, \mathbf{w}), \end{aligned}$$

[3]Here,

$$\delta(x, y) = \begin{cases} 1 & : \; x = y \\ 0 & : \; \text{otherwise} \end{cases}$$

where $\mathcal{V}_{(s,t)} \subseteq \mathcal{V}$ is the subset of derivations in which the derivation step specified by positions s and t does take action α_{ij} (i.e., $act(v_{(s,t)}) = \alpha_{ij}$). As shown in Equation (3.4), the probability of one complete derivation is the product of the likelihood of each step; so we reorganize the right hand side of the above equation by factoring out p_{ij}, the common term at step $v_{(s,t)}$. The remaining terms belong to one of two groups: those derivation steps for substrings *outside* of positions s and t (characterized by $f(X_i, s, t)$), and those properly contained inside s and t. The grouping of the inside terms depends on the form of the action. For example, if action α_{ij} specifies a single nonterminal expansion, $X_i \rightarrow X_j$, then

$$\begin{aligned} \sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w})c_{ij}(\mathbf{v}) &= \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} (p_{ij} \sum_{\mathbf{v}\in\mathcal{V}_{(s,t)}} \prod_{v\neq v_{(s,t)}} Pr(act(v))) \\ &= p_{ij} \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(X_i, s, t) e(X_j, s, t). \end{aligned} \tag{3.9}$$

On the other hand, if action α_{ij} specifies that node X_i expands to two components, X_{j_1} and X_{j_2}, then the inside terms can be further separated into two subgroups: those that are dominated by X_{j_1} and those that are dominated by X_{j_2}. However, because action α_{ij} does not specify *how* to subdivide X_i, we must consider all possible division points between s and t. In other words,

$$\sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w})c_{ij}(\mathbf{v}) = p_{ij} \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(X_i, s, t) \sum_{r=s}^{t} e(X_{j_1}, s, r) e(X_{j_2}, r, t). \tag{3.10}$$

In general, if the action specifies that X_i expands into k components, the inside terms would be divided into k subgroups accordingly. Because the parse trees in the experiment section have at most two branches per node, we only need Equation (3.9) and Equation (3.10).

Inside Probabilities for PLTIGs

Due to the adjunction operation, the definition of inside probabilities for PLTIGs is more complex than that of the PCFGs. In a PLTIG, a substring $w_{s+1}, \ldots, w_t$ can be subsumed by a node η in one of five ways.[4] Each case contributes to the final (total) inside probability of $e(\eta, s, t)$. We denote them as follows:

1. $e(\eta, s, t, \emptyset)$: the probability that η dominates $w_{s+1}, \ldots, w_t$ without any trees adjoining into it.

2. $e(\eta, s, t, L)$: the probability that η dominates $w_{s+1}, \ldots, w_t$ as the result of exactly one left adjunction.

3. $e(\eta, s, t, R)$: the probability that η dominates $w_{s+1}, \ldots, w_t$ as the result of exactly one right adjunction.

4. $e(\eta, s, t, LR)$: the probability that η dominates $w_{s+1}, \ldots, w_t$ as the result of a simultaneous left and right adjunction, with the left adjunction taking precedence over the right.

5. $e(\eta, s, t, RL)$: the probability that η dominates $w_{s+1}, \ldots, w_t$ as the result of a simultaneous left and right adjunction, with the right adjunction taking precedence over the left.

---

[4] Recall from the definition of the formalism in Section 2.3 that a node cannot have more than one left or right adjunction.

The overall inside probability $e(\eta, s, t)$ is the normalized sum of these five parts. Its composite value corresponds to:[5]

$$
\begin{aligned}
e(\eta, s, t) &= p_{NL}(\eta)p_{NR}(\eta)e(\eta, s, t, \emptyset) \\
&\quad + p_{NR}(\eta)e(\eta, s, t, L) \\
&\quad + p_{NL}(\eta)e(\eta, s, t, R) \\
&\quad + e(\eta, s, t, LR) \\
&\quad + e(\eta, s, t, RL)
\end{aligned}
$$

The inside probabilities for all nodes and substrings are computed recursively. There are two possible base cases: an empty substring, and a substring made up of exactly one word. The lowest node to subsume an empty string is either a foot node or a node whose label is the empty token. The lowest node to subsume a single word substring is one whose label is a lexical item. The inside probabilities of these nodes are defined to be 1.

$$
\begin{aligned}
e(\eta, s, s) &= \begin{cases} 1 & : \; Foot(\eta) \text{ or } Label(\eta) = \epsilon \\ 0 & : \; \text{otherwise} \end{cases} \\
\epsilon(\eta, s, s+1) &= \begin{cases} 1 & : \; Label(\eta) = w_{s+1} \\ 0 & : \; \text{otherwise} \end{cases}
\end{aligned}
$$

For the general case, we express each component of an inside probability in terms of other inside probability components that have been already computed.

First, for the no-adjunction component, $e(\eta, s, t, \emptyset)$, we need to consider three possible elementary tree configurations (shown in Figure 3.5): η is the parent of two children η_1 and

[5]If η is a substitution node, then $e(\eta, s, t) = e(\eta, s, t, \emptyset)$, since adjunction is not allowed at a substitution node.

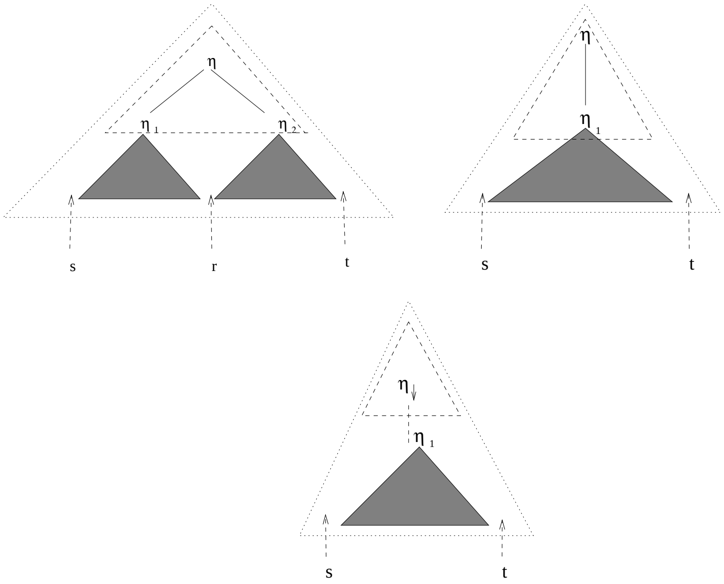


Figure 3.5: Three configurations in which η contributes to $e(\eta, s, t, \emptyset)$: η with two children; η with one child; and η as a substitution node at the frontier of a tree. The dashed triangle represents the elementary tree containing η; the dotted triangle represents the full derived tree The grey triangles represent the portions whose inside probabilities have been already calculated.

η_2; η is the parent of a single child η_1; and η is a frontier substitution node. If node η has two children such that η_1 is the left child and η_2 is the right child, then

$$e(\eta, s, t, \emptyset) = \sum_{r=s}^{t} e(\eta_1, s, r) e(\eta_2, r, t).$$

If node η has one child η_1, then

$$e(\eta, s, t, \emptyset) = e(\eta_1, s, t).$$

If node η is a substitution node, then

$$e(\eta, s, t, \emptyset) = e(Root(\rho_S), s, t).$$

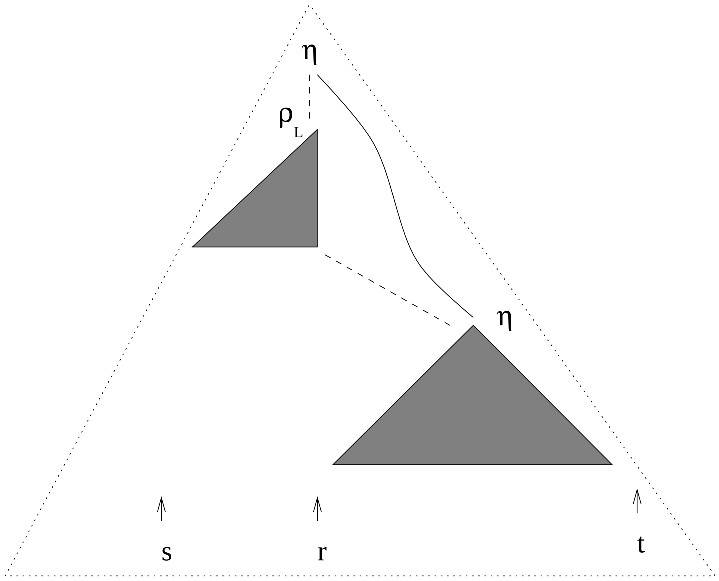


Figure 3.6: One term (i.e., for a fixed r value) in the computation of $e(\eta, s, t, L)$, the left-adjunction component of an inside probability.

Next, for the left-adjunction component, $e(\eta, s, t, L)$, we need to consider all possible adjunctions into η from any left auxiliary tree such that the auxiliary tree dominates some prefix of $w_{s+1}, \ldots, w_t$ (say at some position r), and η dominates the rest of the substring *without any other adjunctions* (see Figure 3.6). Summing over all possible trees and break positions, we get

$$e(\eta, s, t, L) = \sum_{\rho_L} \sum_{r=s+1}^{t} p_L(\eta, \rho_L) e(Root(\rho_L), s, r) e(\eta, r, t, \emptyset).$$

Similarly, for $e(\eta, s, t, R)$, we iterate over all possible right auxiliary trees and break positions (see Figure 3.7:

$$e(\eta, s, t, R) = \sum_{\rho_R} \sum_{r=s}^{t-1} p_R(\eta, \rho_R) e(Root(\rho_R), r, t) e(\eta, s, r, \emptyset).$$

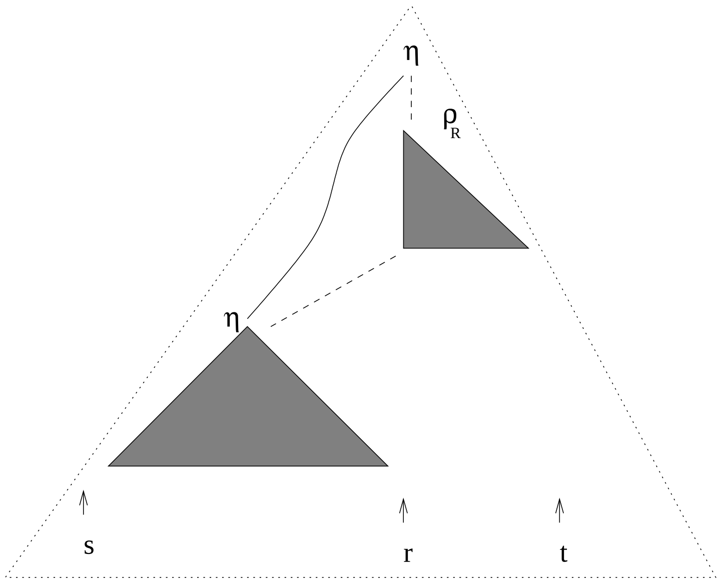


Figure 3.7: One term (i.e., for a fixed r value) in the computation of $e(\eta, s, t, R)$, the right-adjunction component of an inside probability.

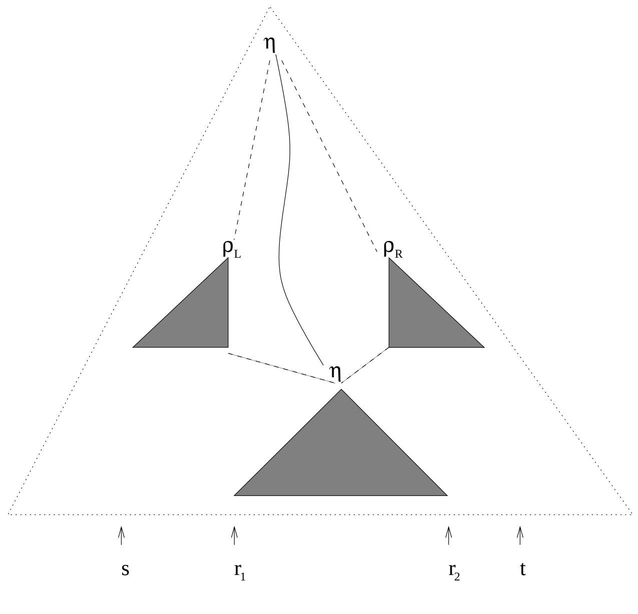


Figure 3.8: One term (i.e., for fixed r_1 and r_2) in the computation of $e(\eta, s, t, RL)$ and $e(\eta, s, t, LR)$, the simultaneous adjunction components of an inside probability. This diagram does not specify the order of the adjunctions.

Finally, for the two simultaneous adjunction components, $e(\eta, s, t, LR)$ and $e(\eta, s, t, RL)$, we consider all possible ways of dividing the substring into three pieces (say at positions $r1$ and $r2$) such that the left auxiliary tree dominates the prefix, the right auxiliary tree dominates the suffix, and node η dominates the remaining middle portion. Note that the product of the inside probabilities of the three small pieces does not specify the precedence ordering of the auxiliary trees (see Figure 3.8). In fact, the two components can be thought of as two weighted portions (specified by $p_{LR}(\eta)$ and $p_{RL}(\eta)$ of this quantity. From an efficiency perspective, however, dividing a substring into three pieces is undesirable, as it requires a complexity of $O(n^4)$. A more efficient way of viewing the simultaneous adjunction process is to reduce it to the one-sided adjunction case with the constraint that the auxiliary tree with higher precedence has already been adjoined.

$$\begin{aligned}
e(\eta, s, t, LR) &= p_{LR}(\eta) \sum_{\rho_L} \sum_{\rho_R} \sum_{r_1=s+1}^{t-1} \sum_{r_2=r_1}^{t-1} \left(\begin{array}{c} p_R(\eta, \rho_R) p_L(\eta, \rho_L) e(\eta, r_1, r_2, \emptyset) \\ \times e(Root(\rho_L), s, r_1) e(Root(\rho_R), r_2, t) \end{array} \right) \\
&= p_{LR}(\eta) \sum_{\rho_R} \sum_{r_2=s+1}^{t-1} p_R(\eta, \rho_R) e(Root(\rho_R), r_2, t) e(\eta, s, r_2, L) \\
e(\eta, s, t, RL) &= p_{RL}(\eta) \sum_{\rho_L} \sum_{\rho_R} \sum_{r_1=s+1}^{t-1} \sum_{r_2=r_1}^{t-1} \left(\begin{array}{c} p_R(\eta, \rho_R) p_L(\eta, \rho_L) e(\eta, r_1, r_2, \emptyset) \\ \times e(Root(\rho_L), s, r_1) e(Root(\rho_R), r_2, t) \end{array} \right) \\
&= p_{RL}(\eta) \sum_{\rho_L} \sum_{r_1=s+1}^{t-1} p_L(\eta, \rho_L) e(Root(\rho_L), s, r_1) e(\eta, r_1, t, R) \\
\frac{e(\eta, s, t, LR)}{p_{LR}(\eta)} &= \frac{e(\eta, s, t, RL)}{p_{RL}(\eta)}.
\end{aligned}$$

Because the inside probabilities of a grammar are computed recursively in a bottom-up manner, they can be seen as a statistical analog of the traditional CKY-styled chart parsing. Figure 3.9 is the pseudo-code for parsing a sentence with PLTIGs.

Initialize:
for i = 0 **to** $|\mathbf{w}|$ **do**
  $\forall$ nodes η such that η is a foot node or an empty node
    $e(\eta, i, i) \leftarrow 1$
for i = 1 **to** $|\mathbf{w}|$ **do**
  $e(Node(w_i), i-1, i) \leftarrow 1$

Main Loop:
for len = 2 **to** $|\mathbf{w}|$ **do**
  for $s = 0$ **to** $|\mathbf{w}| - len$ **do**
    $t \leftarrow s + len$
    for r = s **to** t **do**
      $\forall$ nodes η
        if $\eta_1 = LeftChild(\eta), e(\eta_1, s, r) > 0$ and $\eta_2 = RightChild(\eta), e(\eta_2, r, t) > 0$
          update $e(\eta, s, t, \emptyset)$.
        if $e(\eta, r, t, \emptyset) > 0$ or $e(\eta, r, t, R)$ and for all ρ_L such that $e(Root(\rho_L), s, r) > 0$
          update $e(\eta, s, t, L)$ and $e(\eta, s, t, RL)$.
        if $e(\eta, s, r, \emptyset) > 0$ or $e(\eta, s, r, L)$ and for all ρ_R such that $e(Root(\rho_R), r, t) > 0$
          update $e(\eta, s, t, R)$ and $e(\eta, s, t, LR)$.
    end

    $\forall$ nodes η
      normalize the parts to get $e(\eta, s, t)$.
      if η is a single child or $e(\eta, s, s) > 0$ or $e(\eta, t, t) > 0$
        adjust the inside probabilities of its ancestors.
  end
end

Figure 3.9: The chart-parsing algorithm for PLTIG. It runs in cubic-time with respect to the sentence length.

Outside Probabilities

Like its inside probability counterpart, a PLTIG outside probability is also decomposed into five parts, one for each possible type of adjunction.

1. $f(\eta, s, t, \emptyset)$: the probability that the derivation has completed everything outside of the substring $w_{s+1}, \ldots, w_t$, and the only remaining incomplete node η has not been adjoined yet.

2. $f(\eta, s, t, L)$: the probability that the derivation has completed deriving everything outside of $w_{s+1}, \ldots, w_t$ after a left auxiliary tree adjoining into η.

3. $f(\eta, s, t, R)$: the probability that the derivation has completed deriving everything outside of $w_{s+1}, \ldots, w_t$ after a right auxiliary tree adjoining into η.

4. $f(\eta, s, t, LR)$: the probability that the derivation has completed deriving everything outside of $w_{s+1}, \ldots, w_t$ after a simultaneous adjunction at η, with the left taking precedence over the right.

5. $f(\eta, s, t, RL)$: the probability that the derivation has completed deriving everything outside of $w_{s+1}, \ldots, w_t$ after a simultaneous adjunction at η, with the right taking precedence over the left.

The overall outside probability, $f(\eta, s, t)$, is the normalized sum of its five parts:

$$
\begin{aligned}
f(\eta, s, t) \quad = \quad & p_{NL}(\eta) p_{NR}(\eta) f(\eta, s, t, \emptyset) \\
& + p_{NR}(\eta) f(\eta, s, t, L) \\
& + p_{NL}(\eta) f(\eta, s, t, R) \\
& + f(\eta, s, t, LR) \\
& + f(\eta, s, t, RL).
\end{aligned}
$$

As in a PCFG, the inside and outside probabilities still complement each other to represents the likelihood of deriving the sentence with η subsuming the substring between s and t. With the complication of adjunction, however, the types of components have to match as well. For the overall likelihood of η deriving $w_{s+1}, \ldots, w_t$, in the parse, we multiply the normalized-sum of one with the no-adjunction component of another, ensuring the at-most-one restriction at η:

$$f(\eta, s, t, \emptyset)e(\eta, s, t) = f(\eta, s, t)e(\eta, s, t, \emptyset).$$

The likelihood of η deriving the substring in the parse with exactly one left adjunction can also be computed in two ways:

$$f(\eta, s, t, \emptyset)e(\eta, s, t, L)p_{NR}(\eta) = f(\eta, s, t, L)e(\eta, s, t, \emptyset)p_{NR}(\eta).$$

The likelihood of η deriving the substring in the parse with a simultaneous adjunction (with left taking precedence) can be computed by taking the following products:

$$f(\eta, s, t, \emptyset)e(\eta, s, t, LR) = f(\eta, s, t, R)e(\eta, s, t, L)p_{LR}(\eta) = f(\eta, s, t, LR)e(\eta, s, t, \emptyset).$$

Note that the composite of $f(\eta, s, t, L)$ and $e(\eta, s, t, R)$ would imply letting the right adjunction take precedence.

The outside probabilities are computed recursively in a top-down manner. The base cases are taken to be the set of start tree probabilities:

$$f(\eta, 0, T, \emptyset) = \begin{cases} p_I(\rho) & : \ Root(\rho) = \eta \\ 0 & : \ \text{otherwise} \end{cases}$$

For the general case, we compute each component of an outside probability in terms of those outside probability components already computed.

For the no-adjunction component, $f(\eta, s, t, \emptyset)$, η may belong to one of six non-overlapping configurations (see Figure 3.10). In the simplest case, η is an only child, so it inherits the

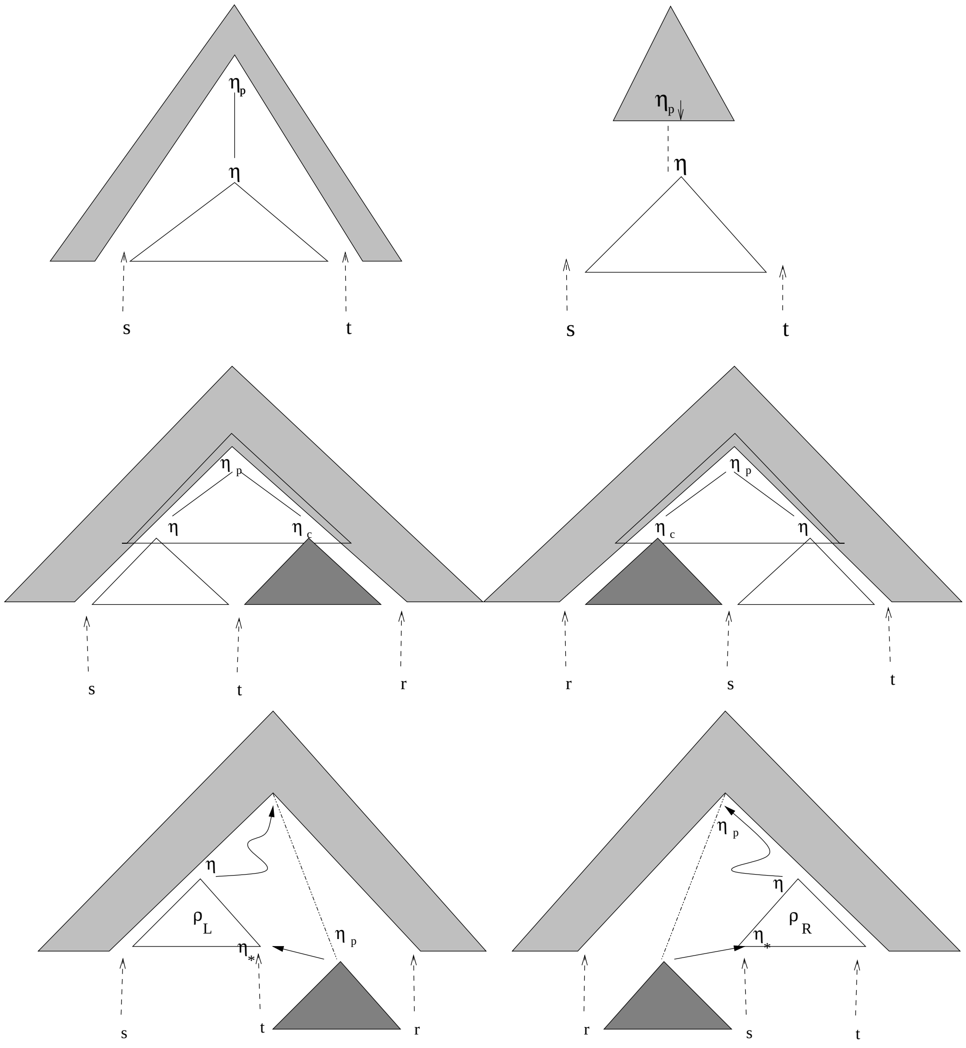


Figure 3.10: Six possible tree configurations to consider for propagating the outside probability on node η without adjoining into it: (1) η is an only child; (2) η is the root of an initial tree that might substitute into a frontier node (η_p); (3)η is a left child; (4) η is a right child; (5) η is the root of a left auxiliary tree; and (6) η is the root of a right auxiliary tree. A dark grey triangle indicates that its inside probabilities have been calculated. The light grey polygon indicates its outer probabilities have been calculated.

outside probability of its parent, η_p,

$$f(\eta, s, t, \emptyset) = f(\eta_p, s, t).$$

The substitution case is similar to the single child case. Suppose η is the root of an initial tree that might substitute into a frontier node η_p, then

$$f(\eta, s, t, \emptyset) = p_S(\eta_p, Tree(\eta)) f(\eta_p, s, t).$$

If η were one of two children of η_p, it would need to "share" the parent's outside probability with its sibling node η_c. It receives the portion not covered by the inside probability of its sibling. As the left child, η gets

$$f(\eta, s, t, \emptyset) = \sum_{r=t}^{|\mathbf{w}|} f(\eta_p, s, r) e(\eta_c, t, r),$$

whereas as the left child, η gets

$$f(\eta, s, t, \emptyset) = \sum_{r=0}^{s} f(\eta_0, r, t) e(\eta_c, r, s).$$

Finally, η may be the root node of an auxiliary tree. While nothing may adjoin into η, the tree it belongs to is free to adjoin into other nodes as long as the *at most one* restriction is observed. For example, suppose η is the root node of a left auxiliary tree, ρ_L; and suppose η_p is an interior node that derives a substring between positions t and r. ρ_L *may not* adjoin into η_p if η_p derived the substring with the aid of another left adjunction. For the adjunction to be possible, η_p must have derived the substring in one of two ways: without adjunction

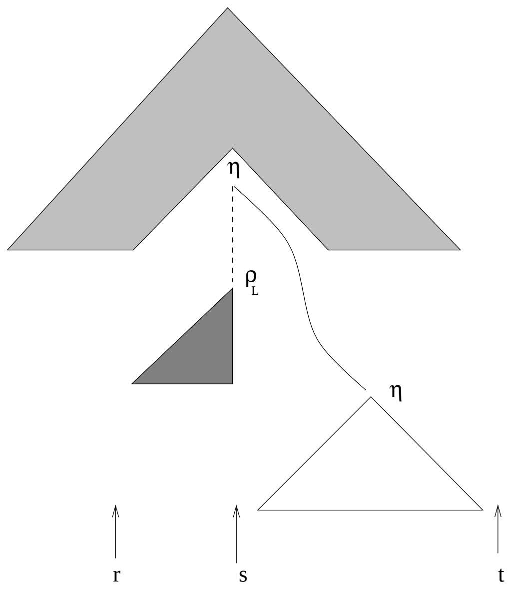


Figure 3.11: One term (i.e., for a fixed r) in the computation of $f(\eta, s, t, L)$.

or with one right adjunction.

The computing of the outside probability for η is, in a way, similar to the sibling case discussed above. From η's perspective, an adjunction site η_p can be thought as both the right sibling (pre-adjunction) and the parent (post-adjunction) of η. However, there are two differences. We need to consider all possible nodes that ρ_L might adjoin into, and we need to limit the nodes to those that have not been left adjoined already. Therefore, we get

$$f(\eta, s, t, \emptyset) = \sum_{\eta_p} \sum_{r=t}^{|\mathbf{w}|} p_L(\eta_p, \rho_L) f(\eta_p, s, r, \emptyset) \times \left(\begin{array}{r} e(\eta_p, t, r, \emptyset) p_{NR}(\eta_p) \\ +e(\eta_p, t, r, R) p_{RL}(\eta_p) \end{array} \right)$$

Similarly, if η were the root node of a right auxiliary tree, ρ_R, we would get

$$f(\eta, s, t, \emptyset) = \sum_{\eta_p} \sum_{r=0}^{s} p_R(\eta_p, \rho_R) f(\eta_p, r, t, \emptyset) \times \left(\begin{array}{r} e(\eta_p, r, s, \emptyset) p_{NL}(\eta_p) \\ +e(\eta_p, r, s, L) p_{LR}(\eta_p) \end{array} \right).$$

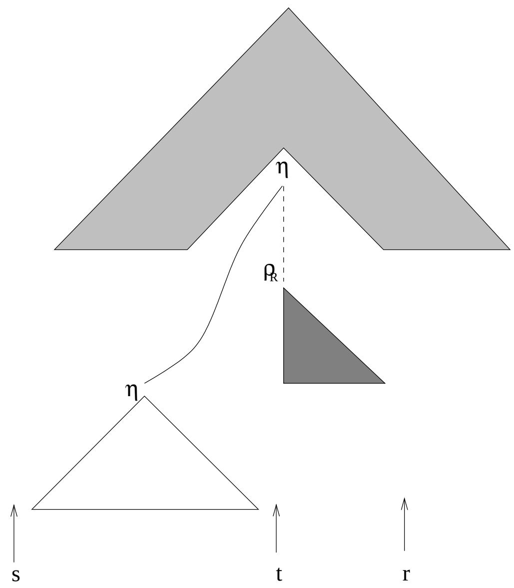


Figure 3.12: One term (i.e., for a fixed r) in the computation of $f(\eta, s, t, R)$.

Next, for the left-adjunction component, $f(\eta, s, t, L)$, we consider η

$$f(\eta, s, t, L) = \sum_{\rho_L} \sum_{r=0}^{s} p_L(\eta, \rho_L) e(Root(\rho_L), r, s) f(\eta, r, t, \emptyset).$$

The calculation of the right-adjunction component, $f(\eta, s, t, R)$, mirrors that of the left-adjunction (see Figure 3.12). It is

$$f(\eta, s, t, R) = \sum_{\rho_R} \sum_{r=t}^{|\mathbf{w}|} f(\eta, s, r, \emptyset) e(Root(\rho_R), t, r) p_R(\eta, \rho_R).$$

Finally, in the case of simultaneous adjunction, both a left auxiliary tree ρ_L and a right auxiliary tree ρ_R are adjoined into node η such that ρ_L covers a partial string from position r_1 to s and ρ_R covers a partial string from position t to r_2, where $0 \leq r_1 < s$ and $t < r_2 \leq T$. As with computing the inside probabilities, while dividing the substring into three segments is conceptually clearer, it is less efficient. Therefore, the equations are rewritten to divide

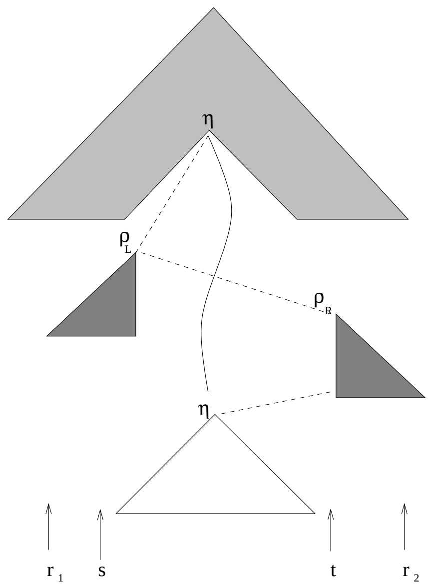


Figure 3.13: One term (i.e., for fixed r_1 and r_2) in the computation of the outside probability of η after both a left and a right auxiliary tree adjoined at η, with ρ_R having precedence over ρ_L.

the substring into two pieces.

$$
\begin{aligned}
f(\eta, s, t, LR) &= p_{LR}(\eta) \sum_{\rho_L} \sum_{\rho_R} \sum_{r_1=0}^{s} \sum_{r_2=t}^{|\mathbf{w}|} \left(\begin{array}{l} \quad p_R(\eta, \rho_R) p_L(\eta, \rho_L) \\ \times f(\eta, r_1, r_2, \emptyset) \\ \times e(Root(\rho_L), r_1, s) e(Root(\rho_R), t, r_2) \end{array} \right) \\
&= p_{LR}(\eta) \sum_{\rho_L} \sum_{r_1=0}^{s+1} p_L(\eta, \rho_L) e(Root(\rho_L), r_1, s) f(\eta, r_1, t, R) \\
f(\eta, s, t, RL) &= p_{RL}(\eta) \sum_{\rho_L} \sum_{\rho_R} \sum_{r_1=0}^{s} \sum_{r_2=t}^{|\mathbf{w}|} \left(\begin{array}{l} \quad p_R(\eta, \rho_R) p_L(\eta, \rho_L) \\ \times f(\eta, r_1, r_2, \emptyset) \\ \times e(Root(\rho_L), r_1, s) e(Root(\rho_R), t, r_2) \end{array} \right) \\
&= p_{RL}(\eta) \sum_{\rho_R} \sum_{r_2=t}^{|\mathbf{w}|} p_R(\eta, \rho_R) e(Root(\rho_R), t, r_2) f(\eta, s, r_2, L) \\
\frac{f(\eta, s, t, LR)}{p_{LR}(\eta)} &= \frac{f(\eta, s, t, RL)}{p_{RL}(\eta)}
\end{aligned}
$$

3.3.3 M-Step

In this section, we show how the general purpose re-estimation rule expressed in Equation (3.5) can be applied to update the different kinds of PLTIG parameters described in Section 2.3. We have previously shown that Equation (3.5) can be expressed in terms of the PCFG inside and outside probabilities with Equation (3.9) and Equation (3.10). We now discuss modifications needed to accommodate updating PLTIG parameters.

Let us begin with updating the parameters of the start tree distribution. Mapping the generic notation to this distribution, we write $p_{ij} = p_I(\rho_j)$, the probability of starting the derivation with the root node of the j^{th} initial tree[6]. The update equation for $p_I(\rho_j)$ is simpler than the general purpose version. Because the action of selecting the start tree is only associated with the entire sentence, we do not need to sum over all substrings. Thus, Equation (3.9) is simplified to

$$C_{Init}(\rho_j) = \sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w})c_{\rho_j}(\mathbf{v}) = p_I(\rho_j)e(Root(\rho_j),0,T).$$

Substituting back into Equation (3.5), we get

$$\hat{p_I}(\rho_j) = \frac{C_{Init}(\rho_j)/Pr_{\theta_0}(\mathbf{w})}{\sum_j C_{Init}(\rho_j)/Pr_{\theta_0}(\mathbf{w})}.$$

For PLTIGs in the canonical tree representation that we discussed in Section 3.3.1, there is only one initial tree, ρ_{Init}, so $p_I(\rho_{Init})$ is always 1.

Although the PLTIGs expressed in our canonical tree representation does not make use of the substitution, we define its update equation here for completeness. Each substitution

[6]There is only one start tree distribution, so the index i is unimportant.

parameter, $p_{ij} = p_S(\eta_i, \rho_j)$, represents the probability of substituting the j^{th} initial tree into a substitution node η_i. Because the substitution operation is the same as expanding a PCFG nonterminal, the update equation for $p_S(\rho_j)$ is a straight-forward application of Equation (3.9) to Equation (3.5), resulting in

$$\hat{p}_S(\eta_i, \rho_j) = \frac{p_S(\eta_i, \rho_j) \sum\limits_{s=0}^{|\mathbf{w}|-1} \sum\limits_{t=s+1}^{|\mathbf{w}|} f(\eta_i, s, t) e(Root(\rho_j), s, t) / Pr_{\theta_0}(\mathbf{w})}{\sum\limits_j p_S(\eta_i, \rho_j) \sum\limits_{s=0}^{|\mathbf{w}|-1} \sum\limits_{t=s+1}^{|\mathbf{w}|} f(\eta_i, s, t) e(Root(\rho_j), s, t) / Pr_{\theta_0}(\mathbf{w})}$$

The equations for updating the adjunction distributions require more modification from the generic case. We discuss the changes primarily in terms of the left adjunction parameters, but the update equations for the right adjunction parameters follow very similar reasoning. As shown in Section 2.3, a left adjunction distribution is made up of a set of adjunction parameters, $\{p_L(\eta_i, \rho_j)\}$ and one no adjunction parameter, $p_{NL}(\eta_i)$, such that

$$p_{NL}(\eta_i) + \sum_{\rho_j \in LAuxTrees} p_L(\eta_i, \rho_j) = 1.$$

Mapping a left adjunction parameter into the generic notation, p_{ij}, $p_L(\eta_i, \rho_j)$ represents the probability of adjoining the j^{th} left auxiliary tree into node η_i. The action associated with the parameter elevates η_i in the derived tree to dominate over both the substring covered by ρ_j and the substring it already covered before the left adjunction. To compute the numerator expression in Equation (3.5) with inside and outside probabilities, we want to mirror the approach taken in Equation (3.10) by drawing an analogy between $Root(\rho_j)$ and X_{j_1} and between η_i and X_{j_2}. However, the mapping is not direct because PLTIGs have the *at most one adjunction* restriction. As shown in Equation (3.10), for a PCFG any derivation of a substring rooted at X_{j_1} can be paired with any derivation of another substring rooted at X_{j_2} as long as their boundaries match. For PLTIGs, however, tree ρ_j may adjoin into node η_i only if no other left auxiliary tree had already adjoined into η_i. The restriction is reflected by explicitly representing the two cases in which left adjunction into η_i is allowed. First, tree ρ_j may adjoin into η_i if no auxiliary tree adjoined into η_i. Second, tree ρ_j may adjoin into η_i if a *right auxiliary tree* has adjoined into η_i. We modify Equation (3.10) to accommodate the left adjunction parameters:

$$\begin{aligned} C_{Ladj}(\eta_i, \rho_j) &= \sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w}) c_{ij}(\mathbf{v}) \\ &= p_L(\eta_i,\rho_j) \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(\eta_i,s,t,\emptyset) \sum_{r=s+1}^{t} e(Root(\rho_j),s,r) \begin{pmatrix} p_{NR}(\eta_i)e(\eta_i,r,t,\emptyset) \\ +p_{RL}(\eta_i)e(\eta_i,r,t,R) \end{pmatrix}. \end{aligned}$$

Slightly abusing the notation, we map the generic parameter p_{ij} onto the no left adjunction parameter, $p_{NL}(\eta_i)$, to represent the probability of taking *no left adjunction* as the j^{th} possible action of the left adjunction distribution for node η_i. Applying the no left adjunction operation to η_i does not change the length of the substring under the node's control; therefore the update of $p_{NL}(\eta_i)$ is based on Equation (3.9). The difference is that we do not want all instances of η_i deriving any substring, only those in which η_i took no left adjunction. Therefore, the derivations can be one of two kinds: η_i deriving a substring with no adjunction, or η_i deriving it after exactly one right auxiliary tree adjoined into it. Similar to the modification on Equation (3.10) for left adjunction parameters, we modify Equation (3.9) for $p_{NL}(\eta_i)$:

$$\begin{aligned} C_{NL}(\eta_i) &= \sum_{\mathbf{v}\in\mathcal{V}} Pr_{\theta_0}(\mathbf{v},\mathbf{w}) c_{(i,NL)}(\mathbf{v}) \\ & p_{NL}(\eta_i) \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(\eta_i,s,t,\emptyset) \times \begin{pmatrix} p_{NR}(\eta_i)e(\eta_i,s,t,\emptyset) \\ +p_{RL}(\eta_i)e(\eta_i,s,t,R) \end{pmatrix}. \end{aligned}$$

One last caveat is concerned with the normalization factor in the denominator expression in Equation (3.5). The complete distribution for left adjunction includes not just the parameters relating left auxiliary trees to the node but also the no left adjunction parameter. That is,

$$\begin{aligned}
& \sum_j \hat{p}_L(\eta_i, \rho_j) + \hat{p}_{NL}(\eta_i) = \frac{\sum_j (C_{Ladj}(\eta_i, \rho_j) + C_{NL}(\eta_i))}{\lambda_i Pr_{\theta_0}(\mathbf{w})} = 1 \\
\Rightarrow\quad & \lambda_i = \sum_j (C_{Ladj}(\eta_i, \rho_j) + C_{NL}(\eta_i)) / Pr_{\theta_0}(\mathbf{w})
\end{aligned}$$

Substituting $C_{Ladj}(\eta_i, \rho_j)$, $C_{NL}(\eta_i)$, and the modified λ_i back to Equation (3.5), we can update the parameters of the left adjunction distribution with these equations:

$$\begin{aligned}
\hat{p}_L(\eta_i, \rho_j) &= \frac{C_{Ladj}(\eta_i, \rho_j)/Pr_{\theta_0}(\mathbf{w})}{(\sum_j C_{Ladj}(\eta_i, \rho_j) + C_{NL}(\eta_i))/Pr_{\theta_0}(\mathbf{w})} \\
\hat{p}_{NL}(\eta_i) &= \frac{C_{NL}(\eta_i)/Pr_{\theta_0}(\mathbf{w})}{(\sum_j C_{Ladj}(\eta_i, \rho_j) + C_{NL}(\eta_i))/Pr_{\theta_0}(\mathbf{w})}
\end{aligned}$$

The update equations for the right adjunction parameters mirror those for left adjunctions. Here, we list them without further explanations.

$$\begin{aligned}
C_{Radj}(\eta_i, \rho_j) &= p_R(\eta_i, \rho_j) \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(\eta_i, s, t, \emptyset) \sum_{r=s+1}^{t} \begin{pmatrix} p_{NL}(\eta_i) e(\eta_i, s, r, \emptyset) \\ +p_{LR}(\eta_i) e(\eta_i, s, r, L) \end{pmatrix} \times e(Root(\rho_j), r, t) \\
C_{NR}(\eta_i) &= p_{NR}(\eta_i) \sum_{s=0}^{|\mathbf{w}|-1} \sum_{t=s+1}^{|\mathbf{w}|} f(\eta_i, s, t, \emptyset) \times \begin{pmatrix} p_{NL}(\eta_i) e(\eta_i, s, t, \emptyset) \\ +p_{LR}(\eta_i) e(\eta_i, s, t, L) \end{pmatrix} \\
\hat{p}_R(\eta_i, \rho_j) &= \frac{C_{Radj}(\eta_i, \rho_j)/Pr_{\theta_0}(\mathbf{w})}{(\sum_j C_{Radj}(\eta_i, \rho_j) + C_{NR}(\eta_i))/Pr_{\theta_0}(\mathbf{w})} \\
\hat{p}_{NR}(\eta_i) &= \frac{C_{NR}(\eta_i)/Pr_{\theta_0}(\mathbf{w})}{(\sum_j C_{Radj}(\eta_i, \rho_j) + C_{NR}(\eta_i))/Pr_{\theta_0}(\mathbf{w})}
\end{aligned}$$

Smoothing

As we have mentioned in our discussion of N-grams, lexical-based formalisms are sensitive to the data sparsity problem because the value for each parameter can only be reliably determined if it is based on enough observations in the training data. But even a large training set cannot guarantee occurrences of all possible word relations. For example, suppose the words "cute" and "boat" never co-occurred within the same sentence in the training data. According to the PLTIG update rules described above, we would assign a zero probability to the parameter of adjoining the tree with anchor "cute" into a node in the tree with anchor "boat." However, there is a non-zero chance that "cute" might co-occur with "boat" in a sentence in the test data. The empirical evidence for the severity of the data sparsity problem is well-known. In a study using a sample of 350 million word English text, Brown *et al.* (1992) have estimated that 14% of the trigrams in a new sample (drawn from the same source distribution) would not have occurred in the original text.

Smoothing is a technique commonly used to address the data sparsity problem (typically for N-gram models). The basic idea is to slightly increase the likelihood of the zero-probability events and slightly decrease the likelihood of other events so that the resulting distribution looks more "smooth." Many smoothing techniques have been proposed in the past, including additive smoothing (Jeffreys, 1948), Katz smoothing (Katz, 1987), and linear interpolation (Jelinek and Mercer, 1980). Based on the comparison study presented in Chen and Goodman (1996), we have decided to adapt the linear interpolation smoothing technique for PLTIGs because it performs consistently well across many different training settings.

The linear interpolation technique defines each smoothed N-grams parameter ($\bar{p}$) as a recursive function:

$$
\begin{aligned}
\bar{p}(w_i \mid w_{i-N+1} \ldots w_{i-1}) &= \kappa_{w_{i-N+1} \ldots w_{i-1}} \hat{p}(w_i \mid w_{i-N+1} \ldots w_{i-1}) \\
&\quad + (1 - \kappa_{w_{i-N+1} \ldots w_{i-1}}) \bar{p}(w_i \mid w_{i-N+2} \ldots w_{i-1}) \\
&\vdots \\
\bar{p}(w_i) &= \kappa_0 \hat{p}(w_i) + \frac{(1 - \kappa_0)}{|W|}.
\end{aligned}
$$

The function is expressed as a weighted sum of two components: the probability of observing the N-gram in the training data ($\hat{p}$) and the smoothed lower-order $N-1$-gram estimate($\bar{p}$). For the base case of smoothing unigrams, one might use the uniform distribution over the vocabulary size ($|W|$) as the smoothed estimates of the lower order distribution. This technique introduces a new set of parameters, each is denoted as κ and takes a value between 0 and 1 to specify the weighting factor. The above equations specify a distinct κ for each lower-order word string, but that results in too many additional parameters for practical purposes. As a more tractable alternative, it has been suggested that those κ's representing words with the same frequencies should be tied to the same values: if $Count(w_a \ldots w_b) = Count(w_c \ldots w_d)$ then $\kappa_{w_a \ldots w_b} = \kappa_{w_c \ldots w_d}$ (Bahl *et al.*, 1983).

In the case of the PLTIGs, we smooth the estimate of an adjunction parameter (say between node η_i and a left auxiliary tree ρ_j) by interpolating the expected likelihood of ρ_j adjoining into η_i (as observed in the training data) and a lower-order estimate (denoted p_A) based solely on the auxiliary tree ρ_j (i.e., the expected likelihood that ρ_j might adjoin into any node). Moreover, p_A itself is a smoothed estimate, interpolated with a uniform distribution over the number of possible adjunction events at node η_i, which is the number of left auxiliary trees plus the no-adjunction option. The smoothed left adjunction parameter

$\bar{p}_L(\eta_i, \rho_j)$ is expressed by the following equations[7]:

$$\bar{p}_L(\eta_i, \rho_j) = \kappa_{\eta_i} \hat{p}_L(\eta_i, \rho_j) + (1 - \kappa_{\eta_i}) \bar{p}_A(\rho_j),$$
$$\bar{p}_A(\rho_j) = \kappa_0 \hat{p}_A(\rho_j) + \frac{(1 - \kappa_0)}{|LAux| + 1}.$$

The smoothing parameters (i.e., the κ's) in the equations above are associated with the nodes of the elementary trees. However, because inducing a PLTIG is a computationally expensive process, we cannot afford to maintain this many smoothing parameters. To reduce the number of parameters to specify, we bundle the κ's associated with similar nodes (in terms of their positions in their respective elementary tree) to the same value. We empirically determine the values for these parameters, using a disjoint data set from the main training corpus. In our experiments, we set aside a section of the training data for estimating these smoothing parameters. This method of training smoothing parameters is called *held-out interpolation*.

3.4 Empirical evaluations of PLTIG

In the following experiments we show that PLTIGs of varying sizes and configurations can be induced by processing a large training corpus, and that the trained PLTIGs can provide parses on unseen test data of comparable quality to the parses produced by PCFGs. Moreover, we show that PLTIGs have significantly lower entropy values than PCFGs, suggesting that they make better language models. Two corpora of very different nature are used for training and testing. The first set of experiments uses the Air Travel Information System (ATIS) corpus. Section 3.4.1 presents the complete results of this set of experiments. To

[7]The equations for the right adjunction parameters are analogous to these.

determine if PLTIGs can scale up well, we have also begun another study that uses a larger and more complex corpus, the Wall Street Journal TreeBank corpus. The initial results are discussed in Section 3.4.1. To reduce the effect of the data sparsity problem, we back off from words to using the part of speech tags as the anchoring lexical items in all the experiments. Moreover, we use the held-out interpolation technique to smooth the parameters for both the N-gram models and PLTIGs. Because the PCFGs are not lexicalized, they do not require smoothing in these experiments.

3.4.1 Experiments

The ATIS Corpus

To reproduce the results of PCFGs reported by Pereira and Schabes, we use the ATIS corpus for our first experiment. This corpus contains 577 sentences with 32 part-of-speech tags. To ensure statistical significance, we generate ten random train-test splits on the corpus. Each set randomly partitions the corpus into three sections according to the following distribution: 80% training, 10% held-out, and 10% testing. This gives us, on average, 406 training sentences, 83 testing sentences, and 88 sentences for held-out testing. The results reported here are the averages of ten runs.

We have trained three types of PLTIGs, varying the number of left and right adjunction sites. The L2R1 version has two left adjunction sites and one right adjunction site; L1R2 has one left adjunction site and two right adjunction sites; L2R2 has two of each. The prototypical auxiliary trees for these three grammars are shown in Figure 3.4. At the end of every training iteration, the updated grammars are used to parse sentences in the held-out test sets D, and the new language modeling scores (by measuring the cross-entropy estimates $\hat{H}(D, L2R1)$, $\hat{H}(D, L1R2)$, and $\hat{H}(D, L2R2)$) are calculated. The rate of improvement of

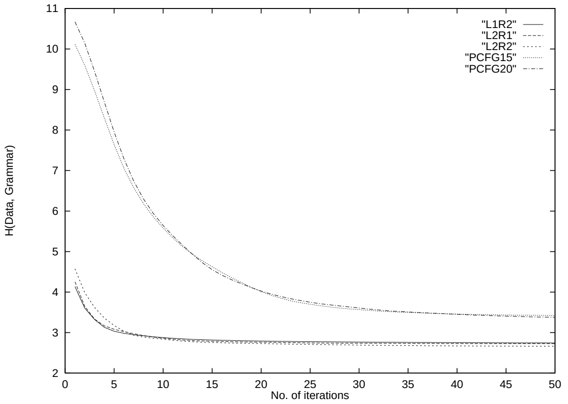


Figure 3.14: Average convergence rates of the training process for 3 PLTIGs and 2 PCFGs.

the language modeling scores determines convergence. The PLTIGs are compared with two PCFGs: one with 15-nonterminals, as Pereira and Schabes have done, and one with 20-nonterminals, which has comparable number of parameters to L2R2, the larger PLTIG.

In Figure 3.14 we plot the average iterative improvements of the training process for each grammar. All training processes of the PLTIGs converge much faster (both in numbers of iterations and in real time) than those of the PCFGs, even when the PCFG has fewer parameters to estimate, as shown in Table 3.1. From Figure 3.14, we see that both PCFGs take many more iterations to converge and that the cross-entropy value they converge on is much higher than the PLTIGs.

During the testing phase, the trained grammars are used to produce bracketed constituents on unmarked sentences from the testing sets T. We use the *non-crossing bracketing measurement*, the percentage of brackets in the proposed parse not crossing brackets of the true parse, to evaluate the parsing quality of each grammar.[8] For evaluating the

[8]The induces grammars (PCFGs and PLTIGs) generate binary branching trees so that the number of proposed brackets in a sentence is always one fewer than the length of the sentence. The Penn Treebank corpus, on the other hand, favors a more flattened tree structure with considerably fewer brackets per sentence. The non-crossing bracketing metric does not unfairly penalize a proposed parse tree for being binary branching.

	bi-/tri-gram	PCFG 15	PCFG 20	L1R2	L2R1	L2R2
Number of parameters	1088 / 34880	3855	8640	6402	6402	8514
Iterations to convergence	–	45	45	19	17	24
Real-time convergence (min)	–	62	142	8	7	14
$\hat{H}(T, G)$	2.88 / 2.71	3.81	3.42	2.87	2.85	2.78
Crossing bracket (on T)	66.78	93.46	93.41	93.07	93.28	94.51

Table 3.1: Summary results for ATIS. The machine used to measure real-time is an HP 9000/859.

grammars' language modeling quality, we measure the *cross-entropy* estimates $\hat{H}(T, L2R1)$, $\hat{H}(T, L1R2)$, $\hat{H}(T, L2R2)$, $\hat{H}(T, PCFG_{15})$, and $\hat{H}(T, PCFG_{20})$. Cross-entropy quantifies the uncertainties of a grammar G on data T. A grammar with a *lower* entropy value indicates that there is a lower uncertainty in modeling the data, suggesting that it makes a better language model. For a baseline comparison, we consider bigram and trigram models using a simple right branching bracketing heuristic with periods attaching high. Our findings are summarized in Table 3.1.

The three types of PLTIGs generate roughly the same number of bracketed constituent errors as that of the trained PCFGs, but they achieve a significantly lower entropy score. While the average entropy value of the trigram model is the lowest, there is no statistical significance (according to the pair-wise t-test ratio) between it and any of the three PLTIGs. In any case, the slight language modeling advantage of the trigram model is offset by its inability to handle parsing.

Our ATIS results agree with the findings of Pereira and Schabes that concluded that the performances of the PCFGs do not seem to depend heavily on the number of parameters once a certain threshold is crossed. Even though $PCFG_{20}$ has about as many parameters as the larger PLTIG (L2R2), its language modeling score is still significantly worse than that of any of the PLTIGs.

	bi-/tri-gram	PCFG 15	PCFG 20	PCFG 23	L1R2	L2R1	L2R2
Number of parameters	2400 / 115296	4095	8960	13271	14210	14210	18914
Iterations to convergence	–	80	60	70	28	30	28
Real-time convergence (hr)	–	143	252	511	38	41	60
$\hat{H}(T,G)$	3.39/3.20	4.31	4.27	4.13	3.58	3.56	3.59
Crossing bracket (T)	49.44	56.41	78.82	79.30	80.08	82.43	80.832

Table 3.2: Summary results for WSJ

The WSJ Corpus

Because the sentences in ATIS are short with simple and similar structures, the difference in performance between the formalisms may not be as apparent. For the second experiment, we use the Wall Street Journal (WSJ) corpus, whose sentences are longer and have more varied and complex structures. We use sections 02 to 09 of the WSJ corpus for training, section 00 for held-out data D, and section 23 for test T. We consider sentences of length 40 or less. There are 13242 training sentences, 1780 sentences for the held-out data, and 2245 sentences in the test. The vocabulary set consists of the 47 part-of-speech tags. We compare three variants of PCFGs (15 nonterminals, 20 nonterminals, and 23 nonterminals) with three variants of PLTIGs (L1R2, L2R1, L2R2). A PCFG with 23 nonterminals is included because its size approximates that of the two smaller PLTIGs. We did not generate random train-test splits for the WSJ corpus because it is large enough to provide adequate sampling. Table 3.2 presents our findings. From Table 3.2, we see several similarities to the results from the ATIS corpus. All three variants of the PLTIG formalism have converged at a faster rate and have far better language modeling scores than any of the PCFGs. Differing from the previous experiment, the PLTIGs produce slightly better crossing bracket rates than the PCFGs on the more complex WSJ corpus. At least 20 nonterminals are needed for a PCFG to perform in league with the PLTIGs. Although the PCFGs have fewer parameters, the rate seems to be indifferent to the size of the grammars after a threshold has been reached. While upping the number of nonterminal symbols from 15 to 20 led to a 22.4% gain, the improvement from $PCFG_{20}$ to $PCFG_{23}$ is only 0.5%. Similarly for PLTIGs, L2R2 performs worse than L2R1 even though it has more parameters. The baseline comparison for this experiment results in more extreme outcomes. The right branching heuristic receives a crossing bracket rate of 49.44%, worse than even that of $PCFG_{15}$. However, the N-gram models have better cross-entropy measurements than PCFGs and PLTIGs; bigram has a score of 3.39 bits per word, and trigram has a score of 3.20 bits per word. Because the lexical relationship modeled by the PLTIGs presented in this paper is limited to those between two words, their scores are close to that of the bigram model.

Because the WSJ is a commonly used corpus for evaluating parsers (with section 23 being the test set), we can place the performance of the induced PLTIGs in the context of other learning methods. Most relevant is the work by Chiang (2000), in which a probabilistic LTAG (restricted to context-free expressivity) is extracted (rather than induced) from the WSJ corpus. When using POS tags as lexical anchors and evaluating with the non-crossing bracket metric, Chiang's system reports a parsing score of 84.4% (compared to our best score of 82.4%). That his system has access to more training information, including labels of the parse tree nodes, might account for its higher parsing accuracy. At the other end of the spectrum, grammars[9] induced from raw text using EM alone would result in a parsing

[9]We used PLTIGs with a canonical tree representation of L1R2.

score of 50.8%. Relying on just bracketing annotation, the PLTIG induction algorithm produces grammars whose performance level is much closer to those grammars extracted from the corpus than those induced without any supervision. An interesting question that arises is whether we can induce grammars using less information in the training examples without degrading the level of parsing performance. This is the subject of the next two chapters in the thesis.

3.5 Chapter Summary

In this chapter, we have presented a learning algorithm based on the Expectation-Maximization principle for inducing PLTIGs. We have derived the equations for updating the values of the parameters. Implementational issues are addressed to make the induction process a practical solution. In particular, we have discussed making the computation process more efficient through dynamic programming algorithms and overcoming the data sparsity obstacle with a smoothing technique.

Empirical experiments have been conducted to compare the relative merits of the induced grammars of several formalisms: PLTIGs, PCFGs, and N-gram networks. Our studies show that the lexicalized tree representation used by PLTIGs dramatically improves the quality of language modeling of a context-free grammar to the level of N-grams without degrading the parsing accuracy.

Chapter 4

Grammar Adaptation

4.1 Introduction

In the previous chapter, we introduced a partially unsupervised learning algorithm for inducing PLTIGs. Although one might acquire higher quality parsers by directly extracting grammars from fully annotated corpora, such methods risk becoming too tailored to the labeled training data such that they might not reliably process sentences from an arbitrary domain. Thus, while a grammar trained on the Wall Street Journal corpus can fairly accurately parse a new Wall Street Journal article, it may not perform as well on a New Yorker article. To parse sentences from a new domain, one would normally *directly induce* a new grammar from that domain, in which the training process would require hand-parsed sentences from the new domain. Because parsing a large corpus by hand is a labor-intensive task, it would be beneficial to minimize the number of additional labels needed to induce the new grammar.

In this chapter, we explore the possibility of *adapting* a grammar already trained on an old domain to the new domain. Our hypothesis is that adaptation can exploit the structural similarity between the two domains so that fewer labeled data might be needed to update the grammar to reflect the structure of the new domain. We present a quantitative study comparing direct induction and adaptation under different training conditions. It is also our goal to understand the effect of the amounts and types of labeled data on the training process for both induction strategies. For example, how much training data need to be hand-labeled? Must the parse trees for each sentence be fully specified? Are some linguistic constituents in the parse more informative than others?

To answer these questions, we have performed experiments that compare the parsing qualities of grammars induced under different training conditions using both adaptation and direct induction. We vary the amount and the linguistic classes of the annotation provided in the training data. The different types of training annotations that we have considered are presented in Section 4.2, and the difference between the two induction strategies is explained in Section 4.3. We also study the effects of corpus complexity by working with the two corpora introduced in the previous chapter (ATIS and WSJ) because they have very different structural characteristics. ATIS consists of short and relatively simple sentences and WSJ consists of long and structurally challenging sentences. The first experiment focuses on adapting a grammar induced from the WSJ corpus to process sentences from the ATIS corpus; the second experiment reverses the adaptation direction: retraining a grammar induced from the ATIS corpus for the WSJ corpus. The details of the experimental setup is described in Section 4.4 and the results are discussed in Section 4.5.

The outcome of the experiments indicates that the training examples do not need to be fully parsed for either strategy, but under the conditions of minimally labeled training data, adaptation produces better grammars than direct induction. For instance, suppose only those constituents higher up in the parse trees (e.g., complex noun phrases and sentential

Categories	**Labeled Sentence**	**ATIS**	**WSJ**
HighP	(I want (to take (the flight with at most one stop)))	17%	21%
BaseNP	(I) want to take (the flight) with (at most one stop)	27%	29%
BaseP	(I) want to take (the flight) with (at most one) stop	32%	30%
AllNP	(I) want to take ((the flight) with (at most one stop))	37%	43%
NotBaseP	(I (want (to (take (the flight (with (at most one stop)))))))	68%	70%

Table 4.1: The second column shows how the example sentence ((*I*) (*want* (*to* (*take* ((*the flight*) (*with* ((*at most one*) *stop*))))))) is labeled under each category. The third and fourth columns list the percentage break-down of brackets in each category for ATIS and WSJ respectively.

clauses) in the training data are identified by human annotators. These constituents account for only 17% of all constituents in ATIS and 21% in WSJ, but our experiments suggest that they are the most informative linguistic class of constituents. Although the resulting grammars trained on this type of data (both through adaptation and direct induction) perform somewhat worse than grammars induced from fully labeled training data, the adapted grammars show less degradation than those induced directly. Training on ATIS sentences labeled with higher-level constituent brackets, a directly induced grammar parses test sentences with 66% accuracy, whereas an adapted grammar parses with 91% accuracy, which is only 2% lower than the score of a grammar induced from fully labeled training data. Training on WSJ sentences labeled with higher-level constituent brackets, a directly induced grammar parses with 70% accuracy, whereas an adapted grammar parses with 72% accuracy, which is 6% lower than the score of a grammar induced from fully labeled training data.

4.2 Annotation Types

The training sets are annotated in multiple ways, each belongs to one of two categories. First, we construct training sets annotated with random subsets of constituents consisting

0%, 25%, 50%, 75% and 100% of the brackets in the fully annotated corpus. Second, we construct sets training in which only a certain type of constituent is annotated. We study five linguistic categories. Table 4.1 summarizes the annotation differences between the five classes and lists the percentage of brackets in each class with respect to the total number of constituents[1] for ATIS and WSJ. In an **AllNP** training set, all and only the noun phrases in the sentences are labeled. For the **BaseNP** class, we label only simple noun phrases that contain no embedded noun phrases. Similarly for a **BaseP** set, all simple phrases made up of only lexical items are labeled. Although there is a high intersection between the set of BaseP labels and the set of BaseNP labels, the two classes are not identical. A BaseNP may contain a BaseP. For the example in Table 4.1, the phrase "at most one stop" is a BaseNP that contains a quantifier BaseP "at most one." **NotBaseP** is the complement of BaseP. The majority of the constituents in a sentence belongs to this category, in which at least one of the constituent's sub-constituents is not a simple lexical item. Finally, in a **HighP** set, we label only complex phrases that decompose into sub-phrases that may be either another HighP or a BaseP. That is, a HighP constituent does not directly subsume any lexical word. A typical HighP is a sentential clause or a complex noun phrase. The example sentence in Table 4.1 contains 3 HighP constituents: a complex noun phrase made up of a BaseNP and a prepositional phrase; a sentential clause with an omitted subject NP; and the full sentence.

[1]For computing the percentage of brackets, the outer-most bracket around the entire sentence and the brackets around singleton phrases (e.g., the pronoun "I" as a BaseNP) are excluded because they do not contribute to the pruning of parses.

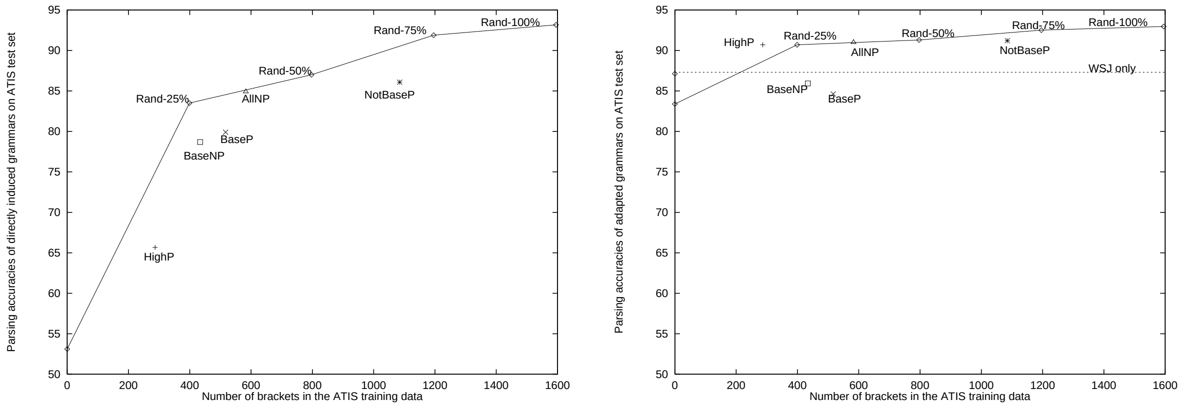
4.3 Induction Strategies

Two training strategies are considered: direct induction, in which a grammar is induced from scratch, learning from only the sparsely labeled training data; and adaptation, a two-stage learning process that first uses direct induction to train the grammar on an existing fully labeled corpus before retraining it on the new corpus. During the retraining phase, the probabilities of the grammars are re-estimated based on the new training data. The PLTIG learning algorithm presented in Chapter 3 is the basis for both the direct and adaptive induction strategies.[2] Because the learning algorithm is essentially an unsupervised training method, it is amenable to sparsely bracketed training data. We expect the adaptive method to induce better grammars than direct induction when the new corpus is only partially annotated because the adapted grammars have collected better statistics from the fully labeled data of another corpus.

4.4 Experimental Setup

We perform two experiments. The first uses ATIS as the corpus from which the different types of partially labeled training sets are generated. Both induction strategies train from these data, but the adaptive strategy pre-trains its grammars with fully labeled data drawn from the WSJ corpus. The trained grammars are scored on their parsing abilities on unseen ATIS test sets. As before, we use the non-crossing bracket measurement as the parsing metric. This experiment will show whether annotations of a particular linguistic category may be more useful for training grammars than others. It will also indicate the comparative merits of the two induction strategies trained on data annotated with these linguistic cate-

[2] The canonical grammar representation L1R2 was used for all experiments.



(a) (b)

Figure 4.1: Parsing accuracies of (a) directly induced grammars and (b) adapted grammars as a function of the number of brackets present in the training corpus. There are 1595 brackets in the training corpus all together.

gories. However, pre-training on the much more complex WSJ corpus may be too much of an advantage for the adaptive strategy. Therefore, we reverse the roles of the corpus in the second experiment. The partially labeled data are from the WSJ corpus, and the adaptive strategy is pre-trained on fully labeled ATIS data. In both cases, part-of-speech(POS) tags are used as the lexical items of the sentences. Backing off to POS tags is necessary because the tags provide a considerable intersection in the vocabulary sets of the two corpora.

4.5 Results and Discussions

4.5.1 Experiment 1: Learning ATIS

The easier learning task is to induce grammars to parse ATIS sentences. As we mentioned in chapter 3.4.1, the ATIS corpus consists of 577 short sentences with simple structures, and the vocabulary set is made up of 32 POS tags, a subset of the 47 tags used for the WSJ. Using the same experimental convention as that in chapter 3.4.1, we generate ten random train-test splits on the corpus to ensure the statistical significance of the outcome.

Before proceeding with the main discussion on training from the ATIS, we briefly describe the pre-training stage of the adaptive strategy.

Pre-training with WSJ

The idea behind the adaptive method is simply to make use of any existing labeled data. We hope that pre-training the grammars on these data might place them in a better position to learn from the new, sparsely labeled data. In the pre-training stage for this experiment, a grammar is directly induced from 3600 fully labeled WSJ sentences. Without any further training on ATIS data, this grammar achieves a parsing score of 87.3% on ATIS test sentences. The relatively high parsing score suggests that pre-training with WSJ has successfully placed the grammar in a good position to begin training with the ATIS data.

Partially Supervised Training on ATIS

We now return to the main focus of this experiment: learning from sparsely annotated ATIS training data. To verify whether some constituent classes are more informative than others, we could compare the parsing scores of the grammars trained using different constituent class labels. But this evaluation method does not take into account that the distribution of the constituent classes is not uniform. To normalize for this inequity, we compare the parsing scores to a baseline that characterizes the relationship between the performance of the trained grammar and the number of bracketed constituents in the training data. To generate the baseline, we create training data in which 0%, 25%, 50%, 75%, and 100% of the constituent brackets are randomly chosen to be included. One class of linguistic labels is better than another if its resulting parsing improvement over the baseline is higher than that of the other.

The test results of the grammars induced from these different training data are summarized in Figure 4.1. Graph (a) plots the outcome of using the direct induction strategy, and graph (b) plots the outcome of the adaptive strategy. In each graph, the baseline of random constituent brackets is shown as a solid line. Scores of grammars trained from constituent type specific data sets are plotted as labeled dots. The dotted horizontal line in graph (b) indicates the ATIS parsing score of the grammar trained on WSJ alone.

Comparing the five constituent types, we see that the HighP class is the most informative for the adaptive strategy, resulting in a grammar that scored better than the baseline. The grammars trained on the AllNP annotation performed as well as the baseline for both strategies. Grammars trained under all the other training conditions scored below the baseline. Our results suggest that while an ideal training condition would include annotations of both higher-level phrases and simple phrases, complex clauses are more informative. This interpretation explains the large gap between the parsing scores of the directly induced grammar and the adapted grammar trained on the same HighP data. The directly induced grammar performed poorly because it has never seen a labeled example of simple phrases. In contrast, the adapted grammar was already exposed to labeled WSJ simple phrases, so that it successfully adapted to the new corpus from annotated examples of higher-level phrases. On the other hand, training the adapted grammar on annotated ATIS simple phrases is not successful even though it has seen examples of WSJ higher-level phrases. This also explains why grammars trained on the conglomerate class NotBaseP performed on the same level as those trained on the AllNP class. Although the NotBaseP set contains the most brackets, most of the brackets are irrelevant to the training process, as they are neither higher-level phrases nor simple phrases.

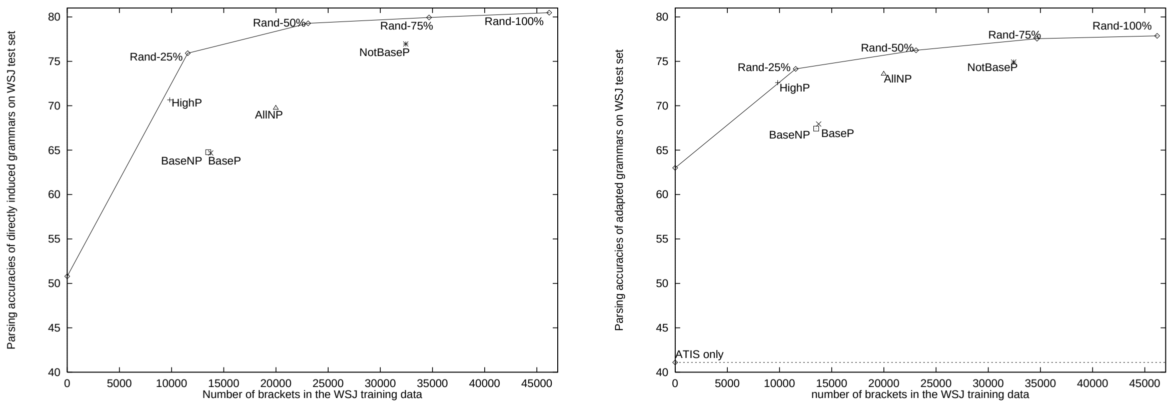
Our experiment also indicates that induction strategies exhibit different learning characteristics under partially supervised training conditions. A side by side comparison of Figure 4.1 (a) and (b) shows that the adapted grammars perform significantly better than the directly induced grammars as the level of supervision decreases. This supports our hypothesis that pre-training on a different corpus can place the grammar in a good initial search space for learning the new domain. Unfortunately, a good initial state does not obviate the need for supervised training. We see from Figure 4.1(b) that retraining with unlabeled ATIS sentences actually lowers the grammar's parsing accuracy.

4.5.2 Experiment 2: Learning WSJ

The result of the previous experiment indicates that, for inducing ATIS-style grammars, labeling complex clauses (e.g. HighP or AllNP) in the training data seems most helpful . One of the goals of this experiment is to verify whether a similar result holds for the WSJ corpus, which is structurally very different from ATIS. The WSJ corpus uses 47 POS tags, and its sentences are longer and have more embedded clauses.

In conducting this experiment, we follow the convention of the previous experiment, validating the statistical significance of the results by running ten trials for each set of training data. For each trial the training data consists of 3600 sentences, generated by moving a fix-sized window across WSJ sections 02 through 05, advancing 400 sentences at a time. For consistency with the earlier study in Chapter 3.4.1, we use the 1780 sentences in WSJ section 00 as the held-out data, and the 2245 sentences in WSJ section 23 as the test data.

In Figure 4.2, we graph the outcomes for the WSJ experiment. Some conclusions we draw from this study are similar to the ATIS experiment. Higher-level phrases still provide the most information; the grammars trained on the HighP labels are the only ones that



(a) (b)

Figure 4.2: Parsing accuracies of (a) directly induced grammars and (b) adapted grammars as a function of the number of brackets present in the training corpus.

scored as well as the baseline. Labels of simple phrases still seem the least informative; scores of those grammars trained on BaseP and BaseNP remained far below the baseline. Different from the previous experiment, however, annotating noun phrases does not seem to help this learning task as much. Although the AllNP grammars in the ATIS experiment performed about as well as the baselines, those in the WSJ experiment scored below the baselines. This observation holds for both directly induced and adapted grammars. The difference may be due to the increase in the sentence complexity of the WSJ, which further de-emphasized the role of the simple phrases. Thus, grammars trained on AllNP labels have comparable parsing scores to those trained on HighP labels. Also, we do not see as big a gap between the scores of the two induction strategies in the HighP case because the adapted grammar's advantage of having seen annotated ATIS base nouns is reduced. Nonetheless, the adapted grammars still perform 2% better than the directly induced grammars, and this improvement is statistically significant.[3]

[3] A pair-wise t-test comparing the parsing scores of the ten test sets for the two strategies shows 95% confidence in the difference.

Furthermore, grammars induced from the WSJ training data with NotBaseP labels do not fall as far below the baseline as their counterpart in the ATIS experiment, and they have higher parsing scores than those trained on HighP and AllNP. This suggests that for more complex domains, other linguistic constituents such as verb phrases[4] become more informative.

A second goal of this experiment is to test the adaptive strategy under more stringent conditions. In the previous experiment, a WSJ-style grammar was retrained for the simpler ATIS corpus. Now, we reverse the roles of the corpora to see whether the adaptive strategy still offers any advantage over direct induction.

In the adaptive method's pre-training stage, a grammar is induced from 400 fully labeled ATIS sentences. Testing this ATIS-style grammar on the WSJ test set without further training renders a parsing accuracy of 41.1%. The low score suggests that fully labeled ATIS data does not teach the grammar as much about the structure of WSJ. Nonetheless, the adaptive strategy proves to be beneficial for learning WSJ *from sparsely labeled training sets*. The adapted grammars out-perform the directly induced grammars when more than half of the brackets are missing from the training data. The most significant difference is when the training data contains no label information at all. The adapted grammar parses with 63.0% accuracy whereas the directly induced grammar parses with 50.8% accuracy. On the other hand, if the training data is mostly annotated, then the direct induction strategy finds better grammars. The adaptive strategy works poorly in this case because the parameters of the grammar have been too deeply set by the data from the simple ATIS corpus.

---

[4]We have not experimented with training sets containing only verb phrase labels (i.e., setting a pair of brackets around the head verb and its modifiers). They are a subset of the NotBaseP class.

4.6 Chapter Summary

In this chapter, we have considered adapting a grammar trained in one corpus to process sentences from a different corpus. We have shown that the structure of a grammar can be reliably learned without having fully specified constituent information in the training sentences and that the most informative constituents of a sentence are higher-level phrases, which make up only a small percentage of the total number of constituents. Moreover, we observe that grammar adaptation works particularly well with this type of sparse but informative training data. An adapted grammar consistently outperforms a directly induced grammar even for adapting from a simpler corpus to a more complex one when these high-level phrases are annotated in the training data.

That the most informative brackets are higher-level constituents and make up only one-fifth of all the labels in the corpus has two implications. First, it shows that there is potential reduction of labor for the human annotators. Although the annotator still must process an entire sentence mentally, the task of identifying higher-level structures such as sentential clauses and complex nouns should be less tedious than to fully specify the complete parse tree for each sentence. Second, one might speculate the possibilities of replacing human supervision altogether with a partial parser that locates constituent chunks within a sentence. However, as our results indicate that the most informative constituents are higher-level phrases, the parser would have to identify sentential clauses and complex noun phrases rather than low-level base noun phrases.

Chapter 5

Sample Selection

5.1 Introduction

With the emergence of the World Wide Web, large collections of text written in different languages are becoming readily available in machine readable format. This could be a valuable resource for training data for language learning tasks such as grammar induction. However, as we have discussed earlier, a completely unsupervised approach in which the training data consists of only raw text usually results in poor solutions. To induce high quality grammars, it is necessary to supplement each training sentence with its parse tree, painstakingly annotated by humans. Although our experiments from the previous chapter suggest that grammars may be induced from incompletely annotated parse trees, the qualities of the resulting grammars were not as good as those induced from fully labeled training data. In this chapter, we continue to explore for ways to induce grammars from less annotated training data but without compromising the quality of the results.

Rather than specifying the type of training annotations to provide a priori, as we have done in Chapter 4, we now focus on an interactive learning technique called *sample selection*, in which the machine takes the initiative of selecting potentially beneficial training examples for the humans to annotate. If the system could accurately identify a subset of examples with high *Training Utility Values* (TUV) out of a pool of unlabeled data, the annotators would not need to waste time on processing uninformative examples. We give a more detailed overview of this approach in section 5.2. Although our primary interest in sample selection is in its application to grammar induction, we shall first look at a simpler but related learning task, training a model of prepositional phrase attachment (discussed in Section 5.3) before returning to inducing grammars (discussed in Section 5.4) so that we may use insights gleaned from the simpler problems. The results for both tasks show that sample selection can significantly reduce the amount of annotated training examples without sacrificing the quality of the learned concepts.

5.2 Learning with Sample Selection

Unlike traditional learning systems that receive training examples indiscriminately, a sample selection learning system actively influences its progress by choosing new examples to incorporate into its training set. There are two types of selection algorithms: *committee based* or *single learner.*

A committee-based selection algorithm works with multiple learners, each maintaining a different hypothesis (perhaps pertaining to different aspects of the problem). The candidate examples that lead to the most disagreements among the different learners are considered to have the highest TUV (Cohn *et al.*, 1994; Freund *et al.*, 1997).

For computationally intensive problems, keeping multiple learners may be impractical. In this work, we focus on sample selection algorithms that use only a single learner that keeps just one working hypothesis. Without access to multiple hypotheses, the selection algorithm can nonetheless estimate the TUV of an example. We categorize some possible ranking criteria into the following three classes:

Problem-space: Knowledge about the problem-space may provide information about the type of candidates that are particularly plentiful or difficult to learn. This criterion focuses on the general attributes of the learning problem, such as the distribution of the input data and properties of the learning algorithm, but it ignores the current state of the hypothesis.

Performance of the hypothesis: Testing the candidates on the current working hypothesis shows the type of input data on which the hypothesis performs weakly. For example, if the current hypothesis is likely to be wrong or uncertain about a candidate, then it might be a good training example (Lewis and Catlett, 1994).

Parameters of the hypothesis: Estimating the potential impact of the candidates will have on the parameters of the current working hypothesis locates those examples that will change the current hypothesis the most.

Figure 5.1 outlines the single-learner sample selection training loop in pseudo-code. Initially, the training set, L, consists of a small number of labeled examples, based on which the learner proposes its first hypothesis of the target concept, C. Also available to the learner is a large pool of unlabeled training candidates, U. In each training iteration, the selection algorithm, $Select(n, U, C, f)$, ranks the candidates of U according to their expected TUVs and returns the n candidates with the highest values. The algorithm computes the expected TUV of each candidate, $u \in U$, with an evaluation function, $f(u, C)$. This function may possibly rely on the hypothesis concept C to estimate the utility of a candidate u. The

U is a set of unlabeled candidates.
L is a set of labeled training examples.
C is the current hypothesis.
Initialize:
    $C \leftarrow Train(L)$.
Repeat
    $N \leftarrow Select(n, U, C, f)$.
    $U \leftarrow U - N$.
    $L \leftarrow L \cup Label(N)$.
    $C \leftarrow Train(L)$.
Until $(C = C_{true})$ **or** $(U = \emptyset)$ **or** (human stops).

Figure 5.1: The pseudo-code for the sample selection learning algorithm

set of the n chosen candidates are then labeled by human experts and added to the existing training set. Running the learning algorithm, $Train(L)$, on the updated training set, the system proposes a new hypothesis consistent with all the examples seen thus far. The loop continues until one of three stopping conditions is met: the hypothesis is considered close enough to the target concept, all candidates are labeled, or all human resources are exhausted.

Sample selection may be beneficial for learning tasks in natural language processing. Although there exist abundant collections of raw text, the high expense of manually annotating the text sets a severe limitation for many learning algorithms. Sample selection presents an attractive solution to offset this labeled data sparsity problem. Thus far, it has been successfully applied to several classification applications. Some examples include text categorization (Lewis and Gale, 1994), part-of-speech tagging (Engelson and Dagan, 1996), and word-sense disambiguation (Fujii *et al.*, 1998).

More difficult are learning problems whose objective is not classification, but generation of complex structures. One example in this direction is applying sample selection to semantic parsing (Thompson *et al.*, 1999), in which sentences are paired with their semantic representation using a deterministic shift-reduce parser. Our main focus of this chapter is to address another complex natural language learning problem: inducing a stochastic context-free grammar that can generate syntactic parse trees for novel test sentences.

Although abstractly, parsing with a grammar can be seen as a classification task of determining the structure of a sentence by selecting one tree out of a set of possible parse trees, there are two major distinctions that differentiate it from typical classification problems. First, a classifier usually chooses from a fixed set of categories, but in our domain, every sentence has a different set of possible parse trees. Second, for most classification problems, the number of the possible categories is relatively small, whereas the number of potential parse trees for a sentence is exponential with respect to the sentence length.

We first apply sample selection to the simpler task of learning a classifier that determines whether a prepositional phrase attaches to its preceding verb or its preceding noun. Although the attachment model described in the next section is simply a binary classifier, it is also a sub-task of parsing, and may afford us insights into the main problem of our interest.

5.3 Sample Selection for Prepositional Phrase Attachment

A common source of structural ambiguities arises from sentential constructs in which there exist multiple constituents that a prepositional phrase might modify. For example, in the sentence *She wrote a book on the Titanic*, the prepositional phrase "on the Titanic" might modify either the verb, *wrote*, specifying the location of the action; or the object noun, *book*, describing a property of the object.[1] The uncertainty over which constituent the

[1] In the latter case, the parse is also semantically ambiguous. The prepositional phrase could be specifying either the location or the content of the book. Semantic ambiguity is out of the scope of this work.

prepositional phrase belongs to is called a PP-attachment ambiguity. In the above example, without any additional knowledge about the context surrounding the sentence, both parses seem equally plausible. However, ambiguities may often be resolved locally (i.e., without appealing to greater context) if one interpretation may be strongly favored over others. Consider the following pair of sentences:

> She wrote a book on the economy.
>
> She wrote a book on her sabbatical.

In the first sentence, the prepositional phrase seems more likely to modify the object noun; whereas in the second sentence, the prepositional phrase seems more likely to modify the verb.

Perhaps as a way to gain insight into the more challenging task of parsing sentences, researchers have proposed many computational models that attempt to resolve PP-attachment ambiguities. Some recent models include a rule-based model (Brill and Resnik, 1994), a backed-off model (Collins and Brooks, 1995), a variant of the backed-off model for a chain of prepositions (Merlo *et al.*, 1997), another variant of the backed-off model that also uses information from a thesaurus (Stetina and Nagao, 1997), and a maximum-entropy model (Ratnaparkhi, 1998). All these models are automatically induced from a set of training data. However, only Ratnaparkhi's model is unsupervised; all others need annotated training data, a large set of examples for which the correct PP-attachment is specified by a human. Consequently, Ratnaparkhi's model performs significantly worse than its supervised counterparts such as the Collins-Brooks model.

The goal of our work is not to improve upon the accuracy reported by the state-of-the-art PP-attachment model, but to augment its training process with sample selection so that

```
subroutine Train(L)
  foreach ex ∈ L do
    extract (v, n, p, n2, a) from ex
    foreach tuple ∈ {(v, n, p, n2), (v, p, n2), (n, p, n2), (v, n, p), (v, p), (n, p), (p, n2), (p)} do
      Count(tuple) ← Count(tuple) + 1
      if a == noun then
        Count_NP(tuple) ← Count_NP(tuple) + 1

subroutine Test(U)
  foreach u ∈ U do
    extract (v, n, p, n2) from u
    if Count(v, n, p, n2) > 0 then
      prob ← Count_NP(v,n,p,n2) / Count(v,n,p,n2)
    elsif Count(v, p, n2) + Count(n, p, n2) + Count(v, n, p) > 0 then
      prob ← (Count_NP(v,p,n2)+Count_NP(n,p,n2)+Count_NP(v,n,p)) / (Count(v,p,n2)+Count(n,p,n2)+Count(v,n,p))
    elsif Count(v, p) + Count(n, p) + Count(p, n2) > 0 then
      prob ← (Count_NP(v,p)+Count_NP(n,p)+Count_NP(p,n2)) / (Count(v,p)+Count(n,p)+Count(p,n2))
    elsif Count(p) > 0 then
      prob ← Count_NP(p) / Count(p)
    else prob ← 1
    if prob ≥ .5 then
      output noun
    else output verb
```

Figure 5.2: The Collins-Brooks PP-Attachment Classification Algorithm.

comparable performance can be achieved using fewer labeled training examples. In the following sections, we first review the Collins-Brooks model, which serves as the basic learning algorithm in the sample selection process. Next, we describe the details of several types of evaluation functions that predict the training utilities of the unlabeled examples. Finally, we present an empirical study that compares the merits of these evaluation functions. We will show that the best evaluation function can reduce the number of labeled examples by 41% without any loss of accuracy.

5.3.1 The PP-Attachment Learner

For the basic learning algorithm (i.e, the subroutine $Train()$ in Figure 5.1) we use the PP-Attachment classification algorithm proposed by Collins and Brooks (1995), which reports the best performance for PP-attachment without the aid of a thesaurus. We summarize their algorithm here because our sample selection process interacts with the basic learning algorithm.

In the Collins-Brooks framework, each training example is a quintuple of the form $(v, n, p, n2, a)$, where v, n, p, and $n2$ are the head words of the verb phrase, the object noun phrase, the preposition, and the prepositional noun phrase, respectively; and a specifies the attachment classification. For example, the phrase *"wrote a book on her sabbatical"* would be annotated as $(wrote, book, on, sabbatical, verb)$. The head words can be automatically extracted using a heuristic table-lookup in the manner described by Magerman (1994). For this learning problem, the supervision is simply the one-bit information of whether p should attach to v or to n. In order to learn to classify the attachment preferences of prepositions, the system builds attachment statistics based on the *characteristic tuples* of the training examples. A characteristic tuple is some subset of the four head words in the example, with the condition that one of the elements must be the preposition. Eight characteristic tuples can be extracted from each training example For instance, the set of characteristic tuples for $(wrote, book, on, sabbatical, verb)$ is: $\{(wrote, book, on, sabbatical)$, $(wrote, on, sabbatical)$, $(book, on, sabbatical)$, $(wrote, book, on)$, $(wrote, on)$, $(book, on)$, $(on, sabbatical)$, $(on)\}$. The attachment statistics are a collection of the occurrence frequencies for all the characteristic tuples in the training set and the occurrence frequencies for the characteristic tuples of those examples determined to attach to nouns. For some characteristic tuple t, $Count(t)$

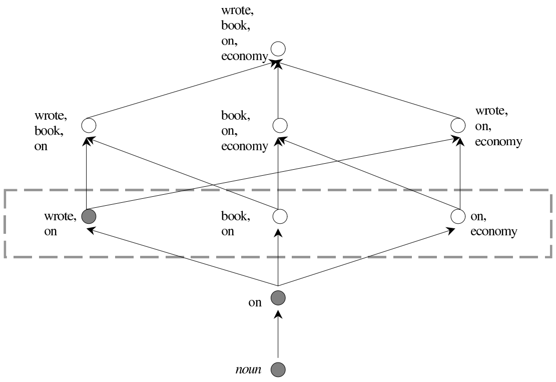


Figure 5.3: In this example, the classification of the test-case preposition is backed-off to the two-word tuple level. In the diagram, each circle represents a characteristic tuple. A filled circle denotes that the tuple has occurred in the training set. The grey rectangular box indicates the back-off level on which the classification is made.

denotes the former and $Count_{NP}(t)$ denotes the latter. In terms of the sample selection algorithm, the collection of counts represents the learner's current hypothesis (C in Figure 5.1). Figure 5.2 provides the pseudo-code for the **Train** routine.

Once trained, the system can then classify the test-cases based on the statistics of the most similar training examples and back off as necessary. For instance, to determine the PP-attachment for a test-case, the classifier would first consider the ratio of the two frequency counts for the 4-word characteristic tuple of the test-case. If the tuple never occurred in the training example, the classifier would then back off to look at its three 3-word characteristic tuples. It would continue to back off further, if necessary. In the case that the model has no information on any of the characteristic tuples of the test-case, it would, by default, classify the test-case as an instance of noun-attachment. Figure 5.3 shows using the back-off scheme on a test case. We describe the model's classification procedure for each back-off level in the **Test** pseudo-code routine in Figure 5.2.

5.3.2 Evaluation Functions

In this section, we explore some possible evaluation functions for selecting examples to train the Collins-Brooks PP-Attachment Learner. We group them according to the three classes of ranking criteria discussed in Section 5.2, and we speculate as to their potential advantages and disadvantages. After we empirically evaluate these functions individually, we will explore the possibility of combining the more successful ranking criteria to form composite evaluation functions.

The Problem Space

One advantage to exploit is our knowledge about the problem space of the learning task. Our understanding of the inner workings of the learning algorithm and properties of English prepositional phrases can guide the selection of training examples.

For instance, we know that the most problematic test cases for the PP-Attachment model are those for which it has no statistics at all. A way to compensate for this weakness is to aggressively go after data the system has not seen before. For instance, we can define an evaluation fucntion, $f_{novel}(u, C)$ that scores a candidate u by its *novelty*, the number of its characterisitc tuples that currently have zero counts.[2]

$$f_{novel}(u, C) = \sum_{t \in Tuples(u)} \begin{cases} 1 & : \quad Count(t) = 0 \\ 0 & : \quad \text{otherwise} \end{cases}$$

This simple evaluation function has some blatant defects. It may distort the data distribution so much that the system would not be able to build up a reliable collection of statistics.

[2]Note that the current hypothesis C is ignored in evaluation functions of this class because they depend only on the knowledge about the problem space.

The function does not take into account the intuition that those data that rarely occur, no matter how novel, probably have overall low training utility. Moreover, the scoring scheme does not make any distinction between the characteristic tuples of a candidate. We know, however, that the PP-attachment classifier is a back-off model that makes its decision based on statistics of the characteristic tuple with the most words first.

A slightly more sophisticated sampling of the data domain should consider not just the novelty of the data, but also the frequency of its occurrence as well as the quality of its characteristic tuples. We define a back-off model based evaluation function, $f_{backoff}(u, C)$, that scores a candidate u by counting the number of characteristic tuples obviated in all candidates if u were included in the training set. For example, suppose we have a small pool of 5 candidates, and we are about to pick the first training example:

u_1 = (put, book, on, shelf),

u_2 = (put, book, on, shelf),

u_3 = (put, idea, on, shelf),

u_4 = (wrote, book, on, shelf),

u_5 = (had, idea, on, topic)

According to $f_{backoff}$, either u_1 or u_2 would be the best choice. By selecting either as the first training example, we could ignore all but the four-word characteristic tuple for both u_1 and u_2 (a saving of 7 tuples each); since u_3 and u_4 each has three words in common with the first two candidates, they would no longer depend on their lower 4 tuples; and although we would also improve the statistics for one of u_5's tuple: *(on)*, nothing could be pruned from u_5's characteristic tuples. Thus, $f_{backoff}(u_1, C) = f_{backoff}(u_2, C) = 7 + 7 + 4 + 4 = 22$ (See Figure 5.4).

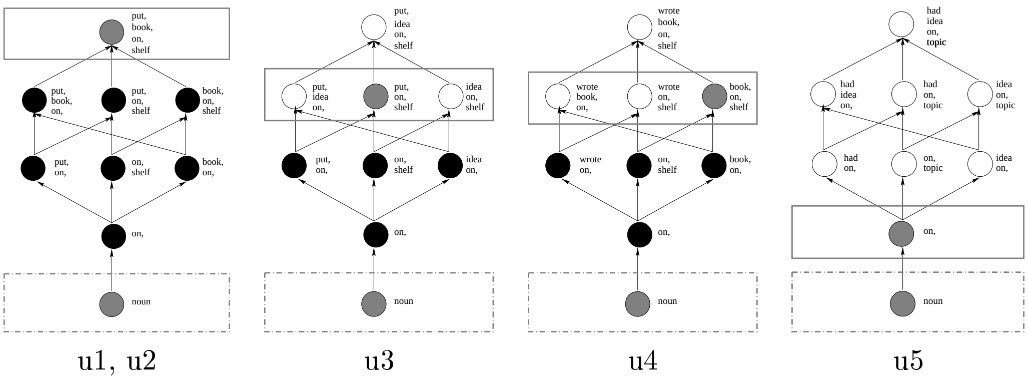


Figure 5.4: If candidate u_1 were selected, a total of 22 tuples can be ignored. The dashed rectangles show the classification level before training and the solid rectangles show the classification level after the statistics of u_1 have been taken. The ignored tuples are represented by the filled black circles.

If u_1 was chosen as the first example, u_2 loses all its utility value because we cannot prune any extra characteristic tuples by learning about u_2. That is, in the next round of selection, $f_{backoff}(u_2, C) = 0$. According to $f_{backoff}$, candidate u_5 would be the best second example because it now has the most tuples to prune (7 tuples).

The evaluation function $f_{backoff}$ improves upon f_{novel} in two ways. First, novel candidates that occur frequently are favored over those that rarely come up. As we have seen in the above example, a candidate that is similar to other candidates can eliminate more characteristic tuples all at once. Second, the evaluation strategy follows the working principle of the back-off model and discounts lower characteristic tuples that do not affect the classification process, even if they were "novel." For instance, after selecting u_1 as the first training example, we would no longer care about the two-word tuples of u_4 such as *(wrote, on)* even though we have no statistics for it.

A potential problem for $f_{backoff}$ is that after all the obvious candidates have been selected, the function would not be very good at differentiating between the remaining candidates that have about the same level of novelty and occur infrequently.

The Performance of the Hypothesis

The evaluation functions discussed in the previous section score candidates based on prior knowledge alone, independent of the current state of the learner's hypothesis and the annotation of the selected training examples. We now consider ways of using the learner's current hypothesis to select new training examples.

Because the current hypothesis represents the learner's best guess given only the data it has seen thus far, the hypothesis is highly inaccurate in the early stage. Nonetheless we can exploit its poor performance to locate potentially helpful training examples. There are two possible ranking strategies. One is to score the candidates based on their *likelihood* of being misclassified by the current hypothesis. Note that we used the word "likelihood" because we cannot know whether the hypothesis misclassified a candidate until it has been annotated by a person. From a hypothesis-centric perspective (i.e., using just the performance of the hypothesis), the likelihood is taken to be one minus the probability of the most-likely class.[3] Another is to score them based on the degrees of uncertainty that the current hypothesis has in classifying them.

For a binary classification problem, the two metrics are essentially the same.[4] Intuitively, if the hypothesis classifies a candidate as *noun* with a probability of 0.5, then it is the most uncertain about the classification and the most likely to be wrong because the classification decision is uniformly random. If the hypothesis classifies a candidate as *noun* with a probability of 1 or 0, then it is the most certain of its answer, and, from its perspective, unlikely to be wrong. Both performance-based ranking strategies can be expressed

---

[3]For example, suppose u belongs to one of three classes and under the hypothesis, C, $P(Class_1 \mid u, C) = 0.2$, $P(Class_2 \mid u, C) = 0.1$, and $P(Class_3 \mid u, C) = 0.7$. Then according to C, the most likely class of u is $Class_3$ and the likelihood of a misclassification is $1 - 0.7 = 0.3$.

[4]We will see later that it is not the case for the more complex domain of grammar induction.

as a function that is symmetric at y=0.5 and monotonically decreases as the probability of *noun* moves further from 0.5. As long as it adheres to these criteria, the specific form of the function is irrelevant, since the selection is not determined by the absolute scores of the candidates, but by their scores relative to each other. We have implemented both the uncertainty evaluation function, f_{unc}, and the error-driven evaluation function, f_{err}, in terms of the likelihood of misclassification:

$$\begin{aligned} f_{unc}(u,C) &= f_{err}(u,C) \\ &= \begin{cases} 1 - P(noun \mid u,C) & : \quad P(noun \mid u,C) \geq 0.5 \\ P(noun \mid u,C) & : \quad \text{otherwise} \end{cases} \\ &= 0.5 - abs(0.5 - P(noun \mid u,C)) \end{aligned}$$

As we have alluded to earlier, a main concern about the performance-based evaluation function is that it makes an unsound assumption that the hypothesis is at least somewhat reliable. Selecting training examples based solely on the performance of a poor hypothesis is prone to pitfalls. On one hand, the hypothesis may be overly confident about the certainty of its decisions. For example, the hypothesis may assign *noun* to a candidate with a probability of 1 based on a single previous observation in which a similar example was labeled as *noun*. Despite the unreliable statistics, this candidate would not be selected since the hypothesis considers this a known case. Conversely, the hypothesis may also direct the selection algorithm to chase after hopeless causes. As an example, let us consider PP's with *in* as the head preposition. These PP's occur frequently and about half of those PP's should attach to the object noun. Even though training on more labeled *in* examples does not improve the model's performance on future *in*-PP's, the selection algorithm will keep on requesting more *in* training examples because the hypothesis remains uncertain about this preposition.[5] Boot-strapped onto a good initial hypothesis, this type of evaluation

[5]This phenomenon is particularly acute in the early stages of refining the hypothesis because most

functions may be effective, but with an unlucky starting point, it may waste much effort on uninformative candidates.

The Parameters of the Hypothesis

The potential problems with the performance-based evaluation function all stem from the fact that the metric completely trusts the hypothesis' subjective diagnosis of its own progress. Although we want the selection of examples to depend on the current state of the hypothesis, we need to be able to step back and ask how good is the current hypothesis and what type of examples will improve it the most? In this section, we define an evaluation function that scores candidates based on their utilities in increasing the confidence about the parameters of the hypothesis (i.e., the collection of statistics over the characteristic tuples of the training examples).

Training the parameters of the PP-attachment model is not unlike empirically determining the bias of a coin. We measure the coin's bias by repeatedly tossing it and keep track of the percentage of times it landed on heads. The more trials we perform, the more confident we grow about our empirical estimation of the bias. Similarly, in estimating the likelihood of a PP to attach to its object noun, we are more confident about the classification decision based on statistics with higher counts. A quantitative measurement of our confidence in a statistic is the *confidence interval*. It is a region around our measured statistic, bounding the area where the true statistic might lie. More specifically, suppose we have measured a

---

decisions are based on statistics of the head preposition alone; in the later stages, the hypothesis can usually rely on higher-ordered characteristic tuple that tend to be better classifiers.

statistic (e.g., expected value) $\bar{p}$ based on n trials, then the confidence interval is defined as:

$$conf\_int(\bar{p}, n) = \frac{1}{1 + \frac{t^2}{n}} \left(\bar{p} + \frac{t^2}{2n} \pm t \sqrt{\frac{\bar{p}(1-\bar{p})}{n} + \frac{t^2}{4n^2}} \right),$$

where t is a *threshold value* that depends on the degree of precision we desire. For instance, if we want to be 90% confident that the true statistic p lies within the interval, then $t = 1.6576$ (Lindgren, 1959). Applying the confidence interval concept to evaluating candidates for the back-off PP-attachment model, we define a function f_{conf} that scores a candidate by taking the average of the lengths of the confidence interval of each back-off level. That is,

$$f_{conf}(u, C) = \frac{\sum_{l=1}^{4} |conf\_int(\bar{p}_l(u, C), n_l(u, C))|}{4},$$

where $\bar{p}_l(u, C)$ is the probability that model C would attach u to *noun* at back-off level l, and $n_l(u, C)$ is the number of training examples upon which this classification is based.

The confidence-based evaluation function has several potential problems. One of its flaws is similar to that of f_{novel}. In the early stage, f_{conf} picks the same examples as f_{novel} since we have no confidence in the statistics of novel examples. Therefore, f_{conf} is also prone to chase after examples that rarely occur to build up confidence of some unimportant parameters. A second problem is that f_{conf} ignores the performance of the hypothesis. Thus, if candidate A has a confidence interval around $[0.6, 1]$ and candidate B has a confidence interval around $[0.4, 0.7]$, then f_{conf} would prefer candidate A. However, training on A will not change the hypothesis's performance since the entire confidence interval is already in the *noun* zone.

5.3.3 Experiments and Results

To determine the relative merits of the proposed evaluation functions, we compare the learning rate of training with sample selection according to each function against a baseline of random selection ($f_{rand}(u, C) = rand()$) in an empirical study. The corpus is a collection of 25280 verb phrases extracted from the WSJ Treebank. We randomly partition the corpus into ten train-test pairs so that each experiment is repeated for ten trials, using a different train and test partition each time. The results we report are an average across ten runs. The average number of training examples is 22487 phrases. The average number of test cases is 2793 phrases.

As shown in Figure 5.1, the learner generates an initial hypothesis based on a small set of training examples, L. These examples are randomly selected from the pool of unlabeled candidates and annotated by a human. Random sampling ensures that the initial trained set reflects the distribution of the candidate pool, and thus the initial hypothesis is unbiased. Starting with an unbiased hypothesis is important for those evaluation functions whose scoring metrics are affected by the accuracy of the hypothesis. In these experiments, L initially contains 500 randomly selected examples.

In each selection iteration, all the candidates are scored by the evaluation function, and n examples with the highest TUV's are picked out from U to be labeled and added to L. Ideally, we would like to have $n = 1$ for each iteration. In practice, however, it is often more convenient for the human annotator to label data in larger batches rather than one at a time. In these experiments, we use a batch size of $n = 500$ examples.

We make note of one caveat to this kind of n-best batch selection. Under a hypothesis-dependent evaluation function, identical examples would receive identical scores. Because identical (or very similar) examples tend to address the same deficiency in the hypothesis, adding n very similar examples to the training set is unlikely to lead to big improvement in the hypothesis. To diversify the examples in each batch, we simulate single-example selection (whenever possible) by re-estimating the scores of the candidates after each selection. Suppose we have just chosen to add candidate x to the batch. Then, before selecting the next candidate, we *estimate* the potential decrease in scores of candidates similar to x once it belongs to the annotated training set. The estimation is based entirely on the knowledge that x is chosen, but not on the classification of x. Thus, only certain types of evaluation functions are amenable to the re-estimation process. For example, if scores were assigned by f_{conf}, then we know that the confidence intervals of the candidates similar to x must decrease slightly after learning x. On the other hand, if scores were assigned by f_{unc}, then we could not perceive any changes to the scores of similar candidates without knowing the true classification of x.

Results and Discussions

This section presents the empirical measurements of the model's performances using training examples selected with the various evaluation functions. We compare each proposed function with the baseline of random selection (f_{rand}). The results are graphically depicted from two perspectives. One view (see Figure 5.5(a)) focuses on the relationship between the model's PP-attachment accuracy on the test data (x-axis) and the number of training examples it received (y-axis). Typically, to improve the model's performance on the test set, more training examples are required. Each curve in the graph represents the learning rate of the model under a different evaluation function. In Figure 5.5(a), the learning rates of f_{novel} and f_{rand} are illustrated. The lines have the same end points because all models have

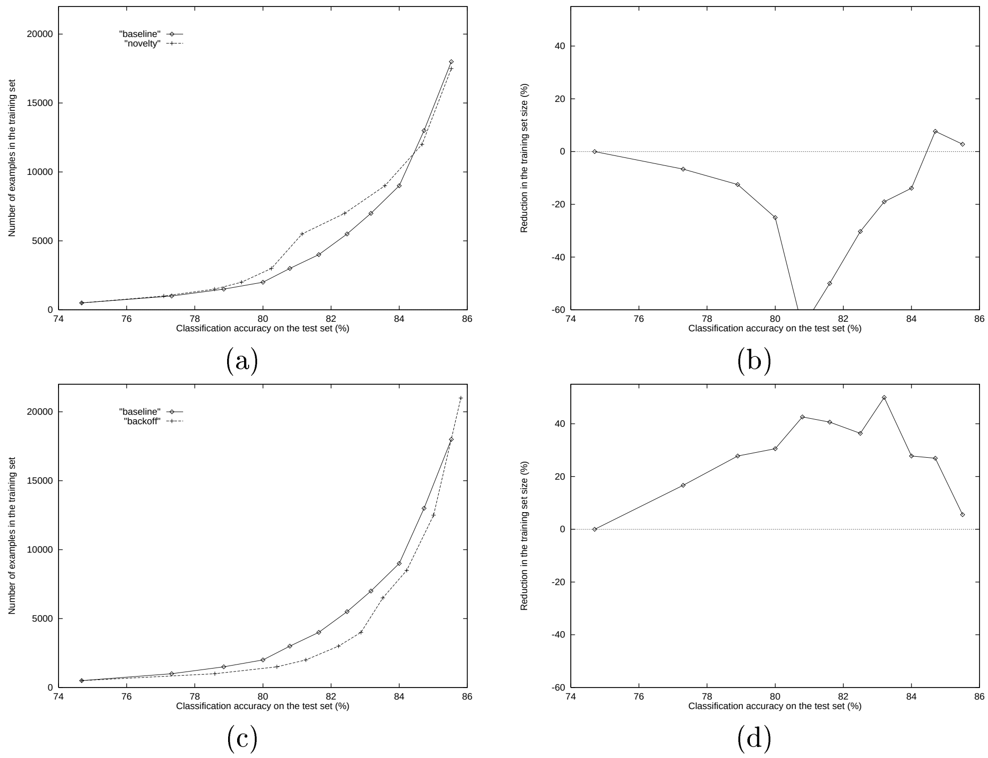


Figure 5.5: The performance of the evaluation functions that rank data based on knowledge about the problem space. (a)-(b) compares f_{novel} to the baseline, and (c)-(d) compares $f_{backoff}$ to the baseline.

the same initial hypothesis (based on the same randomly selected 500 training examples) and the same final hypothesis (based on all training examples). We deem one evaluation function to be better than another if its learning rate is sharper than others (i.e., its curve is under the others). The plot shows that for the most part, f_{novel} has a shallower learning rate curve than the baseline.

The second view (see Figure 5.5(b)) depicts the percentage of reduction in the size of the training set that the proposed evaluation function achieves over the baseline (y-axis) for different levels of proficiency (x-axis). For example, Figure 5.5(b) shows that in order to train a model that attaches PP's with 85.5% accuracy, sample selection with f_{novel} offers a modest 2.7% reduction in the size of the training set. From these two views, we see that

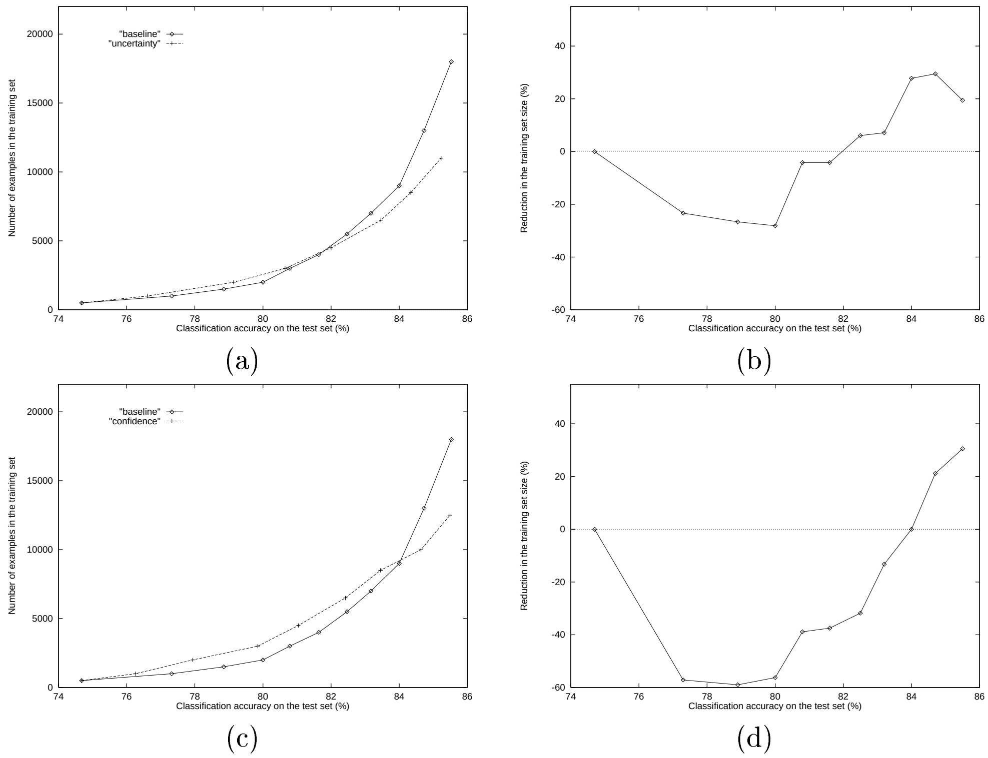


Figure 5.6: (a)-(b) compares the performance-based evaluation function f_{unc} with the baseline. (c)-(d) compares the confidence-based evaluation function, f_{conf} with the baseline.

overall, f_{novel} is not an effective evaluation function.

In contrast, $f_{backoff}$ selects more helpful training examples, especially for the early states of the hypothesis. As shown by Figure 5.5(c)-(d), the learning rate of the model under $f_{backoff}$ is always at least as fast as the baseline. Despite the early advantage, however, the reduction in the number of training examples is negligible for learning a model with high accuracy. This outcome validates our predictions. Scoring candidates by a combination of their novelty, occurrence frequencies, and the qualities of their characteristic tuples, $f_{backoff}$ selects helpful early (the first 4000 or so) training examples. Then, just as in f_{novel}, the learning rate remained stagnant for the next 2000 poorly selected examples. Finally, when the remaining candidates all have similar novelty values and contain mostly characteristic tuples that occur infrequently, the selection became random.

Figure 5.6 graphs the results of the two evaluation functions that score candidates based on the current state of the hypothesis. The learning rate of a model using the performance-based function f_{unc} is shown in (a)-(b), and that of confidence-based function f_{conf} is shown in (c)-(d). Although both functions suffer a slow start, they are much more effective at reducing the training set when learning high quality models. Of the two hypothesis-dependent functions, the learning rate of f_{unc} follows more closely to the baseline in the beginning. Because all the unknown statistics are initialized to 0.5, selection based on the uncertainty of the hypothesis is essentially random sampling. Only after the hypothesis became sufficiently accurate (after training on about 5000 annotated examples) did f_{unc} begin to make informed selections. Following a similar but more exaggerated pattern, the confidence-based function, f_{conf}, consistently under-performed for a longer period before finally overtaking the baseline. As we have noted earlier, because the hypothesis is not confident about novel candidates, f_{conf} and f_{novel} tend to select the same early examples. Therefore, the early reduction rate of f_{conf} is as bad as that of f_{novel}. In the later stage, while f_{novel} continues to flounder, f_{conf} can select better candidates based on a more reliable hypothesis. Comparing these basic evaluation functions, we conclude that the involvement of the hypothesis in the selection process is a key factor in reducing the size of training set without sacrificing the model's attachment accuracy.

Hybrid of Ranking Criteria

While both the uncertainty-based function and the confidence-based function performed well individually, they select examples based on different and possibly complementary principles. A natural extension is to explore the possibility of combining both ranking criteria in one function. The simplest option is to take the product of the two basic evaluation functions. This way, a candidate receives a high score only if the hypothesis' decision is uncertain *and* if we have low confidence in the parameters used to make the decision.

$$f_{u\times c}(u, C) = f_{unc}(u, C) \times f_{conf}(u, C)$$

The main potential short-coming of this composite function is that while taking the product of the basic functions can find good candidates initially, it is not good at differentiating the "mediocre" candidates that make up the bulk of the pool population in the later stages of training. For example, suppose we have to rank the following four candidates:

	f_{unc}	f_{conf}	$f_{u\times c}$
Candidate A	0.5	0.1	0.05
Candidate B	0.1	0.5	0.05
Candidate C	0.4	0.1	0.04
Candidate D	0.05	0.8	0.04

Intuitively, a preferred ranking order might be D-A-B-C. As shown in Figure 5.7, although the classification of Candidate D has a high certainty, the wide confidence interval suggests that it is still very unreliable. Because the confidence interval around the classification of Candidate A is narrow, even though the hypothesis is uncertain about the classification, learning about Candidate A would not garner as much information as learning about Candidate D. Finally, additional information about Candidate B and Candidate C would make little changes to the hypothesis because their confidence intervals lie entirely within one side of the classification decision. That is, even if we were to learn that the hypothesis had misclassified Candidate B, the parameters would not change enough to swing the hypothesis' decision to the other side. Candidate B is still slightly preferred over Candidate C due to its wider confidence interval.

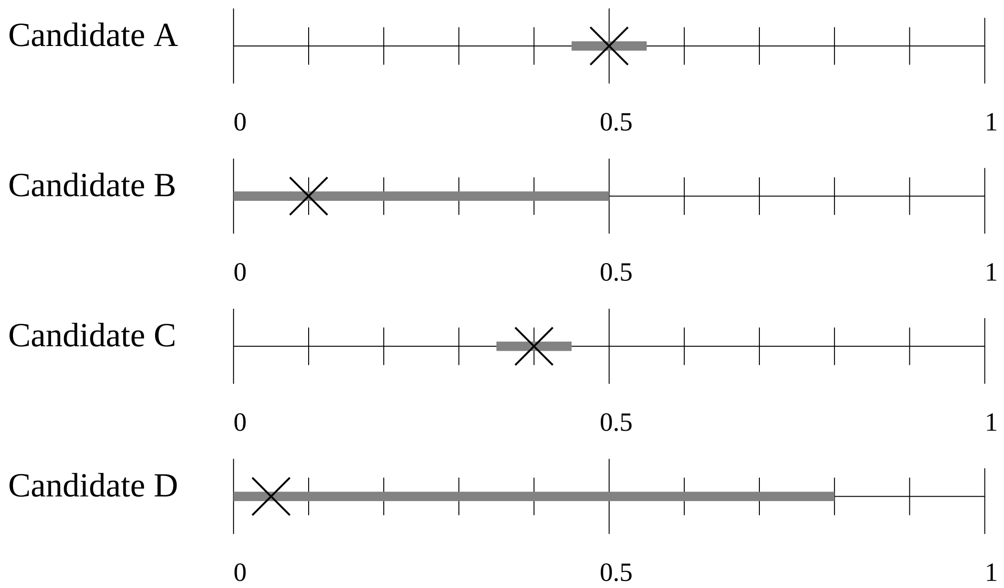


Figure 5.7: The symbol $\times$ marks the probability that the candidate would be assigned attach *noun*. The classification is the most uncertain at $Pr(noun) = 0.5$. The grey region marks the confidence interval for this classification.

The above example illustrates an interplay between the confidence and certainty factors not captured by the simple composite function. In essence, we want to weight the portion of the confidence interval that covers the region around the uncertainty boundary more heavily than the portion that covers the region around the end points. To define a continuous weighting scheme, we convolve the likelihood score with a Gaussian distribution whose mean is at the uncertainty boundary ($P(noun) = 0.5$). We choose the standard deviation value to be 0.1 so that more than 90% of the mass of the distribution is between 0.25 and 0.75. Thus, to score a candidate with its weighted confidence interval, we compute the cumulative distribution confined by the interval and the Gaussian curve. As an example, Figure 5.8 depicts scoring Candidate D with the cumulative distribution. We define an evaluation function based on this cumulative distribution concept, $f_{area}(u, C)$, as an average of the weighted confidence interval for the four back-off levels.

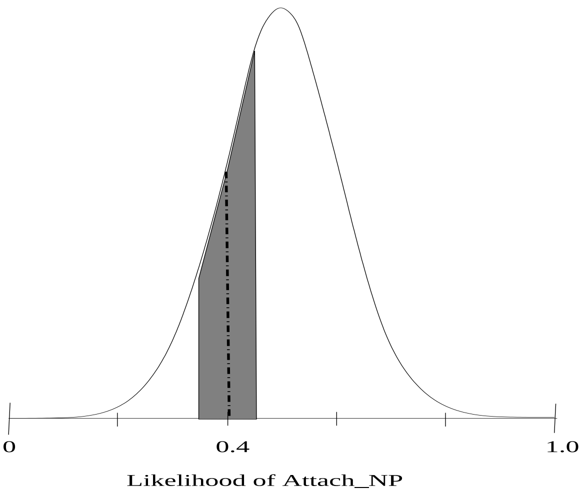


Figure 5.8: An example. The hypothesis assigns Candidate D a likelihood of 0.4 for *noun* (0.6 for *verb*), and the confidence interval for the likelihood has a width of 0.1. To score Candidate D with cumulative distribution, we compute the area bounded by the confidence interval and the Gaussian distribution curve; i.e., the shaded region in the graph.

Figure 5.9 graphs the empirical results of training with sample selection using these two composite evaluation functions. Comparing with Figure 5.6, we see that the composite product function, $f_{u \times c}$, was able to minimize the poor initial performance of f_{conf}, but in the end, it did not perform as well as either of the basic evaluation functions. In contrast, the composite function based on cumulative distribution, f_{area} not only minimized the initial performance hit but also greatly improved the final reduction rate, showing that combining the confidence and uncertainty criteria in one evaluation function is indeed beneficial.

Furthering the idea of composite evaluation functions, we now explore the possibility of incorporating the ranking criterion based on knowledge about the problem space as well. Figure 5.10(a)-(b) shows the result of a simple evaluation function $f_{b \times a}$ that takes the product of the candidate's $f_{backoff}$ score and its f_{area} score. Similar to the case of $f_{u \times c}$, although taking the product gave us an initial performance boost, the quality of the final reduction rate suffers slightly. From Figure 5.10(a), we see that the learning rate

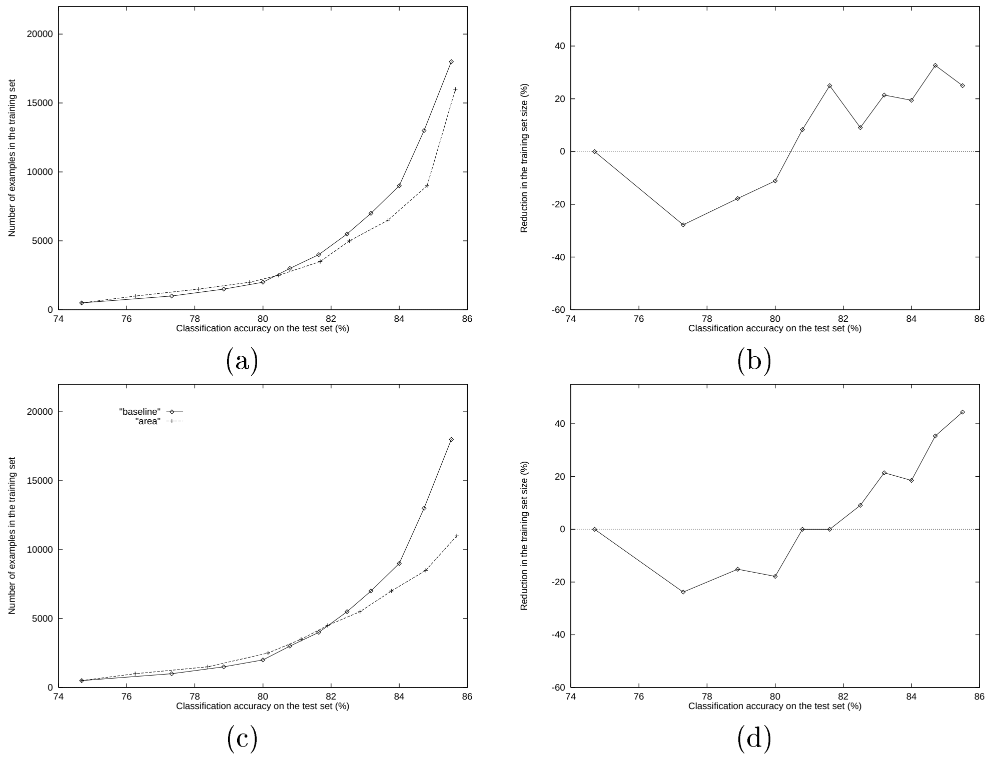


Figure 5.9: The performance of two hybrid evaluation functions that use both performance uncertainty and model confidence. (a)-(b) compares $f_{u \times c}$ to the baseline, and (c)-(d) compares f_{area} to the baseline.

severely slowed down after selecting 4000 very helpful training examples. The learning rate recovered its speed after another 2000 seemingly unhelpful training examples have been selected. Re-examining Figure 5.5(c), we recall that the basic evaluation function $f_{backoff}$ was only helpful for selecting early examples. Thus, we define a piece-meal composite, $f_{b \times a2}$, that relies on the product of $f_{backoff}$ and f_{area} initially, but only on f_{area} when $f_{backoff}$ becomes a hindrance rather than a help. Figure 5.10(c)-(d) show that this hybrid evaluation function successfully combined all three ranking criteria, and reduced the size of training examples needed for all levels of attachment accuracy.

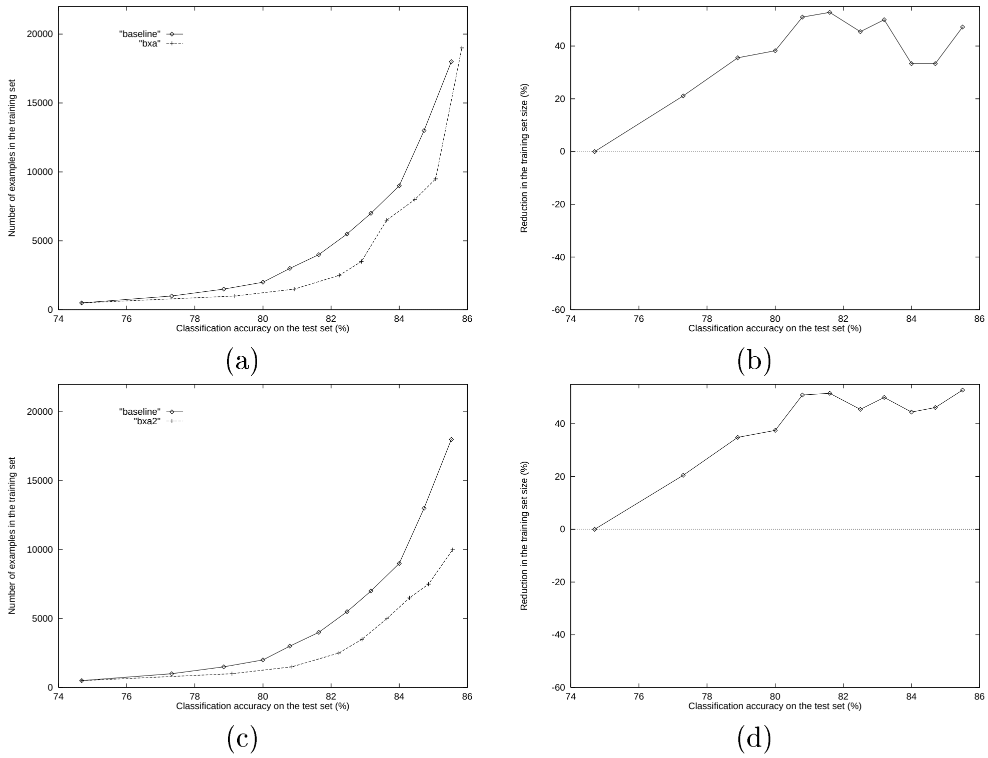


Figure 5.10: The performance of two hybrid evaluation functions that try to use all ranking criteria. (a)-(b) compares $f_{b \times a}$ to the baseline, and (c)-(d) compares $f_{b \times a2}$ to the baseline.

5.3.4 Further Discussion

Based on our experiences with applying sample selection to learning a PP-attachment model, we have made the following observations:

- Knowledge about the problem space seems to be helpful initially, but its importance dwindles in the latter stage of the training process.
- Evaluation functions need to consider the current state of the hypothesis to assign scores to candidates.
- Both uncertainty about classifying the examples and confidence about the parameters of the examples are important ranking criteria.

- It is better to *directly* model the uncertainty and confidence factors together than to consider each independently and take their composite scores at the end.

These findings have helped us gaining a better understanding of the issues we need to address for applying sample selection to learning problems in the language domain. In the next section, we shall tackle the more complex problem of applying sample selection to grammar induction.

5.4 Sample Selecting for Grammar Induction

Because grammar induction is a difficult learning problem, applying sample selection to induce a grammar for parsing is more challenging than applying sample selection to induce a model for PP-attachment. We discuss some general differences between the two here, and address the specific challenges in defining evaluation functions as they are proposed.

One main difference lies in the complexity of the tasks that the trained models perform. Whereas a trained PP-attachment model is a binary classifier that determines whether to attach a preposition to the preceding verb or noun, a trained grammar has to determine the correct parse tree out of all possible parses for a sentence. This classification task is significantly more difficult since the size of the set of all possible parses for a sentence is exponential with respect to its length. Similarly, in supervising the models' training processes, the person labeling the PP-attachment training data always reveals one unit of information (*verb* or *noun*) for each example. The amount of supervised information contained in the training data for grammar induction, however, varies from sentence to sentence, and is usually greater than one unit.[6]

[6]We consider a pair of brackets in the training sentence to be one unit of supervised information because the number of brackets broadly quantifies the amount of effort spent by the human annotator.

The non-uniformity of the supervised information in each candidate sentence gives rise to some new complications. To begin with, sample selection can now support two modes of interaction between the learner and the annotator. The learner could either request the annotator to show the full parse tree for a candidate sentence, or it could ask the annotator to disambiguate a specific part of a candidate sentence. We discuss the potential advantages and disadvantages of both modes of interaction in Section 5.4.1. Moreover, the variations in the amount of information encoded in a candidate and the amount of effort required to annotate it may both depend on its length and its structural complexity. Thus, in scoring the candidates, the evaluation functions need to balance maximizing the potential informational gain and minimizing the expected amount of effort exerted.

Another major difference is that although both models are statistical in nature, their learning algorithms are founded on different principles. The Collins-Brooks PP-attachment learning algorithm uses a history-based approach. Like N-grams, its parameters are estimated by taking concrete counts from the training data (e.g., in the training data, the tuple $(wrote, book, on, economy, noun)$ occurred 12 times). The EM-based grammar induction algorithm, on the other hand, is based on the Maximum-Likelihood-Estimation (MLE) principle. It assigns values to the grammar's parameters to maximize the likelihood of producing data like the training data. Thus, no concrete counts of training examples are taken; instead, the expected counts based on the grammar parameters' values are computed. For example, using the current parameter values, we expect a PLTIG auxiliary tree ρ to adjoin into node η 12 times; however, the training data might actually contain 14 instances of ρ adjoining into η.

This difference imposes additional restrictions on the evaluation functions. As shown in chapter 3, we compute the expected counts with a polynomial-time algorithm. In contrast, the history-based approach can take counts of the training examples in time linear to the length of the training data. Therefore, tractability is a practical concern that we cannot ignore. Although we are still willing to trade computational time for human effort, the evaluation functions that we propose are limited to those with reasonable processing time.

5.4.1 Selecting Sentences vs. Selecting Substrings

Because the parse trees for most sentences are made up of multiple units of annotation (i.e., multiple pairs of parentheses), there are two possible modes of learner-annotator interactions. In the first mode, the learner estimates the overall TUV of each candidate sentence and queries the annotator about the sentences with the highest TUVs. The annotator would then label the complete parse tree for each of these sentences. This mode of interaction potentially wastes human effort since the learner may only need help with parsing some small part of the sentences.

In the second mode, the learner estimates the TUVs of all the substrings of each candidate sentence and queries the annotator about the substrings with the highest TUVs. Assuming that the substring is a proper linguistic unit (i.e., a constituent), the annotator would then further expand it by showing its children constituents. For example, if the learner asked the annotator about the substring: *take the flight with at most one stop*, the annotator would label it as *(take (the flight with at most one stop))*. If the substring is not a constituent, the annotator would only indicate that it is not a constituent.

In terms of computational demands, this mode of interaction requires much more time than the sentential selection mode since the number of substrings in a sentence is proportional to the square of the sentence length. The additional computational demand could be justified, however, if this mode significantly reduces the human effort more than the sentential selection mode.

In terms of reducing the amount of annotation, the substring selection mode may have some advantages over the sentential selection mode, especially when more than one person is annotating the data (Santorini, 2000). First, since only the most informative part of a sentence needs labeling, the amount of annotation may be reduced. Moreover, because the annotators expand the constituents one level at a time, the labeling task may seem less tedious. Furthermore, because each annotator is responsible for some parts of all the sentences, there may be greater consistency in the annotation of the parse trees across the corpus. On the other hand, in quantifying the amount of human effort for this mode, we need to weigh in factors other than the amount of bracket annotations. For instance, the learner may ask about multiple substrings in one sentence. Even if the total number of brackets to label were fewer, the same sentence would still need to be mentally processed by the annotators multiple times. This situation is particularly problematic when there are very few annotators as it becomes much more likely that a person would encounter the same sentence many times. Also, because most substrings are not constituents, much of the annotators' time would be devoted to determining whether the queried substring is a constituent rather than actually labeling brackets. We will discuss these issues in detail when we present the experimental results.

5.4.2 Selecting Sentences

In this section, we propose several evaluation functions, which are grouped into three categories, as we have done for the PP-attachment evaluation functions.

Problem Space

Evaluation functions that only rely on knowledge about the problem space estimate the training utility of a candidate sentence $\mathbf{w}$ without consulting the current hypothesis-grammar, G. For example, we might consider, as we have with PP-attachment, a function that ranks sentences according to their novelty to the learner and their frequencies of occurrences in the training pool. However, we do not expect such a function to be applicable in this domain because the distribution of sentences is much more uniform than the distribution of prepositional phrases. Therefore, most sentences that the learner encounters will seem novel, and have an occurrence frequency count of one.

A new possible problem-space-based evaluation function we look at is one that estimates the training utility of a sentence by its length:

$$f_{len}(\mathbf{w}, G) = length(\mathbf{w}).$$

The intuition behind this function is based on the general observation that longer sentences tend to have complex structures and introduce more opportunities for ambiguous parses.

Evaluation functions based on the problem space may seem overly simplistic, but they have one major advantage: they are easy to compute and require negligible processing time. Because inducing grammars demands much more time than inducing a PP-attachment models, it becomes more important that the evaluation functions be as efficient as possible.

The Performance of the Hypothesis

Our study on selecting examples for training a PP-attachment model suggests that a successful metric ranks candidates based on the performance of the hypothesis-model on classifying them. We now consider a similar approach for selecting examples to induce grammars for parsing. Just as a PP-attachment model can be thought of as a binary classifier, parsing with a grammar can be seen as a multi-class classification task of determining the structure of a sentence by selecting one tree out of a set of possible parse trees.[7]

Because a probabilistic grammar assigns a likelihood value to every parse, we can consider the set of probabilities of the parse trees as a distribution. A distribution is a good indicator of the grammar's uncertainty in its classification decision. A uniform distribution signifies that the grammar is at its highest uncertainty because all the parses are equally likely; whereas a distribution resembling an impulse function suggests that the grammar is very certain because it finds one parse much more likely than all others.

More formally, suppose that a grammar G can generate a candidate sentence $\mathbf{w}$ with probability $P(\mathbf{w} \mid G)$, and that the set $\mathcal{V}$ contains all possible derivations that G generated for $\mathbf{w}$. Then, we can express the probability that G can generate a single derivation, $\mathbf{v} \in \mathcal{V}$ as $P(\mathbf{v} \mid G)$, and the probabilities of all the derivations in $\mathcal{V}$ sums to the probability of $\mathbf{w}$ being in the language modeled by G. That is,

$$P(\mathbf{w} \mid G) = \sum_{\mathbf{v} \in \mathcal{V}} P(\mathbf{v} \mid G).$$

The parse chosen for $\mathbf{w}$ is the most likely derivation in $\mathcal{V}$ generated by G, denoted as $\mathbf{v}_{max}$, where

$$\mathbf{v}_{max} = ArgMax_{\mathbf{v} \in \mathcal{V}} P(\mathbf{v} \mid G).$$

[7] Technically speaking, the grammar does not classify; it merely generates the parses and their likelihoods. However, since our parser uses the maximum-likelihood strategy, in which the tree with the highest probability is chosen, the grammar essentially has control of the classification. We have not considered parsers that re-rank the parses generated by the grammars as suggested by Collins (2000).

Note that $P(\mathbf{v} \mid G)$ reflects the probability of one particular parse tree, $\mathbf{v}$, in the grammar out of all possible parse trees for all possible sentences that G accepts. To compute the likelihood of a parse being the correct parse out of the possible parses of $\mathbf{w}$ according to grammar G, denoted as $P(\mathbf{v} \mid \mathbf{w}, G)$, we need to normalize the tree probability by the sentence probability. So according to G, the likelihood that $\mathbf{v}_{max}$ is the correct parse for $\mathbf{w}$ is:

$$\begin{aligned} P(\mathbf{v}_{max} \mid \mathbf{w}, G) &= \frac{P(\mathbf{v}_{max} \mid G)}{P(\mathbf{w} \mid G)} && (5.1) \\ &= \frac{P(\mathbf{v}_{max} \mid G)}{\sum_{\mathbf{v} \in \mathcal{V}} P(\mathbf{v} \mid G)}. && (5.2) \end{aligned}$$

One possible evaluation function can be based on how likely the grammar considers its best parse to be wrong. We define such an error-driven function, $f_{error}(\mathbf{w}, G)$, where

$$f_{err}(\mathbf{w}, G) = 1 - P(\mathbf{v}_{max} \mid \mathbf{w}, G).$$

This function has an uncertainty interpretation. If the grammar were uncertain about classifying the parse of a sentence, the probability it assigns to the most likely parse would be low relative to the probability of the entire sentence. However, f_{err} does not accurately quantify the grammar's uncertainty. Because only the most likely parse is explicitly accounted for (whereas all other parses are implicitly factored in through the sentential probability), the function fails to precisely model the entire distribution of the parsing likelihoods. Let us consider a concrete example. Suppose we wish to compare a candidate that has four nearly equally likely parses (with the best parse having a probability of 0.25 relative the the probability of the sentence) and a candidate that has 100 parses in which the most likely parse has a probability of 0.2 and the rest have much smaller probabilities, in the range of 0.01. Although the distribution of the parsing likelihoods for the first sentence is more uniform, suggesting that the grammar is more uncertain about it, f_{err} would assign a higher score to the second sentence.

To explicitly model all the parses of a sentence and to quantitatively characterize its distribution, we compute the entropy of the distribution. Entropy measures the uncertainty of assigning a value to a random variable over a distribution. Informally speaking, it is the expected number of bits needed to encode the assignment. A higher entropy value signifies a higher degree of uncertainty. At the highest uncertainty, the random variable is assigned one of n values over a uniform distribution, and the outcome would require $\log_2(n)$ bits to encode.

More formally, let V be a discrete random variable that can take any possible outcome in set $\mathcal{V}$. Let $p(\mathbf{v})$ be the density function $p(\mathbf{v}) = Pr(V = \mathbf{v}), \mathbf{v} \in \mathcal{V}$. The entropy $H(V)$ is the expected negative log likelihood of random variable V:

$$H(V) = -E[\log_2(p(V))]. \tag{5.3}$$

$$= -\sum_{\mathbf{v} \in \mathcal{V}} p(\mathbf{v}) \log_2(p(\mathbf{v})). \tag{5.4}$$

Further details about the properties of entropy can be found in textbooks on information theory (Cover and Thomas, 1991).

Determining the parse tree for a sentence from a set of possible parses can be viewed as assigning a value to a random variable. Thus, a direct application of the entropy definition to the probability distribution of the parses for sentence $\mathbf{w}$ in grammar G computes its *tree entropy*, $TE(\mathbf{w}, G)$, the expected number of bits needed to encode the distribution of possible parses for $\mathbf{w}$. Note that we cannot compare sentences of different lengths by their entropy. For two sentences of unequal lengths, both with uniform distributions, the entropy of the longer one is higher. To normalize for sentence length, we define an evaluation function that computes the similarity between the actual probability distribution and the uniform distribution for a sentence of that length. For a sentence $\mathbf{w}$ of length l, there can be at most $O(2^l)$ equally likely parse trees and its maximal entropy is $O(l)$ bits (Cover and Thomas, 1991). Therefore, we define the evaluation function, $f_{te}(\mathbf{w}, G)$ to be the tree entropy divided by the sentence length.

$$f_{te}(\mathbf{w}, G) = \frac{TE(\mathbf{w}, G)}{length(\mathbf{w})}.$$

We now derive the expression for $TE(\mathbf{w}, G)$. Recall from Equation (5.2) that if grammar G generates a set of derivations, $\mathcal{V}$, for sentence $\mathbf{w}$, the set of probabilities $P(\mathbf{v} \mid \mathbf{w}, G)$ (for all $\mathbf{v} \in \mathcal{V}$) defines the distribution of parsing likelihoods for sentence $\mathbf{w}$:

$$\sum_{\mathbf{v} \in \mathcal{V}} P(\mathbf{v} \mid \mathbf{w}, G) = 1.$$

Note that $P(\mathbf{v} \mid \mathbf{w}, G)$ can be viewed as a density function $p(\mathbf{v})$ (i.e., the probability of assigning $\mathbf{v}$ to a random variable V). Mapping it back into the entropy definition from Equation (5.4), we derive the tree entropy of $\mathbf{w}$ as follows:

$$\begin{aligned}
TE(\mathbf{w}, G) &= H(V) \\
&= -\sum_{\mathbf{v} \in \mathcal{V}} p(\mathbf{v}) \log_2(p(\mathbf{v})) \\
&= -\sum_{\mathbf{v} \in \mathcal{V}} \frac{P(\mathbf{v} \mid G)}{P(\mathbf{w} \mid G)} \log_2(\frac{P(\mathbf{v} \mid G)}{P(\mathbf{w} \mid G)})
\end{aligned}$$

$$
\begin{aligned}
&= -\sum_{\mathbf{v}\in\mathcal{V}} \frac{P(\mathbf{v}\mid G)}{P(\mathbf{w}\mid G)} \log_2(P(\mathbf{v}\mid G)) + \sum_{\mathbf{v}\in\mathcal{V}} \frac{P(\mathbf{v}\mid G)}{P(\mathbf{w}\mid G)} \log_2(P(\mathbf{w}\mid G)) \\
&= -\frac{1}{P(\mathbf{w}\mid G)} \sum_{\mathbf{v}\in\mathcal{V}} P(\mathbf{v}\mid G)\log_2(P(\mathbf{v}\mid G)) + \frac{\log_2(P(\mathbf{w}\mid G))}{P(\mathbf{w}\mid G)} \sum_{\mathbf{v}\in\mathcal{V}} P(\mathbf{v}\mid G) \\
&= -\frac{1}{P(\mathbf{w}\mid G)} \sum_{\mathbf{v}\in\mathcal{V}} P(\mathbf{v}\mid G)\log_2(P(\mathbf{v}\mid G)) + \log_2(P(\mathbf{w}\mid G))
\end{aligned}
$$

Using the bottom-up, dynamic programming technique of computing Inside Probabilities (Lari and Young, 1990), we can efficiently compute the probability of the sentence, $P(\mathbf{w}\mid G)$. Similarly, the algorithm can be modified to compute the quantity $\sum_{\mathbf{v}\in\mathcal{V}} P(\mathbf{v}\mid G)\log_2(P(\mathbf{v}\mid G))$.

For illustrative purposes, we first describe the computation process using a PCFG expressed in Chomsky Normal Form. The basic idea is to compose the tree entropy of the entire sentence from the tree entropy of the subtrees. Just as the Inside Probability of a nonterminal X for a substring $w_i, \ldots, w_j$, $e(X,i,j)$, is defined to be the likelihood of $X \stackrel{*}{\Rightarrow} w_i \ldots w_j$, we define a new function $h(X,i,j)$ to represent the corresponding entropy for the set of subtrees.

$$
h(X,i,j) = -\sum_{\mathbf{x}\in X \stackrel{*}{\Rightarrow} w_i \ldots w_j} P(\mathbf{x}\mid G)\log_2(P(\mathbf{x}\mid G)).
$$

Therefore, $\sum_{\mathbf{v}\in\mathcal{V}} P(\mathbf{v}\mid G)\log_2 P(\mathbf{v}\mid G)$ can be expressed as $h(S,1,n)$.

We compute all possible $h(X,i,j)$ recursively. The base case is

$$
h(X,i,i) = e(X,i,i)\log_2(e(X,i,i))
$$

since a nonterminal X can generate the symbol w_i in exactly one way. For the more general case, $h(X,i,j)$, we consider all the possible rules with X on the left hand side that might have contributed to build $X \stackrel{*}{\Rightarrow} w_i \ldots w_j$.

$$h(X,i,j) = \sum_{k=i}^{j-1} \sum_{(X \to YZ)} h_{Y,Z,k}(X,i,j).$$

The function $h_{Y,Z,k}(X,i,j)$ is a portion of $h(X,i,j)$ that accounts for those derivations in which $Y \stackrel{*}{\Rightarrow} w_i \ldots w_k$ and $Z \stackrel{*}{\Rightarrow} w_{k+1} \ldots w_j$. The nonterminals Y and Z may, in turn, generate their substrings with multiple parses. Let $\mathcal{Y}$ represent the set of parses for $Y \stackrel{*}{\Rightarrow} w_i \ldots w_k$; let $\mathcal{Z}$ represent the set of parses for $Z \stackrel{*}{\Rightarrow} w_{k+1} \ldots w_j$; and let x represent the derivation step of $X \to YZ$. Then, there are a total of $\|\mathcal{Y}\| \times \|\mathcal{Z}\|$ derivations, and the probability of each derivation is $P(x)P(\mathbf{y})P(\mathbf{z})$, where $\mathbf{y} \in \mathcal{Y}$ and $\mathbf{z} \in \mathcal{Z}$. To compute $h_{Y,Z,k}$, we need to sum over all possible parses:

$$\begin{aligned}
h_{Y,Z,k}(X,i,j) &= -\sum_{\mathbf{y}\in\mathcal{Y},\mathbf{z}\in\mathcal{Z}} P(x)P(\mathbf{y})P(\mathbf{z}) \log_2(P(x)P(\mathbf{y})P(\mathbf{z})) \\
&= -\sum_{\mathbf{y}\in\mathcal{Y},\mathbf{z}\in\mathcal{Z}} P(x)P(\mathbf{y})P(\mathbf{z})(\log_2 P(x) + \log_2 P(\mathbf{y}) + \log_2 P(\mathbf{z})) \\
&= -P(x)\log_2(P(x))e(Y,i,k)e(Z,k+1,j) \\
&\quad +P(x)h(Y,i,k)e(Z,k+1,j) \\
&\quad +P(x)e(Y,i,k)h(Z,k+1,j).
\end{aligned}$$

The equations can be modified to compute the tree entropy of sentences under the PLTIG framework. Whereas in a PCFG chart parser only one rule combines two subtrees, in a PLTIG parser, there are five possible ways (no adjunction, left adjunction, right adjunction, or one of two simultaneous adjunctions) to combine two subtrees. We define the subtree entropy function, $h(\eta,i,j)$, to represent the entropy of the set of all possible composite subtrees spanning $w_i,\ldots,w_j$ with node η as the root. We also define a function g for the notational shorthand of $-x\log_2(x)$.

$$\begin{array}{rcl} l(expr) & = & -expr \times \log_2(expr) \\ h(\eta,i,j) & = & l(P_{NL}(\eta)P_{NR}(\eta))e(\eta,i,j,\emptyset) + P_{NL}(\eta)P_{NR}(\eta)h(\eta,i,j,\emptyset) \\ & & +l(P_{NR}(\eta))e(\eta,i,j,L) + P_{NR}(\eta)h(\eta,i,j,L) \\ & & +l(P_{NL}(\eta))e(\eta,i,j,R) + P_{NL}(\eta)h(\eta,i,j,R) \\ & & +l(P_{LR}(\eta))e(\eta,i,j,LR) + P_{LR}(\eta)h(\eta,i,j,LR) \\ & & +l(P_{RL}(\eta))e(\eta,i,j,RL) + P_{RL}(\eta)h(\eta,i,j,RL) \end{array}$$

$$h(\eta,i,j,\emptyset) = \begin{cases} h(\eta_1,i,j) & : \ Child(\eta) = \eta_1 \\ \sum_{k=i}^{j} \begin{pmatrix} h(\eta_1,i,k)e(\eta_2,k,j) \\ +e(\eta_1,i,k)h(\eta_2,k,j) \end{pmatrix} & : \ Children(\eta) = (\eta_1,\eta_2) \end{cases}$$

$$h(\eta,i,j,L) = \sum_{\rho_L} \sum_{k=i+1}^{j} \begin{pmatrix} l(P_L(\eta,\rho_L))e(\rho_L,i,k)e(\eta,k,j,\emptyset) \\ +P_L(\eta,\rho_L)h(\rho_L,i,k)e(\eta,k,j,\emptyset) \\ +P_L(\eta,\rho_L)e(\rho_L,i,k)h(\eta,k,j,\emptyset) \end{pmatrix}$$

$$h(\eta,i,j,R) = \sum_{\rho_R} \sum_{k=i+1}^{j} \begin{pmatrix} l(P_R(\eta,\rho_R))e(\eta,i,k,\emptyset)e(\rho_R,k,j) \\ +P_R(\eta,\rho_R)h(\eta,i,k,\emptyset)e(\rho_R,k,j) \\ +P_L(\eta,\rho_L)e(\eta,i,k,\emptyset)h(\rho_R,k,j) \end{pmatrix}$$

$$\begin{array}{l} h(\eta,i,j,LR) = h(\eta,i,j,RL) \\ = \sum_{\rho_L,\rho_R} \sum_{k_1=i+1}^{j-1} \sum_{k_2=k_1}^{j-1} \begin{pmatrix} l(P_L(\eta,\rho_L)P_R(\eta,\rho_R))e(\rho_L,i,k_1)e(\eta,k_1,k_2,\emptyset)e(\rho_R,k_2,j) \\ +P_L(\eta,\rho_L)P_R(\eta,\rho_R)h(\rho_L,i,k_1)e(\eta,k_1,k_2,\emptyset)e(\rho_R,k_2,j) \\ +P_L(\eta,\rho_L)P_R(\eta,\rho_R)e(\rho_L,i,k_1)h(\eta,k_1,k_2,\emptyset)e(\rho_R,k_2,j) \\ +P_L(\eta,\rho_L)P_R(\eta,\rho_R)e(\rho_L,i,k_1)e(\eta,k_1,k_2,\emptyset)h(\rho_R,k_2,j) \end{pmatrix} \end{array}$$

The Parameters of the Hypothesis

Finally, we attempt to define an evaluation function, $f_{conf}(\mathbf{w},G)$, that ranks candidates by estimating the confidence levels in the current values of the grammar parameters used to parse them. The function assigns a TUV to a candidate sentence by normalizing its composite confidence score, $Conf(\mathbf{w},G)$, by the length of sentence $\mathbf{w}$. The composite confidence score, $Conf(\mathbf{w},G)$, is the sum of the confidence scores of all the parses for sentence $\mathbf{w}$. The confidence score of each derivation, $\mathbf{v} \in \mathcal{V}$, in turn, is a composite of the confidence scores of individual derivation steps, $v \in \mathbf{v}$.

$$f_{conf}(\mathbf{w}, G) = \frac{Conf(\mathbf{w}, G)}{length(\mathbf{w})} = \frac{\sum_{\mathbf{v} \in \mathcal{V}} Conf(\mathbf{v})}{length(\mathbf{w})}.$$

Recall from Section 3.3 that in each derivation step v, an action α (i.e., rule) is applied to the parse, and that a likelihood value p_α (i.e., grammar parameter) is associated with each action α. Therefore, the confidence of one derivation step is essentially the confidence of one parameter. We shall first discuss computing a confidence score for each grammar parameter, then we shall extend it to computing confidence score for an entire parse.

In Section 3.3, we have shown that a parameter value p_α is computed by normalizing the expected count of using α in the training data L (denoted as $c_\alpha(L)$) by the expected count of using all applicable actions in L, $\sum_{\alpha_j} c_{\alpha_j}(L)$:[8]

$$p_\alpha = \frac{c_\alpha(L)}{\sum_{\alpha_j} c_{\alpha_j}(L)}.$$

We can also view estimating the parameter value p_α as taking samples of a binomial random variable A, in which those instances in L that use α count toward success (i.e., $A = 1$), and those that use other actions count toward failure (i.e., $A = 0$). Then, one way to quantify the confidence of parameter p_α is to compute its variance. Because we cannot compute the

[8] This is a somewhat abstracted notation of Equation (3.5).

variance directly, we approximate with its sample variance: [9]

$$Conf(p_\alpha) = SV(A) = \frac{\sum_{\alpha_j} c_{\alpha_j}(L) \times p_\alpha(1 - p_\alpha)}{\sum_{\alpha_j} c_{\alpha_j}(L) - 1},$$

The confidence score of an entire parse depends on all the confidence scores of the parameters that participated in the derivation, so we compute the variance of the product of the random variables of the individual steps:

$$Conf(\mathbf{v}) = Var(\prod_{v \in \mathbf{v}} A_v),$$

which can be calculated recursively. Here, we derive the expression for the base case, considering only a product of two terms. Let A and B be two independent binomial random variables, whose sampled means are p_A and p_B respectively. Extending from the variance of a single variable, we get:

$$\begin{aligned} Var(A) &= E[A^2] - E[A]^2 \\ Var(AB) &= E[(AB)^2] - E[AB]^2 \end{aligned}$$

Because A and B are independent, the expectation of their product is the same as the product of their expectations:

$$E[AB] = E[A]E[B].$$

[9] If a binomial random variable A was sampled n times with a sample mean of p, then its sample variance is:

$$SV(A) = \frac{np(1-p)}{n-1}.$$

So Equation (5.5) can be rewritten as:

$$\begin{aligned}Var(AB) &= E[A^2B^2] - (E[A]E[B])^2 \\ &= E[A^2]E[B^2] - E[A]^2E[B]^2.\end{aligned}$$

But from Equation (5.5), we see that

$$E[A^2] = Var(A) + E[A]^2,$$

thus we can substitute it into Equation (5.5) to get:

$$\begin{aligned}Var(AB) &= (Var(A) + E[A]^2) \times (Var(B) + E[B]^2) - E[A]^2E[B]^2 \\ &= Var(A)Var(B) + Var(A)E[B]^2 + Var(B)E[A]^2 \\ &\approx SV(A)SV(B) + SV(A)p_B^2 + SV(B)p_A^2.\end{aligned}$$

Although the confidence-based function was shown to be effective at selecting training examples for PP-attachment, it is more difficult to apply a similar technique to the grammar induction case. Because the grammar induction algorithm is not completely supervised, it uses the MLE principle to assign values to its parameters, taking expected counts (rather than concrete counts) of examples in training data. When the training examples are ambiguous, the expected counts may be very different from the true counts. Therefore, the degrees of confidence in the parameters cannot be as accurately estimated.

Another difficulty is that the complexity of the induction algorithm deters us from re-estimating the TUVs of the remaining candidates after selecting each new candidate. As we have discussed in section 5.3.3, re-estimation is important for batched annotation. Without some means of updating the TUVs after each selection, the learner would not realize that it had already selected a candidate to train some parameter with low confidence until the re-training phase that only occurs at the end of the batch selection; therefore, it may continue to select very similar candidates to train the same parameter. In the case of training the history-based PP-attachment model, the TUV updates could be easily achieved. After a new candidate has been selected, its count information can be immediately incorporated into the counts of the training examples seen thus far. Using the set of updated counts, we can then slightly increase the confidence of those remaining candidates similar to the selected one. In the case of sample selection for grammar induction, however, it is not feasible to perform the same kind of TUV updates after each candidate selection. Because the statistics taken from a training sentence heavily depend on its parse tree, the statistics taken from a newly selected candidate before it is annotated would not be appropriate. Even if we assume that the statistics can be updated, re-estimating the TUVs is a computationally expensive operation. Essentially, all the remaining candidates that share some parameters with the selected candidate would need to be re-parsed.

5.4.3 Selecting Substrings

The proposed evaluation functions for selecting substrings are classified into one of the three function categories as before. However, we have not defined a confidence-based evaluation function for this mode of interaction because it would require too much computational time.

Problem Space

Viewed from the perspective of the learning condition, there is a strong similarity between the annotated training data resulting from selecting examples on the substring level and those used in Chapter 4. In both cases, we are more interested in minimizing the amount of annotations within each training sentence rather than reducing the total number of training sentences. Somewhat abusing the notions of evaluation functions and TUVs, we might consider those sets of training data in which specific linguistic classes are labeled (as described in Section 4.2) to be the result of selectively annotating training examples based on the problem space. Later in the experimental section, we will compare grammars induced (directly) from training data in which the HighP are labeled and grammars induced by annotated data selected by other evaluation functions.

The Performance of the Hypothesis

To define an uncertainty-based function for evaluating substrings, we modify the tree entropy metric to quantify the uncertainty over the distribution of applicable rules for a substring. The proposed function, $f_{se}(s, t, \mathbf{w}, G)$, assigns as its TUV the expected number of bits needed to encode the likelihoods of all the rules of grammar G that may be applied to derive the substring $w_{s+1}, \ldots, w_t$ in sentence $\mathbf{w}$.

More concretely, let $\mathcal{V}_{(s,t)}$ represent the set of derivations of $\mathbf{w}$ in which each derivation $\mathbf{v}^{(i)}$ contains a step, $v^{(i)}_{(s,t)}$, that derives the substring with an action $act(v^{(i)}_{(s,t)}) = \alpha$. In the context of a PCFG, for example, action α might specify to apply rule $X \rightarrow YZ$ with the split point at some position r. Note that two derivations may have the same derivation step at positions s and t; that is, for two distinct derivations i and j, we may have

$$act(v^{(i)}_{(s,t)}) = act(v^{(j)}_{(s,t)}).$$

We shall denote the subset of derivations that use the same action α to derive the substring

$w_{s+1}, \ldots, w_t$ as $\mathcal{V}_{(s,t,\alpha)}$.

Let $\mathcal{A}$ denote the set of actions prescribed by these derivations. To compute the substring entropy, we need a distribution over the likelihoods of applying $\alpha \in \mathcal{A}$ to substring $w_{s+1}, \ldots, w_t$. To begin, we define the density function for each action α as the likelihood of the derivations that uses α to parse $\mathbf{w}$:

$$p(\alpha) = \frac{P(\mathcal{V}_{(s,t,\alpha)} \mid G)}{\Lambda_{(s,t)}} = \frac{\sum\limits_{\mathbf{v} \in \mathcal{V}_{(s,t,\alpha)}} P(\mathbf{v} \mid G)}{\Lambda_{(s,t)}}.$$

where $\Lambda_{(s,t)}$ is the normalization factor:

$$\begin{aligned} \Lambda_{(s,t)} &= \sum_{\alpha \in \mathcal{A}} P(\mathcal{V}_{(s,t,\alpha)} \mid G) \\ &= \sum_{\mathbf{v} \in \mathcal{V}_{(s,t)}} P(\mathbf{v} \mid G). \end{aligned}$$

To compute the substring entropy for $w_{s+1}, \ldots, w_t$, we map the distribution back into the entropy definition from Equation (5.4), similar to the derivation of TE($\mathbf{w}$, G) shown earlier:

$$\begin{aligned} f_{se}(s,t,\mathbf{w},G) &= H(V) \\ &= -\sum_{\alpha \in \mathcal{A}} p(\alpha) \log_2 p(\alpha) \\ &= -\sum_{\alpha \in \mathcal{A}} \frac{P(\mathcal{V}_{(s,t,\alpha)} \mid G)}{\Lambda_{(s,t)}} \log_2 \frac{P(\mathcal{V}_{(s,t,\alpha)} \mid G)}{\Lambda_{(s,t)}} \\ &= -\frac{1}{\Lambda_{(s,t)}} \sum_{\alpha \in \mathcal{A}} (P(\mathcal{V}_{(s,t,\alpha)} \mid G) \log_2 P(\mathcal{V}_{(s,t,\alpha)} \mid G)) + \log_2 \Lambda_{(s,t)} \end{aligned}$$

Again, we rely on the efficient calculation of the inside and outside probabilities to keep down the complexity of f_{se}. First, since the normalization factor $\Lambda_{(s,t)}$ is just the sum of the likelihoods of all derivations that pass through positions s and t, it can be more efficiently expressed as all product pairs of inside and outside probabilities for all possible actions on these positions.

$$\Lambda_{(s,t)} = \sum_{\alpha \in \mathcal{A}} e(\alpha, s, t) \times f(\alpha, s, t).$$

Similarly, the term $P(\mathcal{V}_{(s,t,\alpha)} \mid G)$ can also be efficiently expressed as a product of inside and outside probabilities. Rearranging the likelihoods of all the derivation steps according to whether they belong to the inside of the substring or the outside of the substring, it can be re-written as

$$P(\mathcal{V}_{(s,t,\alpha)} \mid G) = p_\alpha \times e(\alpha, s, t) \times f(\alpha, s, t).$$

Therefore,

$$\begin{aligned}\sum_{\alpha \in \mathcal{A}} P(\mathcal{V}_{(s,t,\alpha)} \mid G) \log_2 P(\mathcal{V}_{(s,t,\alpha)} \mid G) \quad &= \quad \sum_{\alpha \in \mathcal{A}} p_\alpha \log_2 p_\alpha e(\alpha, s, t) f(\alpha, s, t) \\ &\quad + p_\alpha e(\alpha, s, t) \log_2 e(\alpha, s, t) f(\alpha, s, t) \\ &\quad + p_\alpha e(\alpha, s, t) f(\alpha, s, t) \log_2 f(\alpha, s, t).\end{aligned}$$

As we have discussed in Section 5.4.1, selecting substrings as training examples introduces some unique problems that selecting sentences avoids. One concern is to present the annotators with a well-defined task. Once f_{se} has determined that the current grammar hypothesis is uncertain about a substring, how should the human annotator label the data to indicate the correct derivation for the substring? Because f_{se} measures the uncertainty of applying the best rule for decomposing the substring, we ask the annotator to reveal one step of the derivation by bracketing the *immediate* children substrings of the queried substring. For example, one might label the verb phrase "wrote a book on her sabbatical" as "(wrote (a book on her sabbatical))".

A potential problem with the above formulation of the substring interaction mode is that it has made an implicit assumption that may not always hold. We cannot guarantee that a selected substring is a valid constituent that can be further decomposed. What should be the annotation policy for those selected substrings that do not form linguistic constituents? We have considered two annotation policies. In one case, the annotators are first asked to identify whether the selected substring is a valid constituent. They would continuing to identify its immediate children constituents of the substring only if the queried substring itself is a valid constituent. This policy is potentially very costly (in terms of human effort) because the system may ask many non-constituent substrings throughout the course of its learning. As an alternative policy, we restrict the system to only select those substrings known to be constituents (in the trivial case, an entire sentence is always a valid constituent). In other words, we enforce the selection order to progress in a loosely top-down fashion; therefore, the system would never select an invalid constituent. The two policies are compared empirically, and the result is presented in Section 5.4.5.

5.4.4 Experimental Setup

We compare the effectiveness of sample selection using the proposed evaluation functions against a baseline of random selection ($f_{rand}(\mathbf{w}, G) = rand()$). Similar to previous experimental designs, the task is to induce grammars with sentences from the WSJ corpus and to use the trained grammars to parse test sentences.

We repeat the study for two different candidate-pool sizes. For the first experiment, we assume that there exists an abundant supply of unlabeled data. Based on empirical observations (as can be seen in Section 5.4.5), for the task we are considering, the induction algorithm typically reaches its asymptotic limit after training with 2600 sentences; therefore, it is sufficient to allow for a candidate-pool size of $U = 3500$ unlabeled WSJ sentences. In the second experiment, we restrict the size of the candidate-pool such that U contains only 900 unlabeled sentences. This experiment studies how the paucity of training data affects the evaluation functions.

For both experiments, each evaluation function is applied to the sample selection learning algorithm shown in Figure 5.1, where concept C is the current hypothesis-grammar G, and L, the set of labeled training data, initially consists of 100 sentences. For the sentence-based evaluation functions: f_{len}, f_{err}, f_{unc}, and f_{conf}, $n = 100$ new sentences are picked from U to be added to L in every iteration. For the constituent-based evaluation function, f_{const}, 1400 new brackets are revealed in every iteration. Then, a new C is induced from the updated L. After a new hypothesis-grammar is formed, it is tested. The quality of the induced grammar is judged by its ability to generate correct parses for unseen test sentences.

Following the conventions of the earlier chapters, we perform ten trials for each experiment and report the average of the results.[10] To ensure statistical significance, the paired t-test is applied for comparing two sets of results.

5.4.5 Results and Discussions

Selecting Sentences

Similar to our presentation of the PP-attachment study, the outcomes are illustrated graphically in two plots. The results of the first set of experiment, in which the evaluation functions select training examples out of a large candidate-pool, are shown in Figure 5.11. The plots on the first column compare the learning rates of the induction processes under each pro-

[10]The ten candidate-pools used in this experiment are the same as the ten sets of training sentences in the experiment in Chapter 4.5.2. WSJ section 23 is, as usual, used for testing.

posed evaluation function to that of the baseline. The learning rate relates the quality of the induced grammars to the amount of supervised training data available. In order for the induced grammar to parse test sentences with higher accuracy (x-axis), more supervision (y-axis) is needed. The plots on the second column show the relative reduction in the amount of training annotation for each function. Note that even though these functions select examples on the sentential base, the amount of supervision is measured in terms of the number of *brackets* rather than sentences because it more accurately quantifies the effort spent by the human annotator. Longer sentences tend to require more brackets than short ones, and thus take more time to analyze. We deem one evaluation function more effective than another if the smallest set of sentences it selected can train a grammar that performs at least as well as the grammar trained under the other function and if the selected data contains considerably fewer brackets than that of the other function.

Qualitatively comparing the learning rates in the graphs, we see that with the appropriate evaluation function sample selection does reduce the amount of annotation. Similar to our findings in the PP-attachment study, the simple problem-space based evaluation function, f_{len} is essentially indistinguishable from the baseline (see 5.11(a)-(b)); whereas the uncertainty-based evaluation functions can significantly reduce the amount of annotation in the training data (see 5.11(c)-(f)). Of the two that we proposed for this category, the tree entropy evaluation function, f_{te}, out-performs the error-driven evaluation function, f_{err}, because it measures parsing uncertainty more accurately. Unlike our experience with the PP-attachment study, however, the confidence-based approach learns slower than the baseline (see 5.11(g)-(h)). As discussed in Section 5.4.2, the grammar induction learning task introduces additional complexity not present in the PP-attachment case. The poor performance of f_{conf} may be caused by the somewhat inaccurate counting statistics provided by the EM-based learning algorithm and exacerbated by not having enough computational resources to update frequently.

For a more quantitative comparison, let us consider the set of grammars that achieve an average parsing accuracy of 80.3% on the test sentences. The baseline case requires an average of 33355 brackets in the training data. In contrast, to induce a grammar that reaches the same 80.3% parsing accuracy with the examples selected by f_{te}, the learner requires, on average, 21236 training brackets, reducing the amount of annotation by 36.3% comparing to the baseline. Although the learning rate of f_{err} is slower than that of f_{te} overall, it seems to have caught up in the end; it needs about the same amount of annotation (an average of 21758 training brackets for a reduction of 34.7% from the baseline) as f_{te}.

While the simplistic sentence length evaluation function, f_{len}, is less helpful, its learning rate still improves slightly faster than the baseline. A grammar of comparable quality can be induced from a set of training examples selected by f_{len} containing an average of 30288 brackets. This provides a small reduction of 9% from the baseline [11]. We consider a set of grammars to be comparable to the baseline if its mean test score is at least as high as that of the baseline and if the difference of the means is not statistically significant (using pair-wise t-test at 95% confidence).

Figure 5.12 presents the outcomes of the second experiment, in which the evaluation functions only have access to a small candidate pool. Similar to the previous experiment, grammars induced from training examples selected by f_{te} require significantly fewer annotated brackets than the baseline. Under the baseline, f_{rand}, to train grammars with 78.5% parsing accuracy on test data, an average of 11699 brackets (in 900 sentences) is required.

[11] In terms of the number of sentences, the baseline f_{rand} used 2600 randomly chosen training sentences; f_{len} selected the 1700 longest sentences as training data; and f_{te} selected 1400 sentences.

In contrast, f_{te} can induce a comparable grammar with an average of 8559 brackets (in 600 sentences), providing a saving of 26.8% in the number of training brackets. On the other hand, f_{err} seems to be less effective under this experimental condition. While its learning rate is still faster than the baseline, it is worse than that of f_{te}. To induce grammars with 78.5% parsing accuracy, the training sentences selected by f_{err} contain an average of 9639 brackets, which is 17.6% fewer than those needed by the baseline. This reduction rate is indistinguishable from that of f_{len}.

Another somewhat surprising outcome of the second study is that the grammars induced from the three methods did not parse with the same accuracy when all the sentences from the unlabeled pool have been added to the training set. Presenting the training examples in different orders changes the search path of the induction process. Trained on data selected by f_{te}, the induced grammar parses the test sentences with 79.1% accuracy, a small but statistically significant improvement over the baseline. This suggests that, when faced with a dearth of training candidates, f_{te} can make good use of the available data to induce grammars that are comparable to those directly induced from more data.

Selecting Substring

In the substring selection experiment, we compare the proposed evaluation functions to two baselines: one that selects random substrings, denoted as $f_{randsub}$,[12] and the previous baseline of selecting random sentences, f_{rand}. Let us first consider the substring entropy function (denoted as f_{se}) using the two-stage annotation policy. We found that most of the selected substrings are invalid constituents. Therefore, unless the act of determining whether a substring is a constituent requires negligible effort relative to labeling a pair

[12] The result reported here is taken from the experimental baseline for direct induction in Chapter 4.

of bracket, this evaluation function has too high an overhead to be useful. For instance, Figure 5.14 shows how the learning rate of f_{se} compares to the baseline if the effort that an annotator spent in judging a substring were to equal to one-tenth of the effort in labeling a bracket. In contrast, the top-down annotation policy, whose learning rate is shown in Figure 5.13, does not seem to have as high an overhead.

The graphs also show that as an evaluation metric, substring entropy compares favorably against other substring-level metrics, including those based on linguistic classes. However, none of the evaluation functions for the substring-selection interaction mode performs as well as any of the functions that select sentences, including the baseline. The substring-selection approach may have suffered a slower learning rate because in order to add a newly annotated substring the set of training examples, the rest of the sentence, which may not have been annotated yet, must also be included. Thus, the training examples initially contain too few brackets to effectively restrict the search in the space of possible grammars.

5.5 Chapter Summary

In this chapter, we have shown that sample selection is a powerful learning technique that can significantly reduce the amount of human effort in labeling training data. The empirical studies we have conducted on two natural language applications suggest that it is helpful for both simple learning tasks such as training a PP-attachment classifier and more complex ones such as grammar induction.

For each learning task, we propose several different types of evaluation functions for selecting the most helpful examples and compare them with a baseline of random selection. In both cases, we are able to propose an evaluation function that consistently outperforms the baseline of random selection. For learning a PP-attachment model, the most helpful evaluation function is is a hybrid model that factors in the problem space, the prediction performance of the hypothesis, and the confidence on the values of the parameters of the hypothesis. For the highest level of accuracy at 85.5%, it can reduce the training size by more than 50% (from training with 18,000 examples down to training with 8,500 examples). For inducing a grammar, our proposed evaluation function using tree entropy is shown to have selected helpful training examples. Choosing from a large pool of unlabeled candidates, it significantly reduces the amount of training annotations needed (by 36.3% in the experiment). Although the reduction is less dramatic when the pool of candidates is small (by 26.8% in the experiment), the training examples it selected helped to induce slightly better grammars.

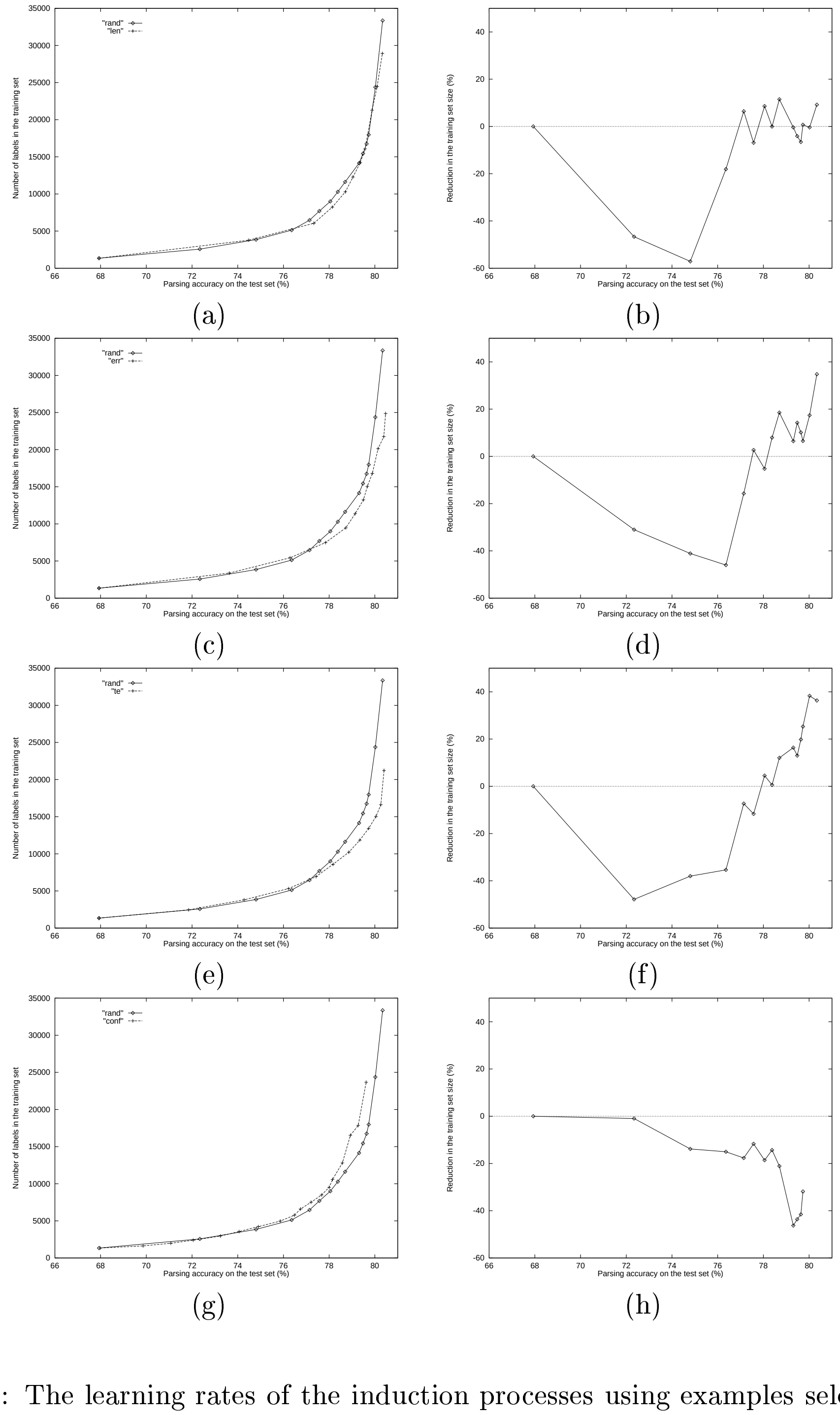


Figure 5.11: The learning rates of the induction processes using examples selected by the evaluation functions when the candidate-pool is large. We compare f_{len} to the baseline in plots (a)-(b), f_{err} in (c)-(d), f_{te} in (e)-(f), and f_{conf} in (g)-(h).

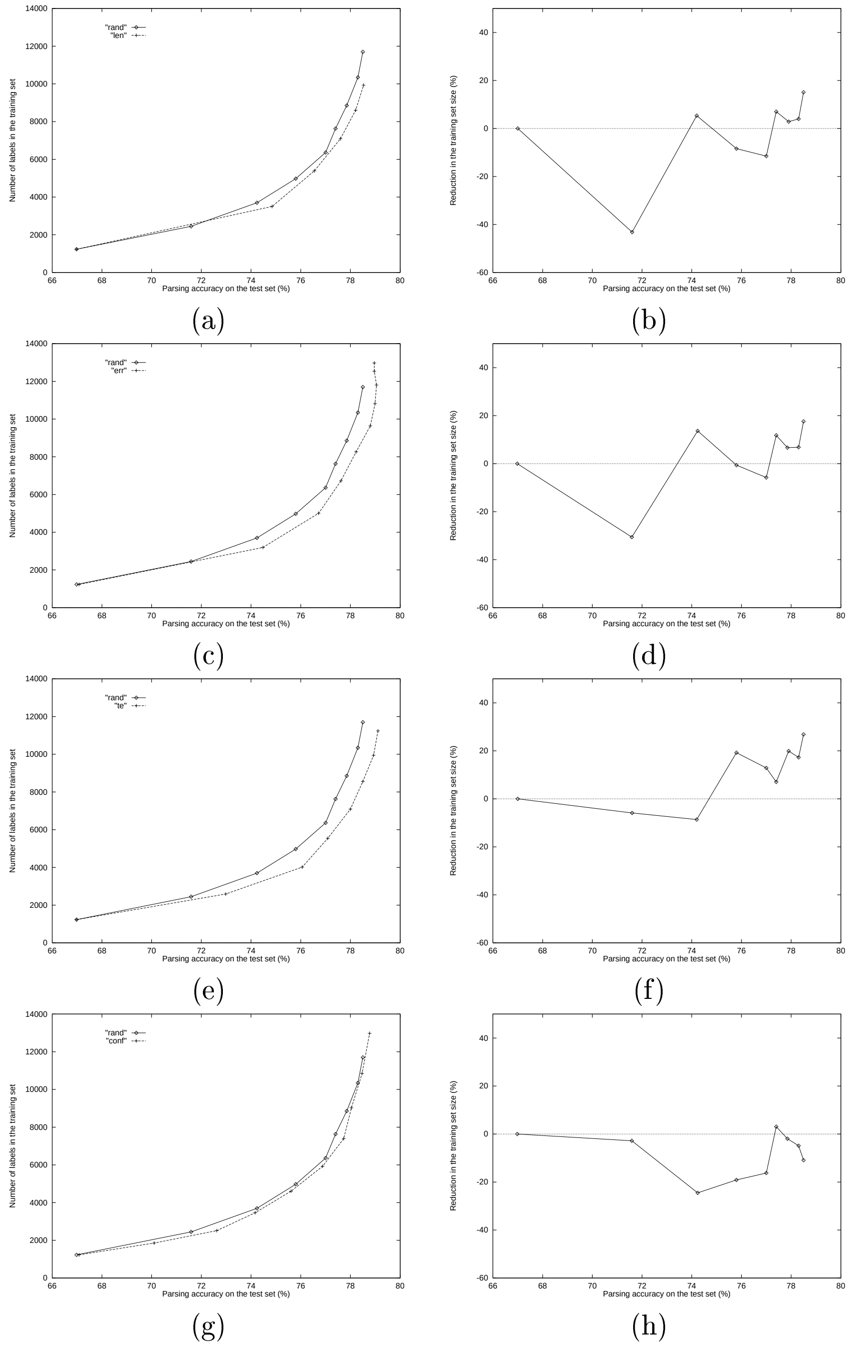


Figure 5.12: The learning rates of the induction processes using examples selected by the evaluation functions when the candidate-pool is small. We compare f_{len} to the baseline in plots (a)-(b), f_{err} in (c)-(d), f_{te} in (e)-(f), and f_{conf} in (g)-(h).

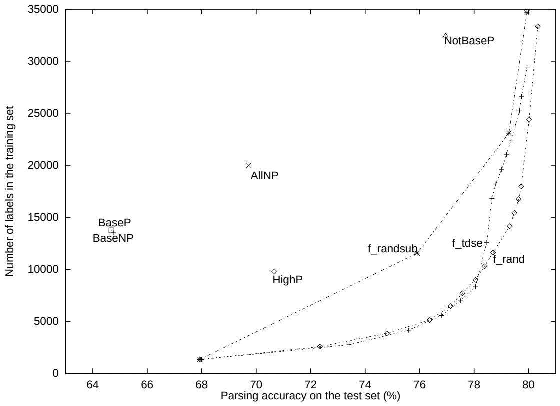


Figure 5.13: The top-down substring entropy evaluation function, f_{tdse} is compared to two baselines: random substring selection and random sentence selection.

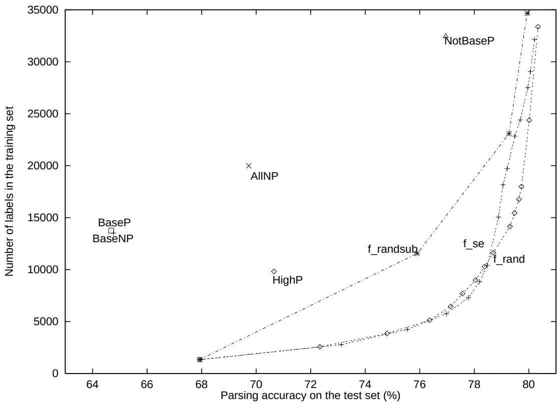


Figure 5.14: The substring entropy evaluation function, f_{se}, is compared to two baselines: random substring selection and random sentence selection. This figure assumes that the annotator's effort in determining whether a substring is a constituent is one-tenth of the effort in actually annotating the constituent.

Chapter 6

Conclusion

In this thesis, we report a significant step forward in addressing the challenges of grammar induction for natural language applications. We have shown through empirical studies that the PLTIG formalism, a probabilistic lexicalized tree rewriting system, is a computationally tractable grammar representation that performs well for natural language processing applications such as parsing and language modeling; we have developed an efficient partially-supervised learning algorithm for inducing PLTIGs from bracketed training corpora; and we have augmented the basic learning algorithm with two training techniques that reduce the amount of manual-annotation required in the training data: grammar adaptation and sample selection. While both techniques can minimize the learner's dependency on manually annotated training data, our studies suggest that sample selection provides a more dramatic reduction. Through experiments, we have found that adaptive methods induce better grammars than direct induction when the training data is sparsely annotated, and although these grammars do not perform as well as grammars induced from fully labeled training data, they show less degradation than those induced directly. Sample selection has an additional advantage in that the resulting grammars suffer no degradation in quality compared to grammars induced from fully labeled training data. With a good evaluation function such as the tree-entropy metric, we were able to reduce the amount of training annotation by 36%. Furthermore, we have successfully applied sample selection to learning a PP-attachment model, showing that it can halve the number of annotated training examples needed.

These studies suggest that inducing grammars represented in a lexicalized formalism such as PLTIG and learning techniques such as grammar adaptation and sample selection are promising directions for further exploration. Although the methods we proposed have shown significant improvements over the baselines, there are undoubtedly more sophisticated approaches that would lead to even better performances. We now speculate as to some potentially fruitful directions of research.

Rather than being anchored by part-of-speech tags, PLTIGs would represent the language better if their trees are anchored by surface words in the corpus. The major challenges of directly applying the PLTIG learning algorithm in Chapter 3 are managing the scale-up of the vocabulary size and addressing the data sparsity problem. As shown in Expression (3.6), the number of PLTIG parameters grows quadratically with respect to the vocabulary size; therefore, inducing a PLTIG that covers all the words of the WSJ would require much more annotated training data than what we have been using in our studies. It would be interesting to test whether the learning algorithm augmented by sample selection may help inducing lexicalized grammars.

Comparing the outcome of our studies on sample selection and grammar adaptation, we have identified several areas of investigation to improve grammar adaptation. First, while the adaptation method trades off annotation effort for performance accuracies of the trained grammars, sample selection typically does not worsen the performance of the trained grammars. It may prove helpful to incorporate both adaptation and sample selection into the learning process. However, we need to prevent the potential conflicts that may arise. For instance, a grammar's knowledge about the existing corpus may mislead the selection algorithm to pick poor examples for the new corpus. Second, currently, it is difficult to adapt grammars between two very different corpora because the structures learned by the grammar in the first corpus are typically too different from those in the new corpus. In fact, the grammar's old knowledge sometimes may even hinder learning the new corpus (by setting some parameters to have near-zero probabilities, for example). We need a way to reset these parts of the grammars. Finally, based on our experimental results suggesting that higher-level constituents are the most informative linguistic units, it may be useful to consider using partial parsers that can automatically detect complex nouns and sentential clauses to further relieve human annotators from their tasks.

Our exploration of sample selection indicates that it has great potential for complex natural-language learning tasks such as grammar induction. This line of research can be furthered in several directions. First, since the ideas behind the proposed evaluation functions are general and independent of formalisms, it would be interesting to determine their effect on other parsers through empirical studies. Second, alternative formulations of evaluation functions for the single-learner system may result in further annotation reduction. In particular, if a computationally efficient confidence-based evaluation function for evaluating grammar induction training examples is developed; one could experiment with new evaluation functions that factor in both the uncertainty and confidence ranking criteria, as we have done for PP-attachment. Although the results of the current study suggests that the interaction mode of selecting entire sentences is more helpful than substring-based selection, we believe that better evaluation functions for substring selection can be developed. Another area of interest is to consider committee-based sample selection using multiple learners. Finally, sample selection may be applied to other natural-language learning algorithms that have been limited by the sparsity of annotated data.

References

Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190, March.

James K. Baker. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA, June.

Srinivas Bangalore. 1997. *Complexity of lexical descriptions: relevance to partial parsing.* Ph.D. thesis, University of Pennsylvania.

L.E. Baum. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, 3:1–8.

Eric Brill and Philip S. Resnik. 1994. A rule based approach to PP attachment disambiguation. In *Proceedings of COLING.*

Peter F. Brown, Vincent J. DellaPietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the AAAI*, pages 598–603, Providence, RI. AAAI Press/MIT Press.

Ciprian Chelba and Fredrick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proceedings of COLING-ACL*, volume 1, pages 225–231.

Stanley F. Chen and Joshua T. Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 310–318, Santa Cruz, California, June.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the ACL*, pages 456–463, Hong Kong, China, October.

Noam Chomsky. 1964. *Syntactic Structures.* Mouton.

David Cohn, Les Atlas, and Richard Ladner. 1994. Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

Michael Collins and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 27–38.

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

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML 2000*, page To appear.

Thomas M. Cover and Joy A. Thomas. 1991. *Elements of Information Theory.* John Wiley.

Christopher Culy. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, 8:345–351.

A.P. Dempster, N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38.

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

Sean P. Engelson and Ido Dagan. 1996. Minimizing manual annotation cost in supervised training from copora. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 319–326.

Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. 1997. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168.

Atsushi Fujii, Kentaro Inui, Takenobu Tokunaga, and Hozumi Tanaka. 1998. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4):573–598, December.

Joshua Goodman. 1997. Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technologies 1997.*

Sheila A. Greibach. 1965. A new normal form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12(1):42–52.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *DARPA Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania, June. Morgan Kaufmann.

Ulf Hermjakob and Raymond J. Mooney. 1997. Learning parse and translation decisions from examples with rich context. In *Proceedings of the Association for Computational Linguistics*, pages 482–489.

H. Jeffreys. 1948. *Theory of Probability.* Clarendon Press, Oxford, second. edition.

Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands: North-Holland, May.

Frederick Jelinek. 1997. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunction grammars. *Journal of Computer and System Sciences*, 10(1).

Arivand K. Joshi, 1985. *How much context-sensitivity is necessary for characterizing structural descriptions – Tree Adjoining Grammars.* Cambridge University Press.

J. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, University of Hawaii.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the langauge model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3):400–401, March.

K. Lari and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

David D. Lewis and Jason Catlett. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156.

David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12.

Bernard William Lindgren. 1959. *Introduction to Probability and Statistics*. Macmillan, New York.

David Magerman. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University University, February.

David Magerman. 1995. Statistical decision–models for parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*, pages 276–283, Cambridge, MA.

Robert McNaughton. 1967. Parenthesis grammars. *Journal of the ACM*, 2(3):490–500.

Paolo Merlo, Matthew W. Crocker, and Cathy Berthouzoz. 1997. Attaching multiple prepositional phrases: Generalized backed-off estimation. In *Proceedings of the Second Conference on Empirical Methods in NLP (EMNLP-2).*

Fernando Pereira and Yves Schabes. 1992. Inside-Outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 128–135, Newark, Delaware.

David M. Perlmutter and Scott Soames. 1979. *Syntactic Argumentation and the Structure of English.* University of California Press.

Stephen G. Pulman. 1986. Grammars, parsers and memory limitations. *Language and cognitive processes*, 1(3):197–225.

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

Sanguthevar Rajasekaran and Shibu Yooseph. 1995. Tal recognition in $O(M(n^2))$ time. In *Proceedings of the 33rd Annual Meeting of the ACL*, pages 166–173, Cambridge, MA.

Adwait Ratnaparkhi. 1998. Statistical models for unsupervised prepositional phrase attachment. In *Proceedings of COLING-ACL*, volume 2, pages 1079–1085.

Philip S. Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th International Conference on Computational Linguistics*, Nantes, France, August.

Beatrice Santorini. 2000. Personal communication, March.

Yves Schabes and Richard Waters. 1995. Tree insertion grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the tree produced. *Computational Linguistics*, 21(4):479–512.

Yves Schabes, Anne Abeille, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING '88)*, August.

Yves Schabes. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, August.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 426–432, Nantes, France, August.

Stuart Shieber and Yves Schabes. 1990. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pages 1–6.

Stuart Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Jiri Stetina and Makoto Nagao. 1997. Corpus based PP attachment ambiguity resolution with a semantic dictionary. In *Proceedings of the Fifth Workshop on Very Large Corpora*, pages 66–80.

Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. 1999. Active learning for natural language parsing and information extraction. In *Proceedings of ICML-99*, pages 406–414, Bled, Slovenia.

K. Vijay-Shanker and Aravind K. Joshi. 1988. Feature structure based tree adjoining grammars. In *12th International Conference on Computational Linguistics*.

The XTAG-Group. 1999. A lexicalized tree adjoining grammar for english. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.