Learning Probabilistic Grammars for Language Modeling

by
Glenn Carroll
B.Sc., UCLA, May 1986
M.Sc., Brown University, May 1989

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1995

This dissertation by Glenn Carroll
is accepted in its present form by the Department of
Computer Science as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ____

Eugene Charniak

Recommended to the Graduate Council

Date ______________ ____

Mark Johnson

Date ______________ ____

Leslie Kaelbling

Approved by the Graduate Council

Date ______________ ____

Dedication

In memory of Pamela Ann Sweeney, 1973-1993.

requiescat in pacem

Acknowledgements

As with any large work, this research was not the product of my efforts alone. My thanks to my committee, Eugene Charniak, Mark Johnson and Leslie Kaelbling, for sharing what were very different, sometimes surprising, and often enlightening views of my research. Thanks also to my parents for their enduring patience and sympathy when the project threatened to stall. Thanks to my friends Kathy Franz and Sandy Lee for being there and being supportive when I needed it most. And finally, thanks to the AI gang, especially Tony, Jak, and Moi, for listening to all those practice talks.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Traditional approaches to language modeling and parsing share essentially no techniques. Parsing is the task of finding the syntactic structure of sentences and, at least for natural languages, has been primarily the domain of linguists. Parsing methods are generally based on elaborate grammars requiring years of laborious development by experts (read: "linguists", such as Alshawi [1] and Hirschman [2]). Language modeling is the task of finding a distribution over strings, a task made feasible by computers and originally of interest to computer scientists and engineers working on speech recognition. It relies on fairly simple statistical models which require large amounts of data and computation (e.g., [3, 4]). The former approach ignores statistical methods in favor of expert knowledge, while the latter commits the inverse error. It seems clear that both sides could benefit by borrowing ideas from the other.

In this research, we develop methods for learning probabilistic context-free grammars (PCFGs). A PCFG is an ordinary context-free grammar with a set of probability distributions over the rules–we will be more detailed later. Such grammars provide a single framework in which both parsing and language modeling can be addressed. As they are context-free grammars, they are recognizable and understandable by linguists. More important for our purposes, they allow straightforward application of linguistic knowledge. As they are probabilistic, they can also be used to define a distribution over strings (a language model) and they can be trained by statistical methods similar to those used for traditional language models.

Our general intention is to build a system which accepts a corpus of text as input and produces a grammar as output. In the long term, our goal is to produce grammars suitable for both parsing and language modeling. The present work concentrates on the latter task, as there are practical methods for training grammars for this task and for evaluating performance on it. The difficulty with evaluating parsing accuracy is that it requires a source of correct parses, as well as some commitment to the grammatical theory underlying those parses. At present, no large, pre-parsed corpora are available, so methods such as those reported by Pereira and Schabes [5] cannot be usefully applied.

The primary claim of this thesis is that linguistic knowledge can be usefully put to work in developing statistically-trained language models. In support of this claim, we describe a system we have built which employs some simple but specific linguistic knowledge. We demonstrate that the learner is *practical.* It learns grammars from real, cross-domain English data as opposed to artificial data or small English subsets. We show that the grammars it learns outperform trigram models, the current standard for language modeling. We give experimental evaluation of various strategies and parameters used, or informal argument where experiment is impractical or unnecessary. We analyze the results of these experiments, so we can give some explanation for how the system works and why. Beyond our basic system, we have developed some learning methods that are "stand-alone" and may be used by other grammar learning systems without adopting all of our framework. We have experimental results and analysis for these methods as well.

Although our orientation is towards modeling, our use of linguistic information rests on the implicit assumption that improved parsing accuracy correlates at least somewhat with improved modeling. Our success in learning provides some evidence that this assumption is not a bad one. A more rigorous evaluation would require that we measure the parsing accuracy of various grammars, and thus falls afoul of our lack of pre-parsed data. Nonetheless, in appendix A we report some tentative results, based on the small amount of pre-parsed data available to us.

The rest of this document follows the outlined thesis contributions, with a few additions at the start and end. In the next section, we formalize what we have so far stated informally, giving definitions for PCFGs and language models, performance criteria for the grammar and learner, and so on. Having estabished a working vocabulary, we place the work in context of other work in the field, particularly related and competing work. Next we describe the basic system, first working through its components, and describing how they work and their intended purpose. We describe the experients we have run, the results obtained, and our analysis of those results. We have elaborated the basic system considerably, and we describe these refinements, together with their motivation and experimental evaluation. Following this, we describe and evaluate the stand-alone methods for improving PCFG performance. We report on experiments testing SINGER's sensitivity to varying its parameter settings, and then we turn to work evaluating SINGER's effectiveness when used with an alternative grammar formalism. We spend a somewhat speculative chapter analyzing the overall system and methods, with an eye towards how these might be adapted to other languages, domains, and so on. Finally, we sketch what appear to be the most interesting next steps in research and summarize our results.

Chapter 2

Problem Description

2.1 Language Models

The canonical task for a language model is to serve as part of a speech recognition system. The usual strategy for such systems is to combine an acoustic model that proposes words based on the acoustic signal with a language model that assigns probabilities to the candidate words based on previously identified words[1]. For example, suppose the acoustic model suggests the current word might be "form" or "farm". If the two preceding words were "1040 tax", this would strongly suggest that the correct choice would be "form". While it might seem that such examples of ambiguity would be few and far between, and therefore a language model would have only a small role to play in speech recognition, quite the opposite occurs in practice. According to one group,

> We know that, in a real task, the importance of the language model is comparable to that of the acoustic module in determining the final performance. [6]

[1]Actually, the acoustic model usually assigns an initial distribution to its proposed words and the language model's distribution is then combined with this information using Bayes' Law.

Formally, a language model defines a distribution over the strings of a language, so for language L with strings s

$$\sum_{s \in L} P(s) = 1.0$$

To compare the probability of two words, we are interested in the probability of the current word w_i given the preceding words, $w_1 \ldots w_{i-1}$. In our example, we are comparing $P(\text{“farm”}|\ldots, \text{“1040”}, \text{“tax”})$ with $P(\text{“form”}|\ldots, \text{“1040”}, \text{“tax”})$. By Bayes' Law, we can compute the conditional probability in terms of the overall string probability as follows

$$P(w_i|w_1, \ldots, w_{i-1}) = \frac{P(w_1, \ldots, w_i)}{P(w_1, \ldots, w_{i-1})}$$

The numerator on the right-hand side of this equation is the string probability defined by the language model. The denominator serves as a normalizing constant which may be ignored when comparing the probabilities of different w_i.

We compute the performance of a language model as the number of bits of extra information required, on average, to predict the words of a test corpus, when the words of the test corpus are drawn from the true (English) distribution, but the model is used to do the prediction. This measure is called the *cross entropy* of the two distributions. Writing the model distribution as P_G and the true distribution of English strings as P_E, the cross entropy is

$$-\sum_{s \in L} P_E(s) \log_2(P_G(s)) \tag{2.1}$$

where s ranges over each string type in L, and P_G is non-zero whenever P_E is.

It can be shown that cross entropy is minimized when the two distributions are identical. Intuitively, this is because it cannot be easier to guess the outcome of a random event when the event is generated using one distribution but a different distribution is used for guessing. This also accords with our intuition that a perfect language model would have exactly the true probability distribution of English, or, in other words, it would assign probabilities to strings equal to their expected frequency of occurrence in an infinitely large corpus. Finding a model with minimal cross entropy is equivalent to finding a model with the true English distribution.

As a practical matter, the cross entropy measure is usually normalized to be cross entropy *per word* to facilitate comparison among models. This can be estimated by choosing a sufficiently large sample of text C, and computing

$$-(1/|C|)\sum_{s\in C}\log P_G(s) \tag{2.2}$$

where s ranges over each sentence token in C. For shorthand, we generally refer to this as the cross entropy, dropping the "per word".

2.2 Trigram models

This history of language modeling has been dominated by a simple and naive class of model called the trigram model. Trigram models condition the probability of a word solely on the two preceding words, assuming independence from all other words. For a string longer than three words, $P(s) = P(w_1, \ldots, w_i) = \prod_i (w_i|w_{i-1}, w_{i-2})$. The name trigram comes from the focus on word triples.

Although trigrams have what can only be termed a myopic view of language, they have been successful because a three-word window is enough to capture many syntactic and semantic phenomena. For example, a preposition is hardly ever followed by an untensed verb, and this can be represented in the model as a set of low probabilities assigned to such strings. Similarly, limited semantics can be modeled in the same way, as in our "form" versus "farm" example.

The performance of trigram models degrades when the word or words which contain the most useful information lie outside of a three-word window. Consider the sentence, "The submarine, which was in dry-dock last week, sank like a stone." In calculating the overall probability of this string, the tri-gram model computes the probability of "dry-dock" given the words "was in", and the probability of "sank" given the words "last week". In both cases, the important preceding words ("submarine" in the first case, "dry-dock" in the second) are missing from the three-word window which is all that the tri-gram model uses. Since the words are lexically distant, they do not contribute to the important probability calculations and the model's performance suffers.

The performance of a language model is determined by three factors: the quantity of training data, the scheme for estimating the model's parameters, and the independence assumptions on which the model is based.

2.3 Grammatical Models

Where trigram models assume that three-word tuples occur independently, the primary assumption of grammatical language models is that sentences occur independently. For probabilistic context-free grammars, the second assumption is that the probability of a rule is conditioned only upon the head of the rule. In other words, for a grammar with non-terminals NT and terminals T, for any non-terminal X,

$$\sum_{\alpha \in (NT \cup T)^*} P(X \rightarrow \alpha) = 1.0$$

The probability of a parse is the product of the probability of each rule occurrence in the parse. The probability of a sentence s can be calculated in terms of the probabilities of the parses of s as follows

$$P(s) = \sum_{t \in parses(s)} P(s,t) = \sum_{t \in parses(s)} P(t)P(s|t) \tag{2.3}$$

where t is a parse tree of s. We return to the second term, $P(s|t)$ in our discussion of future work in chapter 10. The aim of this thesis is to find methods for learning grammatical models which will maximize the first term. As the equation shows, this term is important to the performance of any grammar-based language model. Perhaps more to the point, there is no means for gathering the data for a grammatical model without first having a suitable grammar. The effect of ignoring $P(s|t)$ is to turn from modeling strings of English words to modeling strings of part-of-speech tags assigned to an English text. For the remainder of this paper, we mean modeling these tag strings when we speak of modeling English, unless explicitly stated otherwise.

2.4 Practical Issues and Requirements

For a grammar to have finite cross entropy, it must parse *every* sentence in the test corpus used to evaluate it. This implies that either there is no noise in our sample, or we intend to model the noise right along with the data, as trigram models do. In our own case, there are two kinds of noise to deal with.

The first kind of noise is noise in the word tags, which arises when the machine or human tagger makes a mistake. This kind of noise occurs at very low levels in our data, and we allow the grammar probabilities to model it. Since the error rate is low, we do not believe that these errors significantly alter or distort the learned grammar.

The second kind of noise consists of grammatical errors in the original sentences, which raises the sticky issue of grammaticality. Grammaticality is a problem because no two people seem to agree on what is or is not grammatical (see, for example, [7]). As our system attempts to model language, it is obliged to assign a non-zero probability to *each* sentence that might appear, regardless of anyone's judgment as to the sentence's grammaticality. That being the case, the learner must attempt to find grammar rules to parse "agrammatical" sentences as well as possible, although presumably these rules will end up with very low probability.

Despite such a liberal attitude towards grammaticality, the grammar will improperly assign some sentences zero probability, unless steps are taken to prevent it. The difficulty does not lie with the learning, as it would not be hard to relax the constraints and allow all sentences to be learned. The problem occurs when the learned grammar is tested against new data; given a reasonably large set of test data, which the learner was not given, the chances are good that the learned grammar will fail to parse at least one sentence. If our performance criterion is net cross entropy, then this becomes infinite if even a single sentence in the sample is unparsable. This does not allow for useful comparisons among grammars, nor among learners.

One way out of this problem is to add a set of catch-all rules to the grammar. These would allow the grammar to parse arbitrary sentences, and could be trained using a separate set of data. While such a solution is technically pure, it complicates the grammar learning without improving it in any substantial sense. Instead of this, we report three figures, including the cross entropy per word, the coverage of the grammar (generally $> 99\%$), and a term we have named λ that indicates the relative performance of the grammar model in competition with a trigram model trained over the same data.

The trigram competitor is what is called a "backed-off" trigram model, meaning it is optimally smoothed with a bigram and unigram model. Backing off the model is a standard method for increasing robustness in the face of sparse data. As our vocabularly is closed, the model will never encounter unknown words, so that problem does not arise for us.

We measure the comparative performance of the two models by treating both as components of an overall test model which is the smoothed combination of the the two. In other words, for a grammar model G and trigram model TG, we compute the probability of a string as

$$P_test(s) = \lambda P_G(s) + (1.0 - \lambda) P_{TG}(s)$$

We assign both models an initial weight of 0.5, and tune these values using the forward-backward algorithm[2]on held-out test data. We measure performance as the final weight assigned to the grammar model, and call this λ. (The final weight assigned to the trigram model is given by $1.0 - \lambda$).) A λ of 0.5 would indicate that the grammar and the trigram performed identically well, overall. If either model had superior cross entropy on every sentence, its weight would reach 1.0, indicating complete dominance.

One way to view this is as a scoring procedure in which each sentence is worth an equal, fixed reward. We are dividing this reward between the two models according to the weighted probability (λP_G or $(1 - \lambda) P_{TG}$) that each

[2]Forward-backward is a version of expectation-maximization quite similar to the Inside-outside algorithm described later. It revises the λs so as to maximize the smoothed probability, up to a local maximum.

model assigns the sentence. Whichever model gets the most weight is the better model. For our data, it is usually the case that the two models assign widely differing probabilities, so one model captures most of the reward. This justifies reading the λ score as the fraction of sentences for which the grammar model outperforms the trigram model.

The key advantage of this measure is that it assigns a finite value to each sentence, so it is insensitive to minor holes in the coverage. Aside from this, the two measures almost always track each other, which is desirable, since cross entropy is essentially the correct measure, only sensitive to zero-probability strings.

2.5 Theoretically Optimal Learner Performance

In this section, we discuss what optimal performance for a learner would be, and why our learner is not optimal. So far as we know, no extant scheme for grammar learning is optimal in any sense resembling our definition below.

Intuitively, a theoretically optimal algorithm should provide the correct answer given enough data, or at least an ever-closer approximation to the correct answer as the quantity of learning data increases. Since our data is drawn from a distribution we have to phrase this in terms of expectation. Formally we might write

$$\lim_{|C|\to\infty} E(ce(G(C))) = e(English)$$

That is, as sample size increases, the expected cross entropy of the grammar learned from the sample should converge to the entropy of English, where the expectation is taken over all sample corpora of the given size. This is not a precise definition, but rather than sharpen it we will argue that the time is not quite ripe to construct a theoretically optimal algorithm.

The first shortcoming of this definition is that it is too weak. It requires good performance only in the limit, as sample size approaches infinity. For any finite sample, no guarantees are made, and so an algorithm may be theoretically optimal algorithm and still perform poorly. Conceivably, one could extend the optimality definition to include some notion of speedy convergence, but it is doubtful whether any meaningful promises could be made.

The second shortcoming is, paradoxically, that the definition is too strong, because it rules out algorithms which perform well but do not reach perfection in the limit. This cannot be addressed by altering the definition of optimality, as it is inherent in the problem of learning a complex model from a small amount of data. The realities of language learning are that one must *generalize* from the data. For the present and the foreseeable future, we do not have access to a sample containing all the different sentence types of English. Formally this implies a learner must guess the answer for some portion of the test data, and *it has no information on which to base these guesses.*

For any problem in which a great deal of generalization is required, these guesses are crucial to the overall performance of the system, and while the learner may have no information, often the person building the learner does. We call this information *bias*, both because it amounts to a prejudice and because it is related to the bias error of a learner, which we will describe in a moment. Probably the most common bias used for any statistics-based estimator is *smoothness*. For example in surface fitting, assuming smoothness amounts to claiming that there are no sharp peaks or valleys. To the extent that the assumption is true, it helps the learner by reducing the number of choices when guessing on unseen data. To the extent that it is false, the assumption increases the error by obliterating sharp peaks or valleys in the surface.

The two kinds of error described above correspond roughly to what Geman, et al [8] call bias and variance error. They develop a formula for the expected error of a learner which must generalize, and break it into the sum of these two error terms. As the details are lengthy, we present only the intuitions here. Bias error is the expected difference of the expected learner's answer and the correct answer. A learner which reproduces its training set exactly has zero bias error with respect to that data. Variance error comes from the amont of guesswork a learner carries out. If the learner can produce many quite different answers from a sample of a given size, then it will have a high variance error. A learner which ignores its data and always produces the same answer has zero variance error.

To return to our smoothing example, smoothing trades variance error for bias error. Various tradeoffs can be made, depending on both the kind of smoothing employed, say, splines versus polynomials, and how much smoothness is assumed. A very strong assumption would be that the surface is completely flat; a much weaker one would be that the surface can be fitted with a tenth-order polynomial. It is generally up to the system builder to choose what sort of smoothing to use; optimal parameter settings can usually be chosen algorithmically or by exhaustive search if necessary. Generally this produces a system which, with high probability, will perform about as well on novel data as it does on the test data.

The trouble with bias is that an optimal learner must have both bias and variance error approach zero in the limit, and this is generally not true for a fixed-bias system. To approach zero error, an optimal algorithm either starts off with a zero-bias learner, which generally results in bad performance on realistic data sets, or it follows a schedule for reducing the bias as the training set size grows. This schedule amounts to changing the parameters of the chosen bias so that the bias has less and less effect. For any finite data set, there is an optimal setting for these parameters which can be determined empirically, which makes the schedule something of a moot point.

Another way to summarize all this is to say that an optimal algorithm is portable, because it guarantees good performance for any language within its range. Its disadvantage is that the guarantee only applies in the limit, so that it may perform poorly for any or all finite training sets. A learning algorithm such as ours which does not steadily diminish bias is non-portable, because the bias is adjusted to the data set. Pragmatically speaking, optimal algorithms must make use of some bias to obtain acceptable performance and therefore suffer from some performance penalties when used with other languages/data sets.

Our point here is that some prejudice on the part of the learner is required for good performance. Our claim in this thesis is that linguistics provides a rich source of information that can be mined for use as bias. Statistical strategies alone are not enough to produce a good grammar, nor are “generic” forms of bias such as smoothing.

Chapter 3

Related Work

Grammar learning, and English grammar learning in particular, has become a hot area for experimental work in the past five years. Earlier work on practical systems, as opposed to theory, was hampered by the limitations of the existing hardware and software, and the unavailability of adequate amounts of machine-processable text. In this section, we trace the development of grammar learning from early attempts at learning non-probabilistic grammars, through learning probabilistic grammars on toy domains, up to present efforts at learning grammars that cover substantial portions of English. For the most part, these grammars have been intended for parsing, not modeling, and pay more attention to the linguistics than the learning issues. In the following section, we summarize the development of language modeling, which generally takes the opposite tack, concentrating on the statistics and paying little attention to the linguistics. We begin with a brief discussion of the relevant theoretical results, as they indicate some limits on what levels of performance we can expect to reach.

3.1 A Brief Digression to Theory

Probably the most-cited work in language learning is Gold [9], which showed that it is not possible to *identify* a language even in the limit as the training set size approaches infinity. The task of language identification is to indicate, for any string presented, whether the string is in or out of the target language. The difficulty with learning identification is that with an infinite language there may always be unseen strings, no matter how large the training set size grows. Our own problem may be considered a probabilistic approximation of the language identification problem. We want the learner to identify strings in the language which represent most of the probability mass of the language, which makes the task somewhat easier. We also want the learner to identify the probabilities of those strings, which makes the task considerably harder. There's no such thing as a free lunch.

On the more practical side, [10] and [11] showed that learning the smallest DFA consistent with a set of training data is NP-complete. Even learning a DFA whose size is a polynomial of the smallest DFA is NP-hard, under the assumption $P \neq NP$, [12] and not PAC-learnable, under various cryptographic assumptions [13]. Since a compact machine is a general one, these results say that good generalization is hard to achieve. The mitigating factor is that these results hold for a learner which must process arbitrary input. It is still possible for some learning strategies to do well on some classes of data, which is what we are trying to arrange through our use of linguistic information.

3.2 Grammar Learning Experiments

While statistical grammar learning has roots reaching back into at least the 70's (see Cook [14] and Wolff [15]), early efforts were limited to toy domains and ad hoc learning methods. While Cook and Wolff were using information-theoretic metrics related to cross entropy, they lacked an algorithm for reducing the cross entropy of a PCFG on a set of traiing data. It was not until 1991, when Jelinek et al. [16,Jelinek Mercer 1991] popularized Baker's Inside-Outside algorithm [17, 18] for PCFG training, that statistical grammar learning became popular. The Inside-Outside algorithm (hereafter I-O) iteratively reduces the cross-entropy of a data set with respect to a grammar, up to a local minimum.

With the I-O algorithm available, a very simple approach to grammar learning becomes possible. The strategy is to generate all possible rules, randomly initialize them, and use the I-O algorithm to train the result. Poor rules should have their probabilities reduced to zero, at which point they can be discarded from the grammar. Lari and Young [19] evaluated this approach using a Chomsky Normal Form grammar. They preselected the number of non-terminals, and had the system start with all possible rules, randomly initialized. They reached the effective limits of this approach with about 10 non-terminals, and a simple palindrome language as the target, as permitting all possible rules led to explosive parsing costs. For a 10 word sentence, the number of possible parses is approximately 1.7×10^{13}. To be sure, parsing cost is in principle restricted to being $O(N^3)$, where N is sentence length, but this algorithm forces that worst-case cost. What's more, cost also grows linearly in the size of the grammar, which can add substantially when the grammar becomes complex enough to model English. At the same time, the number of parses stemming from large rule sets bogs down the I-O algorithm, slowing convergence. The result is that I-O alone will not suffice for learning grammars. The problem of rule selection must be addressed in a way that does not lead to excessive parsing cost.

Pereira and Schabes in [5] describe a scheme which addresses some of the training difficulties of Lari and Young. They also start with all possible (CNF) rules initialized with randomized probabilities, but they avoid the worst problems by using much bracketed sentences for training data. The bracketing indicates a partial parse tree for each sentence. They combine this with a modified I-O algorithm that eliminates rule uses that disagree with the bracketing. This greatly reduces the number of parses, allowing faster convergence, and presumably a better final performance. The drawback of this scheme is that it requires fairly large amounts of pre-parsed text, which are expensive and difficult to produce. So far, such text is not available in anything like the overall quantity of text, or the quantity of high-quality pre-tagged text. Moreover, bracketing accuracy is still fairly low.

A very different approach has been to try to combine statistics and linguistic knowledge to learn grammars. Brent [20] uses this strategy to learn subcategorization frames for verbs, working from raw text and some a priori grammatical knowledge deployed as constraints. While his system appears to be successful in its limited domain, it is not obvious how to scale it up. The linguistic knowledge and the statistical methods deployed to learn verb frames were highly specialized to that particular task, and there are no straightforward means to extend them to the rest of the grammar learning task. (This work is similar to earlier non-statistical learning efforts such as Berwick [21], which also built fragments of English.)

We now turn to other work directed towards building grammars intended to cover all or at least most of English. Hindle [22], like us, is interested in building grammars for both parsing and language modelling. Like us, he chooses a version of dependency grammars for his formalism, but his are lexicalized, using words as well as tag information. Lexicalized grammars require far larger amounts of data than non-lexicalized ones, as the number of parameters is greater. Interestingly, Hindle regards his coverage problem as an ongoing condition of the grammar. Instead of turning to smoothing to eliminate or reduce zero-probability rules, he elects to make learning a perpetually ongoing task. In other words, the system will often fail to parse sentences according to its grammar, even after processing very large amounts of text. When a parse fails, the learner creates new rules, using heuristics to choose rules that are similar to those already in the grammar.

While this system is close to our own in many of its aims and practices, we do not believe it demonstrates a grammatical language model. The trouble is that it is not clear how such a system can assign meaningful probabilities in the face of continued parse failures. In our own system, a failed parse implies zero probability for the sentence, and so we go to some trouble to avoid these. In Hindle's scheme, parse failures are expected to occur frequently, and the learner will be invoked to patch up the grammar; the question is, what probability gets returned from a string that was unparsable before it was seen? It seems likely that the probabilities for the grammar must be revised in such a way as to reduce the probability of some sentences that were previously parsable. In that case, however, the reported probabilities do not make sense, as they may sum to more than 1.0.

Like Hindle, Schabes [4,23] reports work on lexicalized grammars, in this case lexicalized tree-adjoining grammars. As with Hindle, he faces a sparse data problem, as each word may be associated with several syntactic rules. His tack on this problem is to use more informative data. Rather than training his grammar with raw text, he uses the pre-parsed Penn Treebank, which allows him to rule out most parses. As we mentioned earlier such data is harder to generate, and not available in the same quantities as tagged text. It appears that even the several million words of pre-parsed Wall Street Journal articles are insufficient to train this model, as there are no published results at this time.

Briscoe and Waegner, [24], are interested in building grammars with reliably correct parsing, and wish to use learning to deal with undergeneration. The idea is to extend a highly developed, hand-built CFG which encodes features (e.g., number, tense, etc.) into the non-terminal categories. Although dynamic rule construction would be a natural choice for dealing with undergeneration, their scheme is strictly batch. The set of possible rules is implicitly defined before training, and the rule probabilities are initialized in two ways: the set of rules from the hand-built grammar are initialized with a "high" probability, and the rules in the remaining set of implicit (permitted) rules are assigned lower probabilities. Since the entire space of possible rules is available for training, they place sharp restrictions on the rules in order to keep the space small.

This work is closest in spirit and practice to our own work. It takes a step in the right direction by using a significant chunk of linguistic knowledge, in the form of their hand-built grammar. This is probably the most straightforward way of leveraging off of existing linguistic knowledge. Unfortunately, they make fairly weak use of the knowledge embodied in the grammar. Rule probabilities are initialized to one of two values, and the entire space of possible rules is considered for parsing. Apparently this is still too much of a burden for the Inside-Outside algorithm, as they report mediocre results on both parsing accuracy and perplexity measures.

Magerman [25] has developed a system that addresses what he calls the *treebank recognition problem.* This is similar to parsing, only oriented towards reproducing the partial-parse style bracketings seen in treebanks. In several respects, Magerman's work is the polar opposite to our own. Where we concentrate on modeling, he works on parsing, where we are bringing linguistic information to a traditionally statistical task, he brings statistical information to a traditionally linguistic task, and, most importantly, we disagree on the importance of using linguistic information as bias. Magerman takes the position that the ideal would be to use no grammarian-supplied data at all. It isn't entirely clear what sort of information could legitimately be supplied by a system builder, but a reasonable reading is that generic statistical bias such as smoothing would be acceptable. This is a roundabout way of saying that no bias specific to the task of natural language learning is necessary. This contradicts the current psychological and linguistic consensus [26,27] that humans come predisposed to learn one or more of a small set of grammars, and other possibilities equally well fitted to the data are simply not learnable by humans. It also contradicts our own claim that bias is absolutely necessary for good performance. Indeed, despite this position, linguistic bias creeps into Magerman's system in at least two forms, large and small. The former is the grammar, which his system does not learn, but accepts as delivered, and the latter is a technical detail (his procedure for deciding which word to use as the head and secondary head of a rule). Both are crucial to his system's performance and difficult to learn, by his own acknowledgment.

While we single out Magerman here, he represents something of a trend in computational linguistics. The fashion seems to be to throw out the linguistics in favor of computation [19,24]. The motivation for this move seems to come out of the great advances made by the language modeling community while parsing technology was stalled. While hand-development of grammars has failed to provide an accurate, wide-coverage English grammar, linguists have developed a wealth of theoretical and practical knowledge about grammars which may be usefully deployed to leverage statistical learning. We do not regard any particulary linguistic theory as written in stone, but then, probably most linguists don't either. Rather, these theories are useful heuristics, whose value for modeling can be evaluated in a statistical framework.

3.3 Language Modeling

The history of language modeling is primarily the phenomenal success story of the humble trigram model. Starting in the 70's, Jelinek, Mercer, Brown, and others in the IBM Speech Recognition group developed the trigram model. They showed how to train and use these models [28], and how to use word clustering to improve performance [29]. Church and Gale at AT&T [3] compared the effectiveness of the elaborate smoothing methods necessary to cope with sparse data, even when millions of words were available for training data. Bahl et al. [30] attempted to use decision trees to extend the basic trigram model to be an n-gram model, for some n greater than three.

The last effort is the crucial one, as the trigram model's failure to handle "long-distance" dependencies was clear from the start. Bahl's work attempted to correct this by allowing the model to look back as much as 20 words. With a reduced vocabulary and great deal of computation, the tree-based n-gram model was capable of a modest performance gain over the trigram model. Judging by its lack of wide-spread acceptance in the speech recognition community, the expense has not been considered worthwhile. Trigrams, together with minor variations of them, remain the mainstay language model of speech recognition systems.

Chapter 4

Systems, Experiments, and Analyses

In this section we describe the system we have built, the choices we made while building it, and the experiments and analysis on which we based our decisions. We have dubbed the program SINGER , a contraction of SINGlE Reader, which reflects our realization that sentences need be read only once during rule acquisition.

We break down the problem of learning a probabilistic grammar into two pieces: inducing the rule set, and tuning the rule probability distributions. Our system addresses these problems essentially in separate phases, which we call rule acquisition and rule training, respectively. Actually, while these phases are distinct, there is some overlap in their solution of the two sub-problems, as the rule probabilities are revised somewhat during acquisition, and some rules are generally weeded out during training.

From a machine-learning viewpoint, the two phases have quite different characteristics. In the first phase, rule acquisition, the dominant issue is how to cut down the search space so that a relatively simple algorithm can succeed. At present, there are no general-purpose, industrial-strength algorithms for finding rules for a PCFG, and the unconstrained search space is prodigiously large. In consequence, most of our solution for rule acquisition will be a matter of constraints, generally in the form of linguistically motivated preferences. The induction algorithm proper will be so simple as to be unworthy of the designation "algorithm".

Rule training, in contrast, has very little in the way of constraints. There simply isn't much to say about the probabilities beyond our desire to maximize the likelihood of the data while generalizing well to new data. Unlike rule acquisition, there is a well-known algorithm, Baker's Inside-Outside algorithm, for revising the rule probabilities so as to maximize the probability of the data. While it is not perfect, it has no serious competitors at present. As the properties of this algorithm drive some of our later decisions, we describe it next.

4.1 Training

The Inside-Outside algorithm[1], mentioned earlier, revises a grammar's probability distribution so as to minimize its cross entropy evaluated on the training data. It operates by using the rule probabilities and the data to estimate the frequency with which each rule is used in parsing, which we call its *count*. Then the rule probabilities are estimated based on these counts. The process is iterative, and asymptotically approaches a local minimum.

The primary requirements for any training algorithm are that it operate without supervision and that it minimize cross entropy. I-O fulfills these conditions with only minor caveats and drawbacks. The caveats are the local minimum problem, and the fact that unparsable sentences can only be ignored by I-O. So long as the number of unparsable sentences is kept small, this is a minor issue. The main drawback to using I-O is that it requires parsing the entire corpus, which can be expensive with large grammars and/or long sentences. There is, however, no better choice for training presently available. For all the experiments

[1] A.k.a. the Baum-Welsh algorithm, a version of an expectation-maximization algorithm. Our particular version of the algorithm is extended to allow context-free grammars which need not be in Chomsky Normal Form.

```
sort the training corpus;

while there are sentences in the training corpus;
{
     read the next sentence;
     if the sentence is not parsable;
          add rules for the sentence;

     gather rule count statistics;
}
```

Figure 4.1: The basic algorithm with for rule acquisition.

reported below, we used 10 iterations of I-O for the training. While this does not reach the (locally) optimal cross entropy, it comes reasonably close to it with a reasonable investment in training time.

4.2 Rule acquisition

Our basic strategy for rule acquisition is to arrange it so that we only add a few rules at a time to an existing, reasonable grammar, and we train these rules immediately, both to cut down the number of rules added, and to give them reasonable values for ongoing parsing.

In figure 4.1 we give some pseudo-code for the rule acquisition phase of our algorithm. We start with a seed grammar, which may be empty. The corpus is sorted, so as to present sentences in increasing length order. Each sentence is parsed and rule usage statistics are kept, as for a normal iteration of I-O. Whenever a sentence cannot be parsed with the grammar, SINGER acquires new rules to correct the deficiency. Acquiring new rules is a three step process, involving generating the rules, initializing them, and selecting which ones to keep.

Figure 4.1 summarizes the scheme for adding new rules. The generation is carried out exhaustively, meaning every rule which could possibly be used to parse the sentence is potentially generated. We use constraints to reject some rules, both before and after generation. (More on these later.) The rules are initialized uniformly so that the sum of their counts is one, the idea being that generally only one new rule is required to parse the sentence, so a count of one is an appropriate weight for new rules. The new rules are added to the grammar, and the result trained using a modified version of I-O. The rule counts obtained from all preceding sentences are treated as fixed, and only the failed sentence is used for training data. On each iteration, the counts for the subject sentence are estimated and added to the frozen values, and then the probabilities are re-estimated from these counts. Although this is not guaranteed to reduce the cross-entropy of the subject sentence, it generally does. (It is guaranteed not to increase the cross-entropy of *all* sentences seen so far, but the cross-entropy of *this* sentence may increase.) This training is terminated when the cross-entropy gain is $< 10^{-5}$ bits.

The primary benefit of this training is to reduce the number of rules added to the grammar. Some generated rules may be useless because they require other rules that violate the constraints. Other rules quickly drop to zero probability and are eliminated. We hasten the process a good deal by discarding any rule whose count drops below 10^{-2}. This immediately eliminates all but the top 100 candidate rules (at most) for each non-terminal head.

A significant secondary benefit is to start the rules off with reasonable probabilities when the regular parsing and statistics gathering resume. As we argue later, initialization is important for good results from I-O, and this approach yields good quasi-initial values.

Using the sentence alone is a compromise made for performance reasons. Despite the constraints, the number of rules generated can be quite large, and parsing costs can quickly become astronomical. The reason is that the rules are all targeted exactly at the subject sentence, so worst-case behavior is almost invariably realized, at least until enough rules are eliminated.

4.2.1 Dynamic rule learning

The chief innovation in our approach is the dynamic character of rule acquisition. The usual approach can be stated in two lines of pseudo-code:

```
generate all possible rules;
initialize the rule probabilities randomly;
```

While details vary significantly, something like this was done in [5,19,24]. The first problem with this scheme is that it tends to generate a large number of rules, which, in turn, lead to excessively expensive parsing costs. The second is that the local optimum found by I-O will generally be poor. Our experience with I-O indicates that it is generally well-behaved, so the better the performance of the starting grammar, the better the performance after training to convergence. The flip side of this is that initializing the rule probabilities or counts becomes important. A moment's reflection shows that the very best initial probabilities would actually be the very best final probabilities, and clearly, if one had these, one wouldn't have to train with the I-O algorithm at all. Initialization would not be an issue if the local optima reached by the I-O algorithm were approximately equal in terms of final performance, but they are not. As we reported in [31], there are a large number of local minima even for grammars learned from small, artificially generated texts, and initialization alone alters both the final grammar reached and the performance of that grammar. For larger, more complex problems, one expects the penalty paid to be worse, and the number of local sub-optima to be greater. If anything, we are lucky that the I-O algorithm generally performs better given more plausible initial rule probabilities; there is no requirement that this should be so, but our work shows that, generally speaking, the better the initial values, the higher the final performance. To summarize: the batch approach combines the maximal training time (largest, most ambiguous set of rules and largest set of data), and the smallest chance of achieving the global or near-global optimum (because it lacks a good initialization mechanism). The dynamic approach addresses these problems fairly successfully.

```
generate all rules for sentence
eliminate rules that violate constraints

initialize counts for new rules

revise new rule probabilities using I-O,
     discarding little-used rules
```

Figure 4.2: Pseudo-code for rule generation, initialization, and selection.

Our own scheme begins with a seed grammar which may be empty. We begin by sorting the corpus so that sentences are presented from shortest to longest. Rules are suggested and added whenever a parse fails. As we process the corpus, we keep track of how often each rule has been used, and after suggesting/adding rules we re-estimate the rule probabilities from these usage statistics. This is effectively a single iteration of the Inside-Outside algorithm, spread over the course of the learning run.

We sketch the procedure for selecting and initializing rules in figure 4.2. This is the procedure invoked whenever a parse fails. We generate all rules that could be used in parsing the given sentence, subject to some intuitive linguistic constraints. We describe these constraints and the grammar formalism that allows us to generate rules below. New rules are initialized uniformly, so that the sum of their rule counts is one. We then train the new rules over the given sentence, as if we were carrying out the I-O algorithm. That is, we re-estimate the rule counts and rule probabilities, for new rules only. The rule counts for the rest of the grammar are frozen at the values they had when the sentence failed to parse. Note that this is not guaranteed to reduce the cross-entropy over successive iterations, although generally it does. The idea is to force the rule probability values to reflect the data.

4.2.2 The Formalism

As mentioned above, the space of all possible context-free grammars is much too large to search, and it includes grammars with such uninteresting features as unused non-terminals and chaining rules ($A \overset{*}{\rightarrow} B$ and $B \overset{*}{\rightarrow} A$). We begin constraining our solution by choosing a suitably restrictive formalism, which in our case is a variation on dependency grammars. [32] These provide us with both an easy means of inducing rules and useful restrictions on the non-terminals. Formally, our grammars are a subset of CFGs that use rules restricted to the form $\overline{X} \rightarrow \alpha X \beta$. Here X is a terminal, α and β are possibly empty strings of non-terminals. In our case, the terminals are the part-of-speech tags accompanying each word, and the words themselves are ignored. For each terminal in the language, there is exactly one corresponding non-terminal, which by convention is given the name of the terminal with an overlying bar. We call the unbarred term the head of the rule. Since every rule must have a terminal on its right-hand side, no chaining is possible. This guarantees that there is a finite number of rules from which a sentence could be generated, although this number may grow exponentially in the length of the sentence. The procedure for generating all rules is straightforward, being similar to bottom-up chart parsing.

Dependency grammars address a couple of issues that CNF grammars, the usual choice, leave open. Dependency grammars give both the number of non-terminals and in some sense the identity of each non-terminal. By identity we mean that the meaning of each NT is human-understandable, and there is a known relationship between terminals and non-terminals. Under CNF, the NTs are generally meaningless, and any NT may dominate any NT or terminal.

We exploit both the accessibility to humans and the imposed structure, in our use of a bias or "seed" grammar and in the constraints we apply to rule generation.

4.3 Rule Generation

As shown in figure 4.2, there are three basic steps involved in adding rules to a grammar: generating new rules, initializing the probabilities of those rules, and revising their probabilities. The process of generating rules is akin to bottom-up chart parsing, the difference being that instead of simply looking to see if there is a rule which can add new constituents higher up in the chart, a rule is added if it is not present. Exhaustive generation causes excessively expensive parsing, so we use constraints to restrict both what rules get generated, and how they are initialized. We turn to these next.

4.3.1 Rule Form Constraints

Our explicit constraints take the form of additional restrictions on the form of rules that are allowed. In essence, we want to bias the learner, using what we already know about which English parts of speech should be allowed to head phrases. For example, nouns should not appear in adjective phrases, but adjectives should be allowed to appear in noun phrases. We declare this information as a set of lists, one list for each non-terminal, which is considered the head. Each list enumerates the non-terminals that are allowed on the right-hand side of rules with the head on the left-hand side. The rule construction procedure then tests candidate rules to ensure that no forbidden part of speech appears on the right hand side of a rule.

Explicit constraints are helpful in biasing the learner towards human-oriented grammars. They push the system towards rules which agree with linguistic intuition, and hence towards rules which linguists might recognize, understand, and regard as suitable for natural language understanding. Not coincidentally, we believe such rules might also be helpful from a language modelling point of view, allowing a low cross entropy.

Such constraints are fairly easy to write down, they capture human intuition about the structure of English grammar that would be hard to learn otherwise, and they provide a solution to the problem of "memorizing" grammars. Memorizing grammars come about because I-O is very good at finding the rule or rules which minimize the resulting net cross entropy, when the search space is kept small. Unfortunately, it is so good that it can often fit the training data exactly by preferring one rule per sentence, where that rule captures the sentence's entire tag sequence. The trouble with this is that no generalization takes place, so the grammar performs poorly on test data. In related work on artificial corpora [31], we have shown that rule form constraints are effective in preventing the learner from memorizing. English text shows much more variety than our artificial data, with the result that the constraints must be much weaker. This limits their usefulness somewhat, with the result that we have developed more flexible preferences, to be described later.

4.3.2 Length Preferences

Length preferences were initially introduced to counteract the problem of memorizing grammars. The idea was to force the learned rules to be shorter, so that more generalization would have to take place. Unfortunately, memorizing degrades gracefully in the face of length restrictions, so long sentences would only be broken into two or three rules, and generalization was still poor.

Although a failure at preventing memorizing, rule length limits are effective at reducing the number of rules generated. The number of possible rules increases exponentially with the permitted length, but we expect only few rules of length 7 or longer to be useful. Since we expect little reward to be gained from permitting long rules, and the parsing cost can be severe, it makes sense to prefer shorter rules.

For the basic SINGER system, we arrange this by imposing a flexible limit on the length of rules which are generated. We carry out the rule generation procedure allowing only rules below the limit; if this fails to produce a parse, we extend the limit by one, and repeat. The initial limit is 5, and almost always ($> 90\%$) results in a parse.

4.4 The Bias Grammar

The bias grammar consists of about 60 rules with probabilities written in a general PCFG format, which we mechanically translate into a dependency grammar. In this form the grammar has about 3500 rules. We train this grammar with one pass of I-O and discard the low and zero-count rules, leaving about 800. The idea here is just to cut down the grammar to a reasonable backbone with reasonable probabilities. Coverage of the resulting grammar is around 52% and cross-entropy is poor. However it forms a reasonable basis, and helps both by speeding learning and improving the final performance.

A bias grammar such as ours may seem an obvious move to make, but it is not always easy to build. It is harder to write one down if one chooses an unorthodox formalism such as link grammars [33]. Other authors have chosen to work with other formalisms such as CNF [24] or LR grammars [34] which, like ours, can be mechanically generated from an ordinary PCFG. The problem with these is it becomes hard to supply any linguistically motivated constraints to the rule generation process, because the non-terminals in the resulting grammars are opaque to humans.

4.5 Experiments with the basic Singer and Extensions

In table 4.3 we list the results of experiments run with the basic system, as described above. The training data is a 300,000 word subset of the Brown Corpus of Tagged English [35], a collection of articles, excerpts from books, and the like, with each word tagged with its part of speech. We filter out sentences with odd words (chiefly foreign words, headlines, or parentheses), and sentences more than 23 words in length. We collapse the 400-odd tags of the raw corpus down to 20. 10,000 words are reserved for testing.

The filtering and tag set collapsing were done to ease the learning task while Singer was under development. While newer versions of our system can handle

	cross entropy	coverage %
basic SINGER	2.8274	97.8
trigram	2.7714	100.0
prev + S = in bias	2.8350	98.2
prev + S = rule gen	2.8035	99.0
prev + new constraints	2.8005	99.3
prev + Q =	2.7978	99.3
no Q=, averaged	2.7456	99.0

Figure 4.3: Performance results

much of the remaining data, we use only the filtered dataset for the sake of consistent comparisons across all experiments.

We begin our clinical history of SINGER with the results of the basic system described up to now. While this is not really the first learner we built, it was the first to successfully process the Brown corpus. The results, as shown in fig. 4.3, are only fair. Cross entropy is quite a bit higher than for trigrams, and coverage is only fair.

Below these basic results we begin listing results for experiments with modified versions of SINGER , which we explain below. Some of these modifications are motivated by obvious shortcomings in the basic scheme, but others, such as bias grammar revisions, are corrections to problems uncovered by analysis. We have a fairly simple tool which computes the probabilities of a sentence according to both the grammar and trigram models, and prints out the sentence and probabilities if the grammar model performs significantly worse. (These sentences are drawn only from training data, naturally.) It also keeps statistics on the frequency with which non-terminals show up in bad sentences and overall. If a particular linguistic phenomenon is poorly handled by the grammar, it often stands out, and we give examples in the next section.

4.5.1 Extending the Formalism Part I—embedded sentences

Analysis of the basic SINGER revealed that it performed poorly in sentences with embedded sentence clauses. For our purposes, these are generally relative clauses, as in

`The man` *that wore funny hats* `also rode a unicycle.`

This failure did not come as much of a surprise, since dependency grammars are too weak to capture a number of linguistic phenomena, embedded sentences being prominent among them.

One possible remedy for this sort of problem would be to add a new non-terminal, and create the appropriate rules by hand, but this misses the learning spirit of our system. More to the point, if the bias grammar makes use of a special non-terminal to represent an embedded sentence, but the rule generation mechanism is unable to generate rules headed by this non-terminal, then performance suffers. The learner cannot generate the appropriate rule when presented with a sentence containing a novel embedded sentence, so it will generate inapropriate rules, not headed by the reserved non-terminal, and these rules will compete with better rules in other contexts.

Extending the rule generator requires modifying code, but the new algorithm extends the old one in a simple and straightforward manner. The modified generator checks the right-hand side of generated rules to see if they have an NP preceding a VP. If so, then the rule may be headed by $S=$, the new non-terminal, but not by any other. Line 3 of figure 4.3 reports the results of modifying the basic scheme by rewriting the bias grammar to make use of an embedded-sentence non-terminal. The result is a slight loss in performance, which shows that we are not covertly improving performance through tweaking the bias alone. The next line reports the results when the system also has a rule generator that uses a simple template to generate embedded-sentence rules. With the addition of this simple learning scheme, the system can handle embedded sentences and performs significantly better.

4.5.2 Extending the Rule Preferences

A second weakness with the basic system lies with the all-or-nothing rule form constraints. The complexity of English, the presence of exceptional or even agrammatical sentences in the data, and poor choices for tags lead to constraints that must permit almost anything. To restore their usefulness, we introduced two small weights associated with each constraint. The first of these indicated the "welcomeness" of the subordinate NT on the rhs of the dominant NT. This aimed at capturing the same phenomena as our previous constraints, only in a more flexible way. The second weight indicated how likely it was that the subordinate NT appeared before the terminal or after. This allowed us to indicate that determiners precede nouns in English, but hardly ever follow them.

The effect of these weights is to alter the initial values of rule probabilities, and so to influence which rules get selected. Since I-O is still used to revise the rule probabilities before selection, a rule which ranks low according to the constraints may still be selected.

Let r be a rule, $\mathcal{H}$ the head of the rule, $\mathcal{P}(r)$ the set of non-terminals which precede the terminal in r's rhs, $\mathcal{F}(r)$ the set of non-terminals which follow. We call the "welcomeness" weight w_1 and the preceding likelihood weight w_2. The initial count of a rule r is

$$\begin{aligned}\prod_{NT\in\mathcal{P}(r)} & w_1(NT,\mathcal{H}(r))w_2(NT,\mathcal{H}(r)) \times \\ & \prod_{NT\in\mathcal{F}(r)} w_1(NT,\mathcal{H}(r))(1.0 - w_2(NT,\mathcal{H}(r))) \end{aligned} \tag{4.1}$$

Rules are also penalized for length instead of being initialized uniformly. After having the counts are assigned above, each is divided by $10^{length(r)-1.0}$.

4.5.3 Extending the Formalism Part II—question sentences

Our analysis program shows that question sentences often have sub-par probability, unless given special treatment. For this reason we introduce the second and

$\overline{\mathsf{PRON}}$ $\overline{\mathsf{NN}}$ $\overline{\mathsf{TO}}$ $\overline{\mathsf{VBG}}$ $\overline{\mathsf{VBN}}$ $\overline{\mathsf{AUXG}}$ $\overline{\mathsf{THAT}}$ $\mathsf{S}=$ $\overline{\mathsf{DET}}$ $\overline{\mathsf{WDT}}$ $\overline{\mathsf{ADJ}}$

Figure 4.4: Noun-type non-terminals permitted in $\mathsf{Q}=$ rules.

$\overline{\mathsf{ADJ}}$ $\overline{\mathsf{ADV}}$ $\overline{\mathsf{AUX}}$ $\overline{\mathsf{AUXG}}$ $\overline{\mathsf{NN}}$ $\overline{\mathsf{TO}}$ $\overline{\mathsf{VB}}$ $\overline{\mathsf{VBG}}$ $\overline{\mathsf{VBN}}$

Figure 4.5: Verb complements required in $\mathsf{Q}=$ rules.

last non-terminal which does not obey the usual dependency grammar schema. This non-terminal is named $\mathsf{Q}=$, as it heads sentence-level phrases which are in question form. As with $\mathsf{S}=$ rules, these are required to follow a simple template, and rules matching the template are not permitted to be headed by any other non-terminal. $\mathsf{Q}=$ rules have an optional wh-word in initial position, either $\overline{\mathsf{WDT}}$ or $\overline{\mathsf{WRB}}$. This is followed by a mandatory AUX, *not* an $\overline{\mathsf{AUX}}$, and a noun-type non-terminal which may be any of the ones listed in figure 4.5.3. Finally, there is a mandatory verb complement phrase, which may be any of the ones listed in figure 4.5.3. As figure 4.3 shows, $\mathsf{Q}=$ buys a slight improvement in performance overall, about .0025 bits/word. We retained the rules and the rule generation mechanism largely on the theory that they were more correct than what the grammar was forced to do without such rules. Without a special head, questions would have to be handled by $\overline{\mathsf{AUX}}$, possibly with some assistance from the wh-words $\overline{\mathsf{WDT}}$ and $\overline{\mathsf{WRB}}$. Unfortunately, if $\overline{\mathsf{AUX}}$ heads rules in question form, it licenses sentences such as

```
* The man is he going to come over today?
```

Since questions are not that uncommon, this overgeneration will have a fairly high probability, which means that ordinary sentences will be penalized. The reason we do not see more gain is probably because the impoverished tag set is covering it up. Our tags do not distinguish between "is" and "has", nor between subject and object pronouns, so the $\overline{\mathsf{AUX}}$ phrase in the legitimate sentence

```
Coach has him running laps.
```

is indistinguishable from the question

```
Is he running laps?
```

A richer tag set would have both of these distinctions and so would show more of a difference between SINGER with and without the $\mathsf{Q} =$ rules.

4.5.4 Averaged Results

Figure 4.6 reports the average performance of SINGER across different orderings of the same data. An astute reader will have noticed that SINGER's performance may vary depending on the order in which sentences are presented, and so an average result is a more accurate measure. Consider two sentences of a given length, say A and B. Suppose that for A our learner will suggest rules that are also sufficient to parse B, but for B the learner would suggest a different set. What rules the grammar ends up with depends on which sentence occurs first. In figure 4.6 we give average performance (the λ) of the basic induction scheme, and the individual performance on 10 learning runs over randomized orderings of the corpus. (The shortest-first ordering was still, of course, preserved.) In figure 4.7 we give the cross entropy results for the same experiment.

In all cases, we evaluate performance immediately after learning and after 10 iterations of training by the Inside-Outside algorithm, which in our experience is enough to reveal any systematic differences among grammars, although performance may continue to climb with further iterations. For this experiment $\mathsf{Q} =$ rules were not used. Figures 4.6 and 4.7 should give one a feel for the behavior of our grammars through the training process. In general, performance immediately following learning is poor, and increases most rapidly for the first couple of training iterations. Performance curves occasionally cross, but there are no wild differences in the shape of the curves, nor are there dips indicating overtraining. Variance at the end of training was $8.69 * 10^{-4}$ for the lambda measure, and $1.89 * 10^{-5}$ for cross entropy. This is more variation in performance than we would like. It seems largely connected with the coverage of the learned grammar.

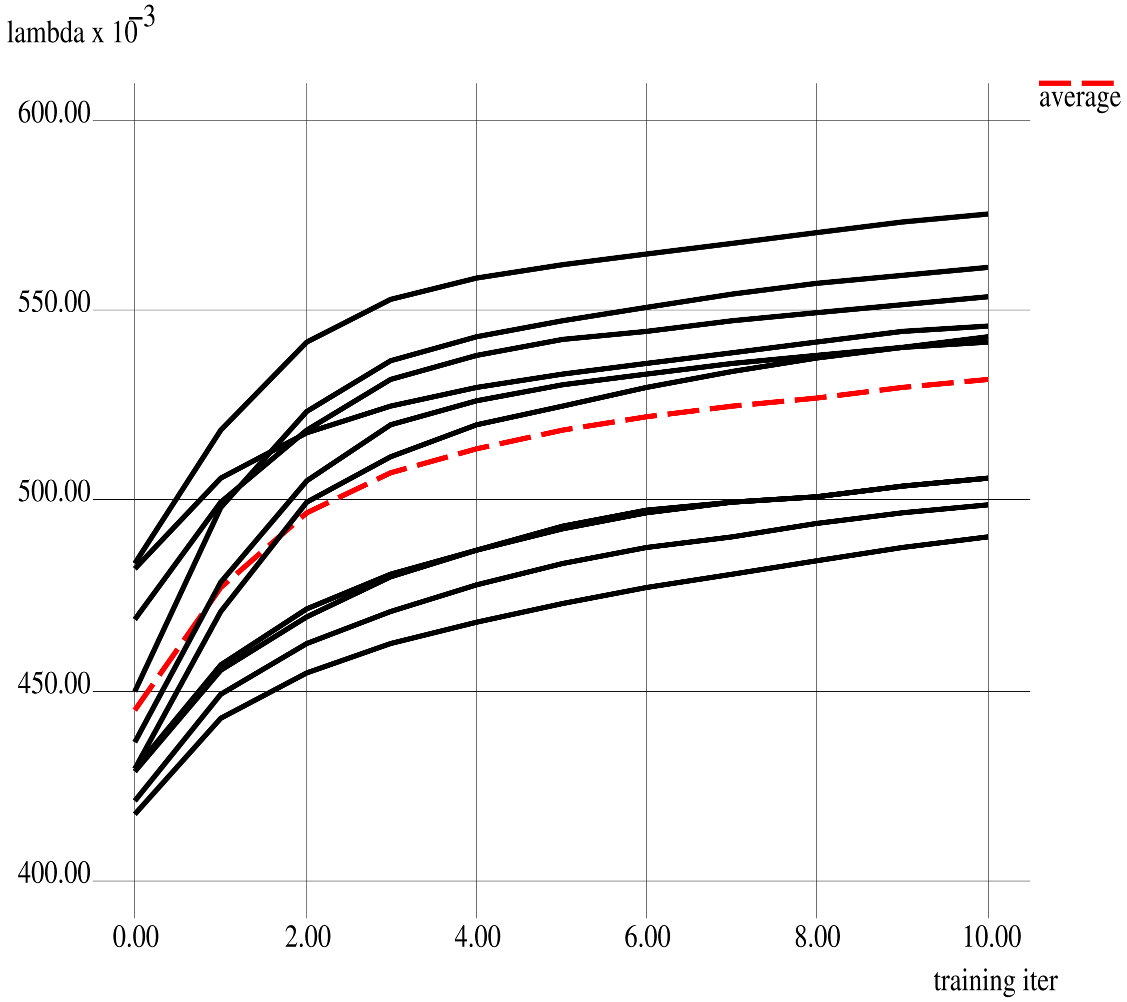

Figure 4.6: λ performance of basic learning, showing individual runs from randomized orderings and overall average.

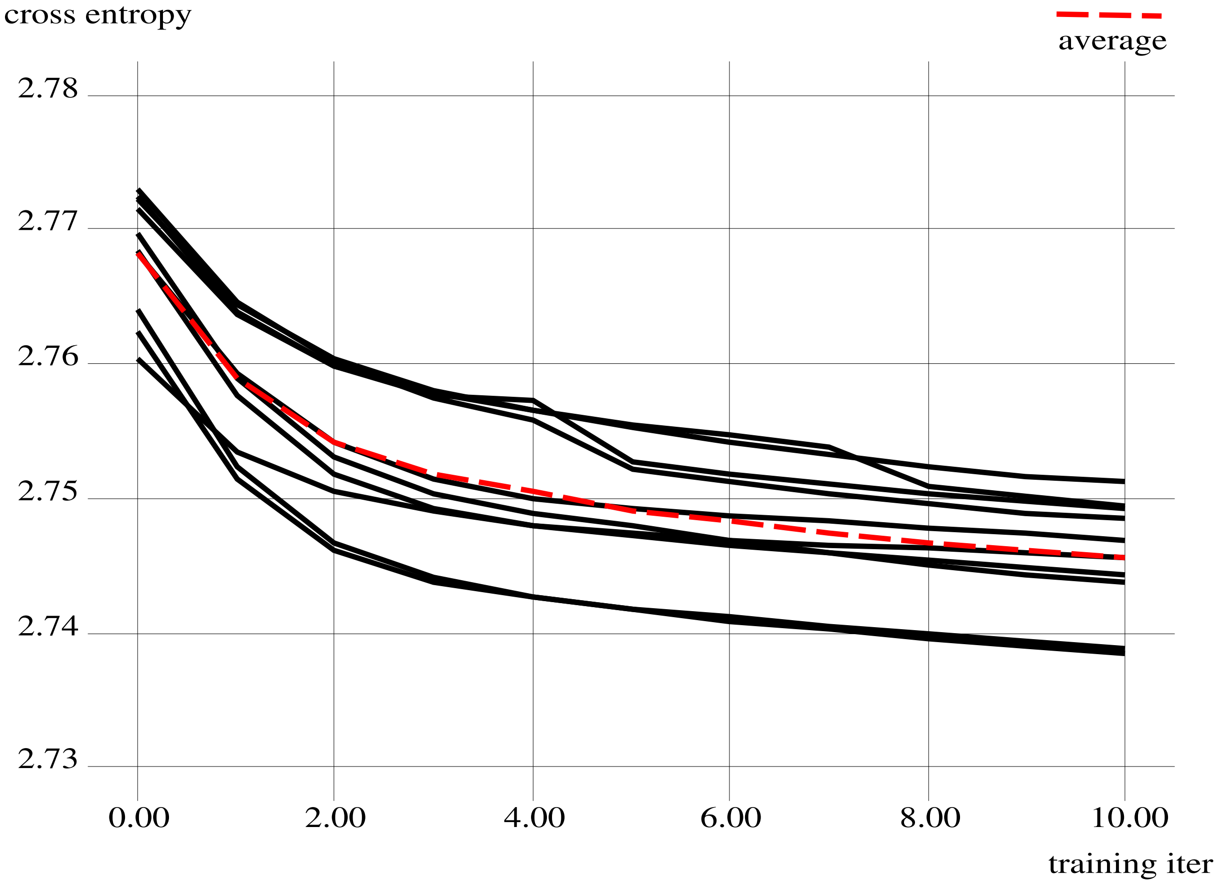

Figure 4.7: Cross entropy performance of basic learning, showing individual runs from randomized orderings and overall average.

Coverage ranges from 98.68 to 99.23% of total words parsed for these grammars, and is almost perfectly correlated to increased performance.

Chapter 5

Stand-alone Learning Methods

Up to this point, the strategies we have reported have all been part of an integrated system, not easily decomposable. Here we report two learning strategies which are separable, the first relying only on having a scheme for learning PCFGs, and the second relying only on having an initial PCFG, obtained by any means whatsoever.

5.1 Generating and Combining Multiple Grammars

This section reports the results of experiments based on adapting an idea from stacked regression and extending it in a domain-dependent fashion to improve the performance of an existing grammar learner. Wolpert [36] lays out a very general theoretical framework for stacked regression that can be reified in a number of quite different learning algorithms. As the framework is based on supervised learning from noise-free data, we can only borrow ideas, without the theoretical support Wolpert develops. Here we focus on the idea of combining information from more than one generalizer, which in our case are PCFGs. Wolpert points out that if one has gone to the trouble to build several generalizers from a single data set, then it is almost always better to combine the information from them, rather than, say, choosing one, as has been standard practice with cross-validation. In the stacked regression framework this combination process may take almost any form, some of which require an error signal (i.e., supervised learning), and are therefore infeasible in our domain. One of the more obvious and feasible ideas for combining generalizers is simply to average across them, and other researchers report good results from this [37]. We can do slightly better than that, by choosing optimal weights to smooth over our grammars.

To be more concrete, our first suggestion is to generate several PCFGs and smooth them together. For a set of grammars G_i, we choose some positive weights γ_i, such that $\sum_i \gamma_i = 1.0$ and we define the smoothed probability of a sentence s to be

$$P(s) = \sum_i \gamma_i P_{G_i}(s) \tag{5.1}$$

We can choose the $gamma_i$ using the forward-backward algorithm [38] and some training data reserved prior to learning the grammars.

The reader may have noticed that this suggestion contains nothing that is domain-specific or, for that matter, original. We report results from this strategy as a baseline for comparison with the domain-specific strategies we develop below. Sadly, performance using this suggestion does not yield performance improvements, perhaps because the assumptions of stacked regression do not hold.

Our real interest was not in the stock approach described above, but rather in a domain-dependent method of combining PCFGs, essentially by "adding" them together to produce a single grammar. The resulting tighter coupling, together with an appropriate training algorithm, results in a substantial performance increase, as we document in our second experiment.

Finally, in our last experiment we vary the way in which the grammars are generated, in order to find the best method to use with our "adding" procedure. Rather than varying an algorithmic parameter, we vary the data available to the learner. This bears some similarity to "leave-one-out" cross-validation, which builds multiple generalizers this way for the purpose of evaluating learner performance. Unlike cross-validation, in which the aim is to select the best generalizer, our aim is to *use* all of the acquired generalizers, so we find a rather different

	lambda	lambda variance	cross entropy	coverage %
basic SINGER	0.532	0.000869	2.7456	99.0
smoothed	0.529	0.0000842	2.7476	99.4
added, 10 segs, l-o-o	0.618	0.0000795	2.7286	99.4
added, 10 segs, l-o-i	0.658	0.0007221	2.7175	99.6
added, 100 segs, l-o-i	0.721	0.0000511	2.7017	99.7
added, 25k segs, l-o-i	0.711	0.0	2.7029	99.7

Figure 5.1: Comparison of smoothing versus unsmoothed grammars

allocation of the data is more suitable.

In the following three sections we give results for three experiments, evaluating the generic smoothing scheme, our domain dependent scheme, and "leave-one-in" versus "leave-one-out" data allocations.

Experiment 1

Figure 5.1 summarizes the results for all our experiments. All results are averages from 10 runs over randomized orderings, except for the last (25,0000 segment) experiment. The entry headed "smoothed" gives the graph of results for our "replication" of ordinary stacked regression, or at least our unsupervised learning approximation to it. For this experiment we divided the corpus into 10 equal sized pieces. Nine pieces were used for grammar learning, and the remaining piece for testing. The different grammars were created by varying the data available to each learning run. In each case, one of the 9 designated learning segments was held out, so the learner saw only 80% of the data. Each grammar was trained once, using all 9 learning segments, and finally smoothing parameters were chosen using the forward-backward algorithm on the reserved segment. Grammars were given only one training pass each in order to allocate training time comparable to 10 passes with a single grammar.

The first two lines of the table in figure 5.1 show that the smoothed grammars do not perform better than the "basic" scheme of learning over the whole corpus all at once. Actually, they perform somewhat worse, but not significantly so.

Experiment 2

As we said earlier, our real hope for performance improvement lay in the idea of "adding" grammars together, so that rules distributed among the different grammars can cooperate in parsing a sentence. The technique relies on information gleaned by the Inside-Outside algorithm, which provides a count for each rule, indicating how many times it is used. I-O works by alternately using the rule probabilities and the training data to estimate the counts, and then re-estimating the probabilities from the counts. For our purposes, the key property of the counts is that they can act as a common currency among grammars. Rule probabilities, which are the most obvious alternative, lack this property of interchangeability, making them unsuitable for our purpose.

We add grammars together, then, by adding the rule counts. If a rule is not present in any grammar, its count is implicitly zero. Probabilities for the new grammar are calculated from the counts, using the same equation that the Inside-Outside algorithm relies on. For a given non-terminal, NT, and some string of terminals and non-terminals, α, the probability of the rule $NT \rightarrow \alpha$ can be calculated from the counts as follows.

$$P(NT \rightarrow \alpha) = count(NT \rightarrow \alpha) / \sum_i count(NT \rightarrow \alpha) \tag{5.2}$$

In this experiment we added together the 9 untrained grammars generated in experiment 1, and then trained them 10 times. Results are shown in figure 5.1 under the entry "added, 10 segs, l-o-o", for adding the grammars, splitting the data into 10 segments, and "leave-one-out" rule induction. This penalizes the summing technique, as it has no chance to learn on 10% of the data, but as the table shows, performance is still superior.

Experiment 3

In this experiment we alter the way we allocate the training data to learn the varying grammars that we subsequently add together. While the "leave-one-out" strategy of learning has a long and honorable history in machine learning, and it is a very natural starting point, it does not seem the most promising for our problem. It is computationally expensive and it does not scale well, as increasing the segmentation, i.e., the number of ways one divides the data, causes the grammars to become more and more alike. In the limit, where the number of segments equals the number of sentences, the grammars are likely to be identical or nearly so, since they will have learned rules on the same failing sentences. Thus, very high segmentation is unattractive for a leave-one-out strategy with our kind of learning.

Instead we take the opposite approach to leave-one-out, and keep only one segment for learning, rather than all but one. As before, we iteratively train the grammars only after they are added, as it is more efficient to do so after, when a single pass through the data trains all new rules. Under this scheme, which we call "leave-one-in", increasing the segmentation will tend to increase the variation among grammars. Moreover, the grammars are cheaper to learn, as fewer sentences are processed for each. Lastly, under leave-one-in, more sentences contribute rules at a given segmentation number; in the limit, when one increases the segmentation to the corpus size, all sentences get a chance to contribute rules. In contrast, under leave-one-out some sentences may never trigger learning, and so will never suggest rules, under *any* segmentation. Figure 5.2 shows results of leave-one-in learning at various segmentation settings, up to the limit case of segmenting the corpus into one-sentence chunks. As the figure shows, performance does not increase up to the limit case, but appears to level off around a segmentation of 100.

The dominant effect of segmentation is to increase the number of learning events, as shown in figure 5.3. The table reports the number of times a parse failure lead to successful learning, and the size of the grammar both immediately after rule acquisition and after 10 iterations of I-O. These results, like the previous are averages from 10 runs for each segmentation setting, except the limit case of 25k segments, which has no variance. Results are rounded to 1.0. The table indicates that, in general, the more often the learner learns new rules, the better its final performance is. Improvement levels out because eventually the learner is

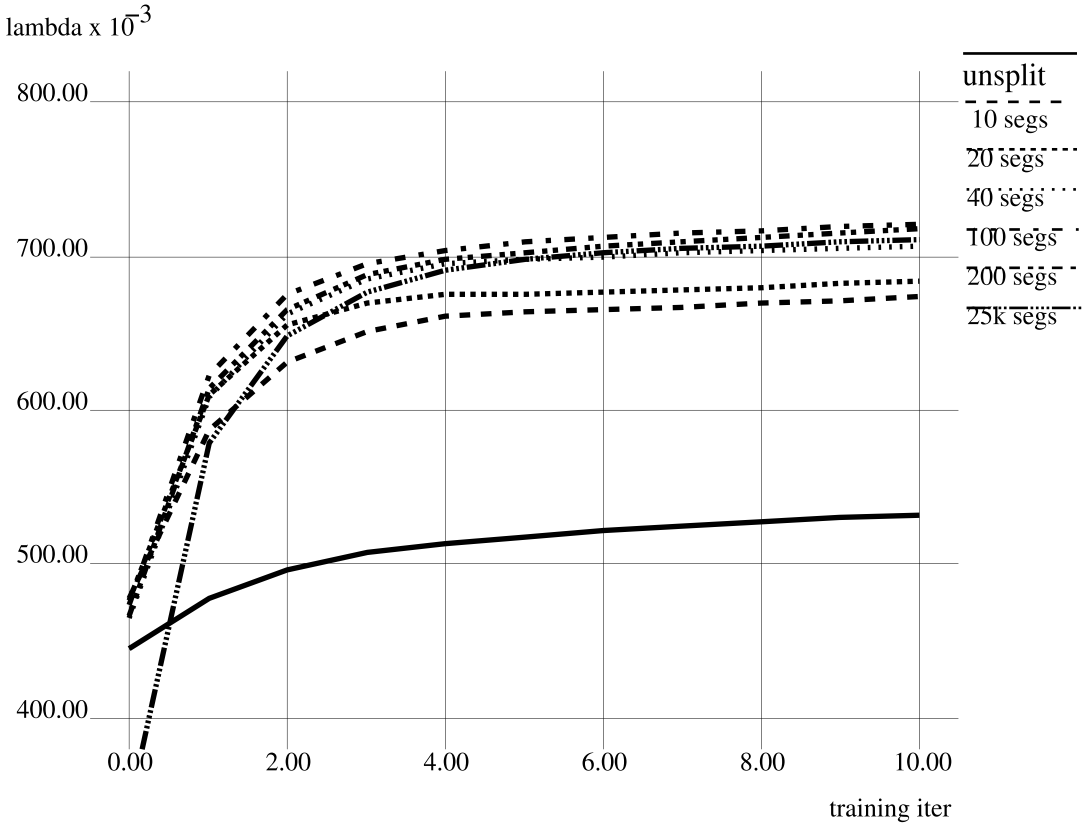

Figure 5.2: Performance of leave-one-in learning, for various segmentation values.

segmentation	learning events	grammar size	
		pre-training	post-training
none	297	1815	1495
15 segments	1961	4319	3213
100 segments	4824	7097	4386
25,000 segments	9825	7300	4694

Figure 5.3: The effect of segmentation on the number of rules acquired for leave-one-learning.

not learning rules, but re-learning the same rules, perhaps on the same sentence appearing in different segments. Thus, the grammar grows only slightly when the segmentation is increased from 100 to 25,000.

Summary

In this section, we have presented positive results from experiments in borrowing an idea from stacked regression and adapting it to our domain. Our experiments do not show promising results from generic smoothing over multiple grammars, but merging them provides the synergy we were hoping for. Grammar merging relies on variety among the constituent grammars, which implies that "leave-one-out" learning offers only limited opportunity for improved performance. For our application, "leave-one-in" is both computationally cheaper and more thorough in extracting information, as the learning is error-driven.

One issue we have not discussed so far is the relative CPU cost of the various schemes. The current version of the basic learning scheme takes about 7 hours for rule acquisition and 1.5 hours for each training iteration. Since rule acquisition involves the same parsing as training, this means that about 5.5 hours are spent purely learning rules. The bad news is that for small segmentation levels, up to around 15, say, the increase in CPU cost is linear, i.e., it takes about 10 times longer to learn the rules when the segmentation is 10, whether the learning is leave-one-out or leave-one-in. (Training the rule probabilities remains about the same, as segmentation has no effect on training. The only increase in time comes from the increase in grammar size that comes with increased segmentation.) The reason is that rule learning is error-based, and for small segmentation levels, the error rate per segment does not significantly differ from the error rate for the basic, unsegmented learner. At high segmentation levels, (around 100 segments and up), leave-one-out and leave-one-in behave differently. Leave-one-out continues to climb linearly in cost, as each learning pass comes closer to processing all of the data. For leave-one-in learning, the error rate per segment begins to drop, so performance cost likewise drops to less than linear. Note, also, that leave-one-out learning parses the entire corpus approximately $N-1$ times at a segmentation of N, whereas leave-one-in parses it only once.

Segmenting the data to the point where each segment has only one sentence suggests an entirely different way of thinking about this learning strategy. Rather than imagining it as a process taking place in parallel, one machine per segment, it can be thought of as sequential. Instead of adding new rules to the bias grammar, as is done in the basic SINGER, the new rules are kept in a separate grammar. The rule count information for parsable sentences is likewise added to this learned grammar. In other words, no incremental development of the grammar is allowed to take place. Incremental development leads to much cheaper learning, in terms of cpu hours, but it clearly costs in the final performance of the grammar. Hence, splitting up the learning and then combining the grammars represents a trade-off between time spent learning and performance level reached. Clearly, as long as machines are available to permit the learning to take place in parallel, the trade-off is an attractive one.

In our own environment, we have all the data available in batch form, and enough computers to handle up to about 100 segments in parallel. This allows for very fast experimentation. In an online environment, one would not expect to have either these resources, or all the data available at once. Our results suggest a strategy such as the following would be effective even in this more constrained environment. Incoming data should be processed in chunks of a few hundred sentences. (For our data set, a segmentation of 100 means each learner sees about 250 sentences.) Sort the sentences, and use the basic learner and the bias grammar to acquire new rules. The learned grammar, to which the new rules are added, is kept separate from the bias grammar. When rules are acquired, they are always added to the new grammar. The rest of rule selection operates as before, with the learned grammar and the new rules being revised with I-O, and so on.

This strategy allows some incremental development from the chunking; it puts all rules in a single grammar to reap the benefits of cooperation; and it maintains a relatively high error-rate to drive continued learning. While overall the learning must, inevitably, be slower under this regime, only a few thousand sentences should suffice to reach reasonable performance, and the scheme has the advantage of accomodating arbitrarily large data sets.

Chapter 6

Context-Sensitive Learning

In this section we investigate a scheme for gathering and using context sensitive statistics with our context-free grammars. The idea is to extend a standard PCFG by replacing the probability of each context-free rule with a set of probabilities, one for each non-terminal used by our grammar. Each of these new probabilities will reflect the probability of the rule occurring in a particular context, which in our case is simply the head of the parent rule. For example in figure 6.1, the rule $\overline{\mathsf{vbg}}\rightarrow\mathsf{adv}\ \mathsf{vbg}$ has the non-terminal $\overline{\mathsf{n}}$ as its parent. Formally, we write

$$P(N^i\rightarrow\alpha^j \mid \rho(N^i) = N^s) \tag{6.1}$$

where $\rho(x)$ is the non-terminal that immediately dominates x — its *parent*. We refer to N^i as the *child*. Continuing our example in Figure 6.1, we would require

$$P(\overline{\mathsf{vbg}}\rightarrow\mathsf{adv}\ \mathsf{vbg} \mid \rho(\overline{\mathsf{vbg}}) = \overline{\mathsf{n}})$$

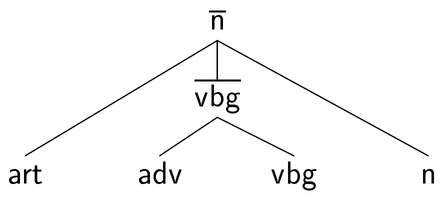

Figure 6.1: Application of a rule within a $\overline{\mathsf{n}}$

This is the probability that we expand $\overline{\mathsf{vbg}}$, (a verb phrase headed by an "ing" verb) as an adverb followed by the verb, given that its parent is $\overline{\mathsf{n}}$, a noun phrase. Such a situation might occur in the parse of a phrase like "the slowly dripping faucet."

Although the statistics characterized by equation 6.1 include context information, we have not adopted the standard notation for context-sensitive grammars, nor, in fact, do these grammars move beyond what is expressible by a PCFG. The pseudo PCSG can be expanded into a PCFG roughly as follows. For each parent-child pair, begin by creating a new non-terminal to represent the chosen pair. For any rule with the parent as its head, substitute the new non-terminal for each occurrence of the child. For each rule headed by the child, add a new rule headed by the new non-terminal. With some trivial math, one can compute new probabilities for all the rules in the expanded grammar. We return to this process later.

Although technically we have not moved beyond PCFGs, clearly our formalism has something of a context-sensitive flavor. We are gathering context-*sensitive* statistics for a context-*free* grammar, so we compromise by calling the combination "pseudo context-sensitive." Clearly, the extra information provided by context allows us a good deal more flexibility in assigning probabilities to parses. For example, while the above rule for $\overline{\mathsf{vbg}}$ would be a not-uncommon one to find as part of a noun-phrase, consider instead the following rule for $\overline{\mathsf{vbg}}$

$$\overline{\mathsf{vbg}} \rightarrow \mathsf{vbg}\ \overline{\mathsf{n}}$$

This rule would be used in "Alice was planting the flowers." While this rule is a common one at the sentence level, at the noun-phrase level it would be quite uncommon. Our new probabilities would allow the system to capture this regularity. As just noted, this regularity could also be captured through the use of a different non-terminal dominating the gerund. However, our new scheme allows us to find and capture such regularities automatically.

To use this model requires first that one can efficiently estimate the probabilities specified in Equation 6.1 and second, that given these probabilities one can efficiently calculate the probability of a parse. The second of these is reasonably straight forward, and we leave it as an exercise for the reader. The former we cover in the next section.

6.0.1 Calculating the Probabilities

In the generic inside-outside algorithm one re-estimates the probability of an event e based on its frequency in a training corpus. Our events are rule occurrences, or uses, in sentence parses and our training data is a set of sentences. Typically each sentence in the corpus has many parses, and it is possible that e occurs zero, one, or more times in any particular parse. Since the parses, and hence the events, are not directly observable, one must estimate these occurrences from the parse probabilities. An e-count for a sentence is estimated as the sum of the probability of each parse in which e occurs, given the sentence. Our goal in this section is to come up with the equations for this sum. We show the equations for the case where the grammar is in Chomsky-normal form.

We want to count the number of times an event occurs ($C(e)$).

$$\begin{aligned} C(e) &= \sum_{o \,\epsilon\, \mathsf{occurrences}(e)} P(o \mid w_{1,n}) \\ &= \frac{1}{P(w_{1,n})} \sum_{o \,\epsilon\, \mathsf{occurrences}(e)} P(o, w_{1,n}) \end{aligned}$$

More specifically, we wish to count the occurrence of the rule $N^i \rightarrow N^p\ N^q$ in the context of $N^i\ N^s$. The sum over all possible ways this event could occur includes (1) the positions N^i, N^p, and N^q in the parse, (2) the position of N^s in the parse, and (3) the rule that relates N^s to N^i. These are shown in Figure 6.2. Note that for part (3) there are two cases we need to consider, where N^i occurs as the left and right constituents of the higher level rule. Figure 6.2 only shows the second of these. These two possibilities correspond to the two sums in the following equations.

$$C(N^i \rightarrow N^p N^q, \rho(N^i) = N^s)$$

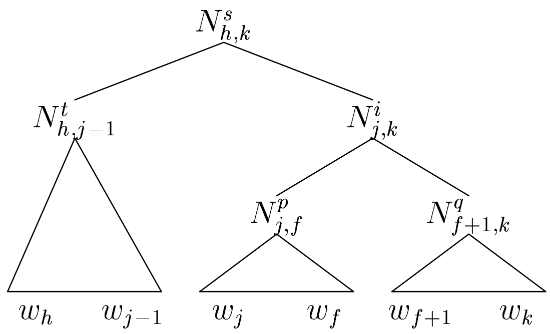

Figure 6.2: The situations for $N^i \rightarrow N^p\ N^q$ under N^s

$$
\begin{aligned}
&= \frac{1}{P(v_{1,n})} \sum_{j,k,t,h,f} P(N^i_{j,k}, N^p_{j,f}, N^q_{f+1,k}, N^s_{h,k}, N^t_{h,j-1}, v_{1,n}) \\
&\qquad + P(N^i_{j,k}, N^p_{j,f}, N^q_{f+1,k}, N^s_{j,h}, N^t_{k+1,h}, v_{1,n}) \qquad (6.2)\\
&= \frac{1}{P(v_{1,n})} \sum_{j,k,t,h,f} P(v_{1,h-1}, N^s_{h,k}, v_{k+1,n}) \\
&\qquad P(N^i_{j,k}, N^t_{h,j-1} \mid v_{1,h-1}, N^s_{h,k}, v_{k+1,n}) \\
&\qquad P(v_{h,j-1} \mid N^i_{j,k}, N^t_{h,j-1}, v_{1,h-1}, N^s_{h,k}, v_{k+1,n}) \\
&\qquad P(N^p_{j,f}, N^q_{f+1,k} \mid v_{1,j-1}, N^i_{j,k}, N^t_{h,j-1}, N^s_{h,k}, v_{k+1,n}) \\
&\qquad P(v_{j,f} \mid N^p_{j,f}, N^q_{f+1,k}, v_{1,j-1}, N^i_{j,k}, N^t_{h,j-1}, N^s_{h,k}, v_{k+1,n}) \\
&\qquad P(v_{f+1,k} \mid N^p_{j,f}, N^q_{f+1,k}, v_{1,j-1}, N^i_{j,k}, N^t_{h,j-1}, N^s_{h,k}, v_{f+1,n}) \\
&\qquad + P(v_{1,j-1}, N^s_{j,h}, v_{h+1,n}) \\
&\qquad P(N^i_{j,k}, N^t_{k+1,h} \mid v_{1,j-1}, N^s_{j,h}, v_{h+1,n}) \\
&\qquad P(v_{k+1,h} \mid N^i_{j,k}, N^t_{k+1,h}, v_{1,j-1}, N^s_{j,h}, v_{h+1,n}) \\
&\qquad P(N^p_{j,f}, N^q_{f+1,k} \mid N^i_{j,k}, N^t_{k+1,h}, v_{1,j-1}, N^s_{j,h}, v_{k+1,n}) \\
&\qquad P(v_{j,f} \mid N^p_{j,f}, N^q_{f+1,k}, N^i_{j,k}, N^t_{k+1,h}, v_{1,j-1}, N^s_{j,h}, v_{k+1,n}) \\
&\qquad P(v_{f+1,k} \mid N^p_{j,f}, N^q_{f+1,k}, N^i_{j,k}, N^t_{k+1,h}, v_{1,f}, N^s_{j,h}, v_{k+1,n}) \\
&= \frac{1}{P(v_{1,n})} \sum_{j,k,t,h,f} P(v_{1,h-1}, N^s_{h,k}, v_{k+1,n}) P(N^i_{j,k}, N^t_{h,j-1} \mid N^s_{h,k}) \\
&\qquad P(v_{h,j-1} \mid N^t_{h,j-1}, N^s_{h,k}) P(N^p_{j,f}, N^q_{f+1,k} \mid N^i_{j,k}, N^s_{h,k}) \\
&\qquad P(v_{j,f} \mid N^p_{j,f}, N^i_{j,k}) P(v_{f+1,k} \mid N^q_{f+1,k}, N^i_{j,k})
\end{aligned}
$$

$$
\begin{aligned}
&+ P(v_{1,j-1}, N^s_{j,h}, v_{h+1,n}) P(N^i_{j,k}, N^t_{k+1,h} \mid N^s_{j,h}) \\
&\quad P(v_{k+1,h} \mid N^t_{k+1,h}, N^s_{j,h}) P(N^p_{j,f}, N^q_{f+1,k} \mid N^i_{j,k}, N^s_{j,h}) \\
&\quad P(v_{j,f} \mid N^p_{j,f}, N^i_{j,k}) P(v_{f+1,k} \mid N^q_{f+1,k}, N^i_{j,k}) \qquad (6.3) \\
= \frac{1}{P(v_{1,n})} \sum_{j,k,t,h,f} &\alpha_s(h,k) P(N^s \rightarrow N^i N^t) \beta^s_t(h, j-1) \\
&\quad P(N^i \rightarrow N^p N^q \mid N^s) \beta^i_p(j,f) \beta^i_q(f+1,k) \\
&+ \alpha_s(j,h) P(N^s \rightarrow N^i N^t) \beta^s_t(k+1,h) \\
&\quad P(N^i \rightarrow N^p N^q \mid N^s) \beta^i_p(j,f) \beta^i_q(f+1,k) \qquad (6.4)
\end{aligned}
$$

Here Equation 6.2 formally defines what we count as an example of the event we are looking for. The next version breaks it apart into the pieces we need, and the next, Equation 6.4 introduces independence assumptions appropriate for our pseudo context-sensitive PCFGs. These are the same as those for standard PCFGs, except whenever we have the opportunity to condition on the parent of the conditioning non-terminal, we do so. Finally, Equation 6.4 replaces the various terms of the equation with abbreviations for the required probabilities. The outside probabilities ($\alpha_l(m,n)$) should be familiar to those acquainted with the inside-outside algorithm. The probabilities of rules is unchanged, except that when we can, we condition on the parent of the right-hand-side non-terminal. Finally we have introduced a new symbol, $\beta^y_x(j,k)$, that is the inside probability $\beta_x(j,k)$, conditioned on the fact that the parent of N^x is N^y. It can be shown that this last probability is computable in polynomial time (and, to be specific, within a constant factor of the time required to parse the sentence).

Note how Equation 6.4 uses the term $P(N^i \rightarrow N^p N^q \mid N^s)$, the probability we wish to estimate. The same thing happens in the more traditional use of the inside-outside algorithm, where the expression for the counts on a rule's usage involves the probability of the rule. The idea is that one makes an initial estimate of this probability and the inside-outside algorithm modifies this estimate to bring it closer to what it sees in the training corpus. This revised number can be fed in again, leading to the iterative nature of the scheme. The same thing happens here. Fortunately there is a reasonably straight-forward number to use here for the initial estimate, namely the probability of the rule independent of the parent node N^s. Thus the first time through the algorithm we use the unmodified probabilities of the rules. Subsequently the estimates computed on the previous iteration would be used in the next go-round. (However, in our experiment our training corpus was too small for this. Overfitting of data occurred after the first iteration.)

It is not too hard to see how equation 6.4 translates into a form that is not dependent on the CFG being in Chomsky-normal form. We omit this transformation for the sake of brevity. The version implemented, however, is the general one.

6.0.2 Sparse Data

We now want to apply pseudo context-sensitivity to improving the performance of our grammars. Before we can do so, however, we need to overcome one more problem, sparse data.

While the context-sensitive technique we have developed work for any context-free grammar, it was developed in conjunction with our work on grammar induction. The grammars we produce in our learning scheme are typically quite large, as we have sacrificed expressiveness of our grammar formalism in exchange for ease of learning. Thus a typical grammar is about 3500 rules or so. As we have 20 non-terminals our pseudo context-sensitive rules permit about 70,000 (= 3500·20) parameters. As the corpus we have been using has about 300,000 words, and figuring about one rule application per word, it is clear that we do not have enough data to reliably estimate all of these parameters, particularly as some rules are quite rare.

Thus, rather than use the context-sensitive parameters "raw", we smooth them by mixing them in with the non-context-sensitive rule probabilities. The equation we use for the smoothing is

$$P(N^i \rightarrow \alpha^j \mid \rho(N^i) = N^s) = \sigma_1 P(\rightarrow \alpha^j)\rho(N^i) = N^s + \sigma_2 P(N^i \rightarrow \alpha^j) \qquad (6.5)$$

where $\sigma_1 + \sigma_2 = 1$ are the mixing constants. In a smoothing equation like

Equation 6.5 the σ_is can be functions of the conditioning terms in the probability distribution we are calculating. In fact, as the probability of a rule is implicitly conditioned on the left-hand side of the rule (note that the probabilities of rules sum to one for each left-hand side), our σ_is can be functions of both N^i and N^s. The only requirement is that for each pair, N^i, N^s, we still have $\sigma_1(N^i, N^s) + \sigma_2(N^i, N^s) = 1$.

We have used a reasonably standard method for finding optimal settings for the σ_is. We split our training data into two pieces, and used one piece, together with the inside-outside algorithm to estimate our context-sensitive and context free probabilities, $P(N^i \rightarrow \alpha^j)\rho(N^i) = N^s$ and $P(N^i \rightarrow \alpha^j)$. The optimal settings for the σ_is are those which make their probability-weighted sums as high as possible; we used an iterative search procedure and the remaining data to find approximately optimal settings for the σ_is.

6.0.3 Results

What would correspond to a good improvement in the cross-entropy of the tag sequence? To get some idea of this we took one of our best pure PCFGs and generated an artificial corpus from the grammar. We then compared the cross entropy the correct grammar assigned to the tag sequence with that assigned by a tri-tag model. We found that the correct grammar is only .15 bits/tag better than the tri-tag model (2.65 vs 2.80 bits/tag). In our learning work we have been aiming at an improvement over tri-tag of about half of that, in light of the difficulties presented by the complexity of real English, limited availability of data, and limited computational resources.

We derived our context-sensitive grammar from a PCFG developed from related work on grammar induction. We built the context sensitive version of this grammar by applying Equation 6.4 and training over the same corpus from which the PCFG was learned. Our results are obtained using a corpus of 10,000 words drawn from the same source, reserved for testing. Both the context-sensitive and context-free grammars assigned some parse to 99.5% of the words in the testing

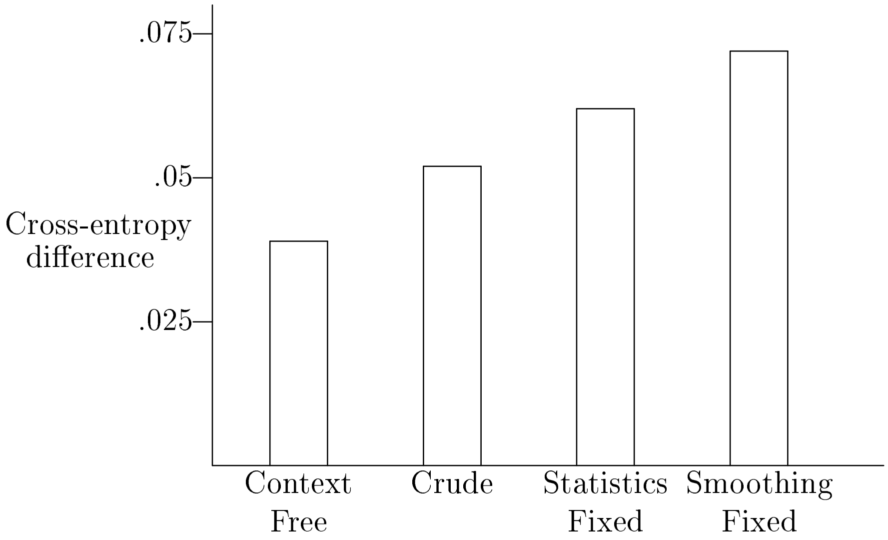

Figure 6.3: Per-word cross-entropy of held-out data with/without context-sensitive probabilities

corpus (99.6% of the sentences). Unparsed sentences are ignored when collecting further data. As the exact same sentences are unparsed by both the context-free and context-sensitive grammars, and the percentage of unparsable sentences is .4% it does not seem likely that these sentences are influencing the results given here.

The results of using our context-sensitive probabilities is shown in Figure 6.3. In all cases we show the improvement over the tri-tag model, which had per-tag cross-entropy of 2.738 bits/tag. (Thus 0 bits/tag in our graph would correspond to a grammar that is no better or worse than the tri-tag model on average.) The left-most entry (.039 bits/tag) shows the results we obtained prior to the use of the context-sensitive statistics. The right-most (.072 bits/tag) shows what was obtained after their use. The difference, .033 bits/tag, is quite large, at least when compared to the goal of a .075 bits/tag improvement.

The intermediate figures are also of some interest. The first, labeled "crude," was obtained getting the counts for the probabilities using the viterbi approximation rather than the correct Equation 6.4. The second, labeled "statistics fixed" was obtained using Equation 6.4 but smoothed the probabilities using a "seat of the pants" guess for the σ_i's ($\sigma_1 = 0$ if the combination of N^i and N^s was seen less than 1000 times, .6 otherwise). Using optimal σ_i's gave the right-most, final, figure. As can be seen, attending to such details does make a difference.

We do not indicate computational resources expended in context-free vs pseudo context-sensitive as there is no significant difference in this regard. The actual parsing is the same in both cases, the only difference appearing after the parse when calculating the probabilities of the tag sequence. While parsing and both probabilistic calculations have big-O complexity n^3, in fact actual time is dominated by the former, as the probabilistic calculations are quite simple.

6.0.4 Analysis

We now turn to the question of the source of this .033 bits/tag improvement. Roughly speaking there are two possible hypotheses. The first is that the context sensitivity sharpens the probabilities across the board. The second is that there are particular situations where it is important to know the context in which a rule occurs, and these provide the lion's share of the benefit. Our initial hypothesis was that the second of these would prove to be the case. In this section we offer evidence that this is so.

We start by remembering that for each parent s and child t there is a distinct distribution for the rules R_t that expand the non-terminal t. This distribution is $P(R_t \mid s, t)$. It gives the probability that t in the context of s is expanded using each $r \in R_t$. The question we pose for each s, t pair is "Is $P(R_t \mid s, t)$ significantly different from $P(R_t \mid \neg s, t)$?" If the difference is large, then the context sensitive technique is buying us a lot in the situation in which s is the parent of t. We estimate significant difference using a likelihood ratio analysis described in [41].

The data for our estimate are the number of times that each of the k rules $r \in R_t$ is used when s is the parent of t, which we designate $C_1(s,t) = \{c_{1,1}, c_{1,2}, \ldots, c_{1,k}\}$, and similarly for the number of times when s is *not* the parent of t, which we designate $C_2(s,t) = \{c_{2,1}, c_{2,2}, \ldots, c_{2,k}\}$.

We estimate $P(R_t \mid s, t)$ using the "obvious" choice:

$$P(r_i \mid s, t) = \frac{c_{1,i}}{\sum_{j=1}^{k} c_{1,j}} \tag{6.6}$$

(and similarly for $P(r_i \mid \neg s, t)$).

Loosely speaking, we compare the chance of seeing our data, C_1 and C_2, given that the distributions are distinct, versus the chance of seeing the data, given that the distributions are really the same. We name the former hypothesis

$$H(P(R_t \mid s, t), P(R_t \mid \neg s, t), C_1, C_2) \tag{6.7}$$

In the latter case, we have

$$H(P(R_t \mid t), P(R_t \mid t), C_1, C_2) \tag{6.8}$$

since in this case

$$P(R_t \mid s, t) = P(R_t \mid \neg s, t) = P(R_t \mid t)$$

Finally, following [41] we consider the quantity

$$-\log \lambda(s, t) = -\log \left[\frac{H(P(R_t \mid t), P(R_t \mid t), C_1, C_2)}{H(P(R_t \mid s, t), P(R_t \mid \neg s, t), C_1, C_2)} \right] \tag{6.9}$$

We lack space to show an exact form for H and $-\log \lambda(s, t)$ (but see [41] for details). Intuitively, however, this is a measure of how likely it is that the context sensitive probabilities for the rules given s, t are really just the context-free probabilities. The advantage of this quantity for our purposes is that it can be computed exactly, starting from the multinomial distribution, and thus is accurate even in the presence of rare events, which, if we may be excused the oxymoron, are quite common in our data. (Many of the rules occur less than ten times in our data. Thus the number of times we would expect them to occur with a particular parent s may well be less than one.)

Note, also, that in most normal circumstances $-\log \lambda(s, t)$ grows linearly in the number of times s, t are observed together, $C_1(s, t)$. Intuitively this captures the idea that more data allows one to make finer discriminations. The other

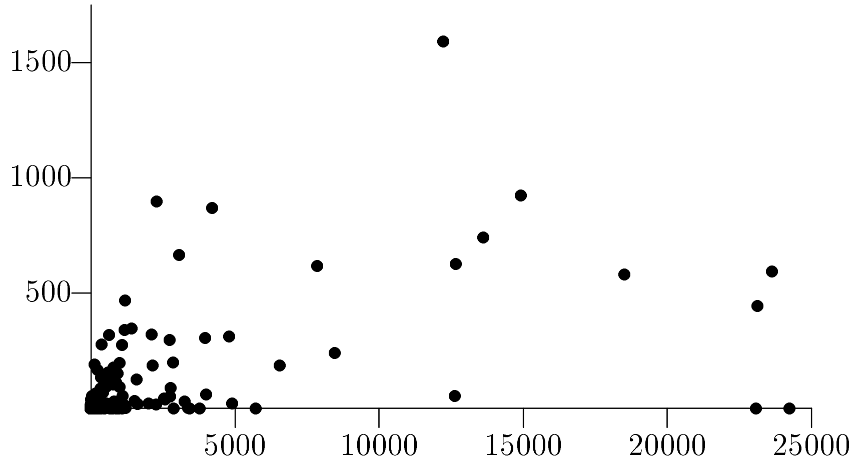

Figure 6.4: Plot of $-\log\lambda(s,t)$ against $C(s,t)$

contributing factor, naturally, is the difference between the observed distributions $P(R_t \mid s,t)$ and $P(R_t \mid \neg s,t)$. Because of this we decided to plot $-\log\lambda(s,t)$ against $C_1(s,t)$, with one point for each s,t combination. If the result were a straight line it would indicate that the various $P(R_t \mid s,t)$ distributions differed to approximately the same degree from their context-free equivalents, $P(R_t \mid t)$, and that the difference in the $-\log\lambda(s,t)$ is just due to having more data for some points, the larger $C(s,t)$'s, than others.

The results shown in Figure 6.4 are quite different. While there is clearly a positive correlation between $C_1(s,t)$ and $-\log\lambda(s,t)$, it is hardly a straight line. Instead a quick glance at the chart suggests that a small number of s,t combinations account for the overwhelming majority of the context-sensitive effect.

To further test this hypothesis, we modified the smoothing equation 6.5 to assign a σ_1 of zero when calculating the probability of a rule given an s,t pair not in the top n pairs, when sorted by $-\log\lambda(s,t)$. Figure 6.5 shows that by the time we have considered 51 out of the 400 s,t combinations we have captured virtually all of the context-sensitive effect, and even by 21 s,t combinations (5% of the data) we have most (80%) of the effect. This suggests that our initial hypothesis, that the effect is concentrated in a small number of cases, is basically correct.

Recall that our pseudo PCSG is not truly context-sensitive, because it does

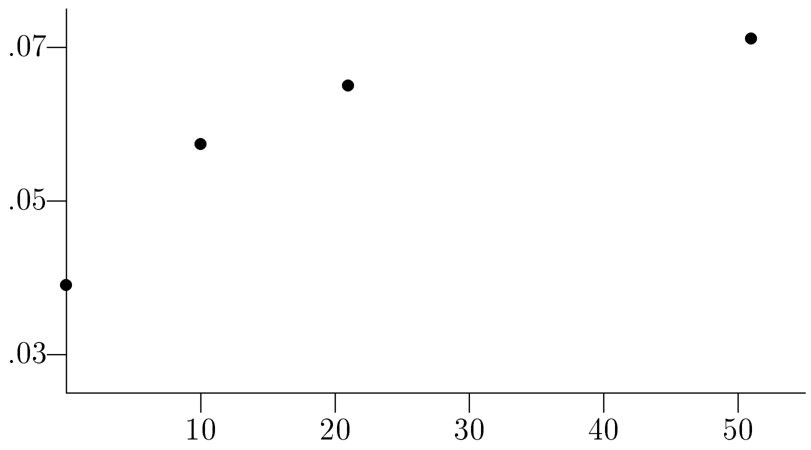

Figure 6.5: Cross entropy as a function of increasing numbers of s, t's

not move out of the range of languages generated by PCFGs, and it is possible to "compile out" our context-sensitive statistics. Since performance benefits are concentrated in a small number of s, t pairs, and the compilation procedure can be carried out incrementally, on a per s, t pair basis, the transformation appears to be practical. The worry here is that the grammar might be so large as to be useless. Even limiting ourselves to the 20 best s, t pairs, adding a non-terminal for each pair doubles the number of non-terminals in our grammar, and could more than double the number of rules. Further, we did not smooth the expanded grammar as we did for the pseudo PCSG.

Nonetheless, the observed localization was encouraging, and back-of-the-envelope calculations suggested that the expanded grammar would be only about 10,000 rules, which is a manageable size. We carried out the experiment of transforming the grammar and evaluating its performance. Ignoring rules with zero probability, the transformation added 5384 rules to the existing 3500. This more than doubles the size, but it was actually less of an increase than expected. After two iterations of the inside-outside algorithm, the grammar began to overfit the training data, but performance reached the same level as that of the pseudo PCSG. To be precise, the trained PCFG showed an *improvement* over the pseudo PCSG of 0.001 bit per word, even though it had fewer parameters (about 8,000 vs. 17,000). We do not regard this improvement as significant, but the fact the PCFG can recover the missing 20% performance gain is very satisfying.

This experiment shows that our scheme can be used as a novel form of PCFG induction, one which adds both rules and non-terminals, and revises the probabilities to produce significantly lower cross entropy. Adding non-terminals is a particularly sticky problem for grammer learners, as the unconstrained space is too large to search. What is usually done is to fix the number of non-terminals in some other way, either using outside sources of information [24], or, as we do, via a restricted formalism [42]. Another approach, suggested in [19] is to use a CNF grammar, simply guess an upper bound on the number of non-terminals, and deploy a grammar minimization procedure periodically during the grammar training. The appeal of our approach is that it does not require guesses, but can automatically identify a set of promising new non-terminals, and associated rules and probabilities. It is, admittedly, highly constrained, but we regard this as more of a feature than a drawback. Overly large search spaces require learners to deploy constraints. Our procedure is restricted enough to be feasible for a large problem (English), but loose enough to allow significant performance gains.

6.0.5 Summary

We have presented a PCFG model in which the probability of a rule also depends on the parent of the node being expanded. The scheme is applicable to any PCFG and the equations we have derived allow one to collect the necessary statistics without requiring preparsed data. In the experiment we ran, the improvement over the context-free version is quite large, given the expected range. We have analyzed the context-sensitive statistics, and shown that most of the effect is fairly localized.

This localization encouraged us to attempt to use statistics gathered for our pseudo PCSG to extend our original PCFG. By using the most promising s, t pairs, we demonstrated that expanding the grammar retains the performance gains of the pseudo PCSG, despite the reduction in the number of parameters. This implies that our scheme is not only good for improving performance by means of a pseudo PCSG, but it may also be viewed as a systematic means for inducing non-terminals, rules, and probabilities for a PCFG.

Chapter 7

Sensitivity Analysis

In this chapter we evaluate the behavior of our scheme under different parameter settings. The reference system is the basic SINGER with both $\overline{\overline{\mathsf{S}}}$ and $\overline{\overline{\mathsf{Q}}}$ non-terminals and learning split over 60 segments. The bias grammar was trained once, pruned to 629 rules, and scaled so that the start rule has a count of 4,000. The pruning threshold used during rule selection is 0.01. The meaning of these parameters is described in more detail below. Rule acquisition takes between 1 and 6 hours per segment, and anywhere from 20 to 80 hours for 10 training iterations. 60 segments is a compromise between the cost of computation and the need to reduce the variance of the learned grammar. These results are derived from a single ordering, not an average over multiple orderings.

Segmentation also acts to reduce the cross-talk between learning events, the effect whereby rules generated for on event prevent learning later on. This cross-talk is at a amaximum for unsegmented learning and is completely eliminated when the segmentation equals the number of sentences. The latter is the cleanest test, but too expensive for multiple experiments.

7.1 Pruning Threshold for Rule Selection

The pruning threshold controls how many rules are allowed to pass rule selection. Following each parse (during rule selection), the count of each new rule

threshold	cross entropy	lambda	coverage	acquired rules	retained rules	parse failures
0.1	2.7054	0.711	99.8	3461	2774	4996
0.031623	2.7045	0.719	99.8	4419	3261	4529
0.01(standard)	2.7039	0.715	99.8	5810	3659	4184
0.00316	2.7027	0.722	99.8	7110	3713	3866
0.001	2.7007	0.727	99.8	8503	3629	3596
0.000316	2.6949	0.740	99.7	10003	3659	3455

Figure 7.1: Results from various pruning thresholds used during rule selection.

is compared with the threshold, and rules with lower counts are discarded. Recall that each terminal appearing in a sentence contributes a count of 1.0 to rules headed by the corresponding non-terminal. Therefore, for a given terminal occurring O times in a sentence, setting the threshold at $1/N$ permits at most $N \cdot O$ rules headed by the corresponding non-terminal. The maximum occurs when rules have uniform counts; since the counts are generally highly non-uniform, the effective threshold is much lower.

Thresholding on the rule count has the disadvantage that it is possible for all rules to be eliminated. If, for example, too many rules are generated, and the counts are divided too finely on the initial parse, the counts for all rules may fall below threshold. We do not see this in practice, as under those circumstances we terminate rule generation and abandon learning to prevent the chart from overflowing memory. The most obvious alternative strategy to thresholding is to take the N best rules. This avoids the possibility of all rules being eliminated but it has the converse problem of permitting rules with arbitrarily low counts.

At high values, such as 0.1, the pruning threshold permits very few rules to pass selection. Therefore we expect more learning events and fewer rules at the end of learning. Since the learning is being more tightly constrained to fit the training data, we also expect to see some signs of overtraining, such as lower coverage on test data. Figure 7.1 shows the results of our experiments. The cross entropy and lambda columns should be self-explanatory. Coverage indicates the

	pruning threshold					
	0.1	0.0316	0.01	0.00316	0.001	0.000316
training cross entropy	2.6687	2.6615	2.6563	2.6559	2.6533	2.6547

Figure 7.2: Training cross entropy for various pruning thresholds.

percentage of words parsed in the test corpus. The acquired rules column gives the number of rules in the grammar at the end of the rule acquisition phase. The rules retained indicates the number remaining at the end of 10 training iterations. The number of parse failures indicates the number of times rule learning was attempted.

The figure shows that our expectations concerning the number of learning events and learned rules are met, but the overtraining is not apparent. Instead, cross entropy slowly but steadily drops as the threshold decreases, while coverage remains constant at nearly 100%. Another sign of overfitting would be better performance on the training data coupled with worse performance on test data. Figure 7.2 shows the cross entropy results on training data, but again no overtraining is apparent. Rather, the training cross entropy correlates with test cross entropy until the lowest setting, at which an insignificant rise in the training cross entropy is seen. It appears that even allowing fewer than 10 rules per learning event is not enough to cause overtraining. Improved performance comes from permitting more rules through the filter, which is not surprising since our selection mechanism is imperfect, and not all the best rules have the highest count. Working against this trend is the diminished number of learning events, but this seems to be have no effect.

While learning with a lower pruning threshold is more expensive, the final grammar is not more expensive to use. Figure 7.1 shows that, although the number of rules at the start of training is quite high, the final number is stable at around 3600-3700. This means the cost of parsing with the final grammar will be about the same, regardless of how it was learned.

7.2 Rule Form Constraints

The unparameterized rule form constraints, those which had no welcomeness or precedence numbers, are crucial to preventing memorizing. Without them, the prototype SINGER could not learn even a simple artificial grammar with fewer than 50 rules [31]. Instead, it would memorize the data, acquiring one rule per sentence type. With rule length limits in place to prevent this, the learner would break the longest rules into a few shorter ones, but generalization would still be minimal.

Singer now uses more sophisticated constraints with numerical parameters for the welcomeness of one non-terminal in a rule headed by another and the likelihood of the first non-terminal appearing before the rule's terminal. Our experience has been that it is most important to get the permissions right, allowing the right rules to be formed. The numerical values are less important, which is why we have not bothered to try and estimate them from the data.

We initially adopted the parameterized constraints as a package together with the exponential penalty on rule length. The data for these results come from one unsegmented run of SINGER and one run with 10 segments. This version of SINGER has no $\overline{\overline{\mathsf{Q}}}$ non-terminal in the bias grammar or the learning procedure, but is otherwise identical to the reference system. The combination of parameterized constraints and rule length penalty showed a net benefit of 0.0030 and 0.0006, in the unsegmented and segmented run, respectively. Coverage improved 1% and 0.5%, and the number of rules acquired dropped by 840 and 3164, alo respectively. The last benefit is the only one of above the level of noise.

As a second experiment, we tested the importance of the welcomeness parameters by making them uniform. All parameters were converted to 0.1, which gives reasonable values for rule initialization. (The remaining rule selection mechanisms, including the length penalty and the usual training and pruning were left in place.) We did not test the before/after parameters because they are often at or near extrema, 1.0 or 0.0, and so they act more like symbolic than numerical constraints. The final grammar from this experiment had a cross entropy of

2.7140 and a lambda of 0.685, for a loss of 0.0101 bits/word and .03 in lambda score, in comparison with the reference system. This is a substantial loss, especially in cross entropy. It indicates that the parameters should correspond at least approximately to linguistically plausible rule structures. Uniform welcomeness parameters allow us no discrimination between the plausible and implausible rules.

7.3 Rule Length Penalty

The rule length penalty is the exponential divisor applied to each new rule after its weight is calculated from the welcomeness and before/behind constraint parameters. The intent is to prefer shorter, more general rules, as opposed to long rules that model sentences as flat lists. Our results show that preferring shorter rules incurs a performance penalty rather than a benefit. Compared with standard SINGER on 60 segments, it costs 0.0056 bits/word and 0.19 on our lambda measure. One would expect the length penalty to be somewhat redundant with the length limits and the masking of those limits induced by ordering the sentences. Given the ineffectiveness of changing the length limits, described below, it is odd that the length penalty should have such a strong effect.

The cost of this cross entropy benefit is an increase in both learning and final grammar size. The unpenalized SINGER acquires 8717 rules during learning versus 5810 for the standard, and finishes with 4674 versus 3659. Thus, we see the usual correlation of more rules and higher performance. It appears this penalty might be reasonably discarded, as it somewhat redundant with the length limits, the pruning threshold, and the welcomeness constraints. The latter act as something of a length penalty, since longer rules multiply more welcomeness weights, and hence end with lower initial values.

7.4 Rule Length Limits

The original impetus for limiting the length of generated rules came from our earliest experiments with grammar learning [31]. The main difference between that learning scheme and the present one is that the former had no constraints on the form of rules. As a result the learner would overfit the data, in a very stylized way that we dubbed "memorizing", as there was essentially one rule per sentence type. The grammars were large, the rules were long, and coverage of test data was low, but cross entropy of the covered sentences was very good. Blocking rule generation at a given length was an attempt to prevent memorizing the training data, the idea being that if the rules could not grow so long, then some generalization would have to place. As an anti-overfitting measure, the length limits failed even in the original experiment, as the memorization behavior degraded very gracefully.

A second motivation for these limits is that they save computation time. We do not expect to find any useful rules of length 9 or longer. Moreover, the number of useful rules of length 7 is probably very small, but searching for such rules is much harder because the size of the space is an exponential function of rule length. In the present scheme, we have three "firewalls" in place to prevent the kind of slowdown that occurs when the number of rules blows up. The first terminates the rule generation and declares the sentence an exception whenever the number of rules is more than 100 times the sentence length. The second is the rule pruning during selection, described above. The last is a monitor that checks the size of the chart. If, despite the limits on rule generation, the grammar has managed to grow in such a way that the chart exceeds 100,000 entries, the attempted parse is abandoned and the sentence is declared an exception.

Results from various settings of the length limit are given in figure 7.3. Chart and rule overflow columns indicate the number of times those two firewalls described above screened out a sentence.

Figure 7.3 shows no significant difference in performance at the various limit settings. Both the cross entropy and lambda numbers are so close as to be

length limit	cross entropy	lambda	coverage %	# of rules
4	2.7022	0.720	99.8	5768
5	2.7027	0.720	99.8	5637
7	2.7026	0.720	99.8	5682
9	2.7030	0.720	99.8	5682

Figure 7.3: Results from varying the rule length limit. 5 is standard.

length limit	parse failures	learning successes	chart overflows	rule overflows
4	4182	3802	38	269
5	4219	3722	45	359
7	4205	3745	36	346
9	4215	3738	50	328

Figure 7.4: Overflow esults from varying the rule length limit.

indistinguishable. This is not surprising, given the other protections in place to control rule generation.

What is somewhat surprising are the figures on parse failures and rule overflows, shown in figure 7.4. We gathered these figures in the expectation that the raised limits would show more overflows, especially rule overflows. Instead there is no clear trend. The shortest length has the least rule overflows, but the next higher length has the most. One explanation is that the sentence ordering is masking the effect of the length limits. If most learning is done before longer sentences are reached, the effect of raising the length limits is diminished. A second possibility is that these parameters are redundant with the pruning threshold and rule length penalty. Since longer rules are still discriminated against during rule selection, generating longer rules is ineffective.

Despite these mixed results we are disinclined to raise or remove the length limit, for the simple reason that rule generation would become more expensive,

weight	cross entropy	lambda	coverage	rules	failures
40	2.7009	0.719	99.7	6170	3904
400	2.7029	0.726	99.8	5926	4006
4000	2.7039	0.715	99.8	5810	4184
12,138	2.7022	0.718	99.8	5782	4249

Figure 7.5: Results from scaling the bias grammar to various weights.

perhaps vastly so. As the generation process closely resembles parsing, it faces the same problem of overflowing memory, and the same abrupt loss in performance once memory bounds are exceeded. Roughly speaking, it takes 10,000 times as long to parse a sentence when using a disk for memory, changing rule generation and parsing time from seconds to hours. In the absence of clear performance gains, we prefer not to exercise our firewall code by moving to longer limits.

7.5 Bias Grammar Experiments

7.5.1 Weighting

By "weight" of the bias grammar we mean the count of the unique starting rule prior to rule acquisition. (The unique starting rule is the only rule in our grammar headed by the start symbol.) Following training, the count of this rule equals the number of sentences the raw bias grammar was able to parse. We scale the grammar by changing this count, multiplying it by an appropriate factor, and simultaneously multiplying the counts of all other rules.

The weight of the bias grammar essentially measures how much credibility it has to start out with. The difference between grammars with high weight and grammars with low weight lies in how generated rules are used during the learning phase. For a low weight grammar, new rules may quickly overtake bias rules in both count and probability. The archetypal example would be a rule suggested early and used often. Eventually, when the grammar's probabilities are re-estimated (each time a parse failure is encountered), the probability is raised enough for it to overtake some bias rules. For purposes of later parsing and learning, then, generated rules and bias rules are on an even footing.

If the grammar has very high weight, on the other hand, then acquired rules can never be used often enough to overtake the bias rules in count or probability. This leads to discrimination against generated rules during learning, since the selection mechanism is probability based and it favors those parses which use as few generated rules as possible. The net effect is to discriminate against any rule that requires another non-bias rule, whether the latter first appears at the same time or any time previously. Lightweight grammars discriminate only against rules that require each other's cooperation and appear simultaneously or within a small number of sentences.

Figure 7.5 shows the results of our experiments at various weights. The standard scale factor is 4000 and the upper limit of 12,138 reflects the number of sentences parsable by the bias grammar. In all cases, we retained the 629 rules standardly used, i.e. no pruning was done on the lightweight grammars.

The discrimination effect can be seen in the number of rules acquired and the number of parse failures at various weights. Lighter grammars have fewer failures and more rules. Since rules generated early license later ones, more rules pass the selection phase at each learning event, on average. Since more rules are acquired at each event, there are fewer parse failures overall.

This leaves unexplained the crucial performance figures. For the most part, the numbers are gratifyingly flat—the difference between the best and worst cross entropy are just a little above significance, and the lambda figures are even closer to insignificance. There is no clear trend among them. The lowest weight shows the best cross entropy, but the lambda figure is comparable to the lambda's of the other three weights, presumably because coverage drops slightly. The next weight up, which has the best lambda, also has the second-worst cross entropy.

The mixed results leave us with some somewhat speculative conclusions. High-quality bias grammars deserve high weight. On the other hand, good mechanisms for generating and initializing rules should perform better when the bias grammar is lighter. Our own versions of bias and rule generation are both of

iterations	cross entropy	lambda	coverage	rules	failures
1	2.7039	0.715	99.8	5810	4184
5	2.7023	0.732	99.9	5573	4272
10	2.7052	0.726	99.9	5627	4289

Figure 7.6: Results from pretraining the bias grammar various numbers of times.

medium-high quality, so the choice has only minor impact on the learned grammar.

7.5.2 Pretraining

Before the rule acquisition phase, the raw bias grammar is trained once. The intention is to discard unused rules and to give reasonable values to those that are retained. Since the bias grammar has low coverage, only about 50% of the sentences are parsed. Our expectation prior to running this experiment was that no significant performance difference would result. It is worth noting that training the bias grammar can cause good rules to disappear or develop anomalously low probabilities, since many rules are missing.

The results, shown in figure 7.6, fit our expectations fairly well. Pretraining for five iterations performs slightly better than pretraining only once, but not enough to be significant. Pretraining 10 times leads to a performance drop, presumably because the grammar is shaped too much by the 50% of sentences it can parse. The number of rules generated does not vary widely.

7.5.3 Pruning

Following training and scaling, low count rules are pruned from the grammar. The idea is that some of these rules may not be justified, because even though they were used for some parses, it could still be the case that those phrases would be better handled by some other rules not present in the grammar.

In figure 7.7 we show results from a single run with 18 segments and no

remaining rules	cut-off	cross entropy gain	lambda	coverage	final rules
792	0.5	.0317	0.683	99.5	3125
629	1	.0332	0.686	99.5	3127
440	2	.0331	0.690	99.5	3180
326	4	.0337	0.697	99.8	3327
264	6	.0337	0.690	99.5	3549
207	10	.0329	0.687	99.5	3687
149	20	.0252	0.658	99.5	4065

Figure 7.7: Results from pruning the bias grammar to various levels.

$\overline{\overline{\mathsf{Q}}}$ non-terminal in either the bias grammar or the learning mechanism. Other parameters were set as for the reference system. The cross entropy gain indicates the difference in cross entropy between a trigram model and the grammar model; a positive gain indicates lower cross entropy (higher gain is better performance for the grammar). These results indicate that the 3k rules that are discarded have many bad rules or rules whose probability is so distorted as to interfere with good learning. There may be some such rules among the top 400 as well, but their contribution is outweighed by the benefit of the good rules. At 325 rules or below, the rules are no longer adequate to guide learning.

7.6 Conclusions

Overall, Singer does not show an undue sensitivity to small changes in its parameter settings. The values we have chosen seem reasonable and/or convenient to us. Our experiments show that several parameters provide some tradeoff between final performance and grammar size, which corresponds roughly to trading variance and bias error. (Larger grammars have more parameters, and therefore higher variance, but are generally able to better fit the data.)

A careful reader may have noticed that our standard Singer differs from the "optimal" settings in several experiments. As a final experiment, we employed the optimal settings, as follows: the bias grammar was pretrained 5 times, and scaled to 400. Rather than explicitly prune it, we let the default pruning take place, which reduced the number of rules to 286. The rule length penalty was dropped, the length limit lowered to 4, and the pruning threshold lowered to 0.001. The final grammar had a cross entropy of 2.6866 and a lambda of 0.767, for a gain of 0.01727 bits/word and 0.052 in lambda score. Unfortunately this grammar was also quite a bit larger than usual, starting at 15991 rules and ending with 5354. Coverage was 99.7% and there were 4275 parse failures. This fit our expectations that the gains would not be additive for different parameter adjustments, but neither would the adjustments undermine each other. This grammar has the best performance of any reported in this section; so far as we can tell, significantly better performance is only possible with more segments or the use of context.

Chapter 8

Evaluation of an Alternate Formalism

Real linguistic theories are much more complicated than our stripped-down "dependency grammar" formalism. While our simplifications make it much easier both to implement and adapt our learning scheme, it is natural to wonder how well a more linguistically faithful formalism would work. In this section, we examine the performance impact of migrating from our simplified formalism back to its more complicated progenitor, $\overline{\mathsf{X}}$, as described by Jackendoff [32].

8.1 $\overline{\mathbf{X}}$ Theory

The major structural difference between dependency grammars and $\overline{\mathsf{X}}$ grammars lies in the way rules are headed. For dependency grammars, each rule is headed by a single-barred non-terminal, $\overline{\mathsf{X}}$, and has exactly one terminal on the right-hand side. $\overline{\mathsf{X}}$ grammars allow additional non-terminals with two bars, following the same naming scheme as for dependency grammars. (Some theories extend this to triple-barred non-terminals, but here we consider only double-bar versions.) The basic $\overline{\mathsf{X}}$ rule format is somewhat more restrictive than dependency format. Each single-barred non-terminal heads rules dealing with the head word's complement, so the form is

$$\overline{\mathsf{X}}\rightarrow\mathsf{X}\ (\overline{\overline{\mathsf{Y}}})^{*}$$

No non-terminal may precede the head word, and only double-barred non-terminals may follow it. Double-barred non-terminals head rules dealing with the head's specifiers, so rules take the form

$$\overline{\overline{\mathsf{X}}}\rightarrow(\overline{\overline{\mathsf{Y}}})^{*}\ \overline{\mathsf{X}}$$

A real $\overline{\mathsf{X}}$ theory might be more restrictive than this, permitting only zero, one or two specifier or complement non-terminals in the appropriate rule form. We have no such restriction here, for reasons that will become apparent when we discuss adjunction rules.

Besides the difference in bar level, $\overline{\mathsf{X}}$ permits special rule forms for handling conjunctions and adjunctions. The former take the form

$$\overline{\overline{\mathsf{X}}}\rightarrow(\overline{\overline{\mathsf{BTH}}})\ (\overline{\overline{\mathsf{X}}})^{*}\ \mathsf{AND}\ \overline{\overline{\mathsf{X}}}$$

where $\overline{\mathsf{BTH}}$ is the pseudo-determiner non-terminal governing "both", "neither", and "either". This may appear only once or not at all, and only given a non-terminal between it and the $\overline{\mathsf{AND}}$. These rules allow better generalization of the basic specifier and complement rules, since otherwise conjunctions can only be handled by tacking $\ldots\overline{\overline{\mathsf{ANDX}}}$ onto the end of complement rules. (This mishandling is approximately what we do in our dependency grammars.) We allow zero preceding items in order to handle occurrences such as sentences beginning with "but".

Last, we come to adjunction rules, which handle modifiers. Formally modifiers can occur in unlimited numbers, so the rules take the following special forms:

$$\overline{\overline{\mathsf{X}}}\rightarrow\overline{\overline{\mathsf{Y}}}\ \overline{\overline{\mathsf{X}}}$$

$$\overline{\overline{\mathsf{X}}}\rightarrow\overline{\overline{\mathsf{X}}}\ \overline{\overline{\mathsf{Y}}}$$

$$\overline{\mathsf{X}}\rightarrow\overline{\overline{\mathsf{Y}}}\ \overline{\mathsf{X}}$$

$$\overline{\mathsf{X}}\rightarrow\overline{\mathsf{X}}\ \overline{\overline{\mathsf{Y}}}$$

This is not a complete $\overline{\mathsf{X}}$ theory, nor is it likely to please every linguist, even as a partial theory. However, these rule schemes are among the most basic, so they are appropriate candidates for investigation. The aim is to see whether they help or hurt performance much.

Clearly the definitive result here would come from incorporating all of these changes, testing them together, and showing that no significant loss in performance is incurred. This would be good evidence that our learner is at least somewhat formalism-independent. A negative result would be more ambiguous, since there are many points at which the failure could occur. We have carried out the experiment and obtained negative results.. For this reason, and because more focused experiments give a clearer picture, we first present the experiments that tested these ideas piecemeal.

8.2 Specifier/Complement Rule Format

The first experiment concerns the suitability of phrasing rules in specifier and complement forms. From a statistical point of view, this form asserts that the specifiers and complements occur independently of each other. If this is true, then it should allow a more compact grammar to perform as well as or possibly better than a dependency grammar, since dependency grammars cannot capture this independence.

Consider the category $\overline{\mathsf{NN}}$. Suppose that nouns may take articles, determiners, or nothing as specifiers, then we have the following rules in $\overline{\mathsf{X}}$:

$$\begin{aligned}
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\overline{\mathsf{ART}}}\ \overline{\mathsf{NN}} \\
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\overline{\mathsf{DET}}}\ \overline{\mathsf{NN}} \\
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\mathsf{NN}}
\end{aligned}$$

For complements, suppose only prepositions or nothing are permitted. This gives

$$\begin{aligned}
\overline{\mathsf{NN}} &\rightarrow \mathsf{NN}\ \overline{\overline{\mathsf{PREP}}} \\
\overline{\mathsf{NN}} &\rightarrow \mathsf{NN}
\end{aligned}$$

	cross entropy	coverage %	lambda	size
standard dep gram	2.7006	99.7	0.7266	4176
same in spec/comp form	2.7063	99.8	0.7026	3613

Figure 8.1: $\overline{\mathsf{X}}$ Performance results

There are 6 corresponding dependency rules, which may be obtained by taking the cross product of the specifier and complement rules. This illustrates the essential tradeoff involved in moving from one bar level to two. The set of dependency grammar rules for a given non-terminal corresponds to the cross product of specifier and complement rules in $\overline{\mathsf{X}}$, so a dependency grammar will usually have more parameters and require more data to set them equally well. On the other hand, if complements and specifiers do not occur independently, then $\overline{\mathsf{X}}$ will perform poorly because it cannot capture this dependence.

To test this hypothesis, we converted one of our better dependency grammars to $\overline{\mathsf{X}}$ by splitting each rule at its right-hand terminal, and increasing the bar level as appropriate. The one exception to this was for rules headed by $\mathsf{S} =$, which are not in dependency grammar format, and were therefore passed on unmodified. For split rules, the count of the parent was used for both new rules. In fig. 8.1 we tabulate the results of evaluating the derived grammar. The cross entropy increases significantly and the lambda likewise drops, indicating that this grammar is unlikely to be more useful for speech recognition. However, the penalty, 0.006 bits/word, is not severe, so the independence assumption is not terrible.

If the resulting grammar were significantly smaller, $\overline{\mathsf{X}}$ grammars would be attractive for learning. Unfortunately, the shrinkage is only about 13%, primarily because our dependency grammar, even though it only has one bar level, is biased towards a quasi-$\overline{\mathsf{X}}$ form. For example, the main verb rules, headed by $\overline{\mathsf{VB}}$ and $\overline{\mathsf{AUX}}$ almost never have a non-terminal preceding the terminal. There are 427 $\overline{\mathsf{VB}}$ rules in the standard grammar, which divide into 7 specifier rules and 379 complement rules. For $\overline{\mathsf{AUX}}$, the figures are 433 standard, which split into 5 specifier and 387

terminal	ADV	ADJ	PREP	S=	VBG	VBN	AUX	VB	NN	.
standard	39	58	66	185	200	249	427	433	684	1760
$\overline{\overline{\mathsf{X}}}$	8	16	5	186	16	12	5	7	195	1736
$\overline{\mathsf{X}}$	20	24	48	–	130	177	387	379	184	1

complement rules. Table 8.1 gives the the 10 non-terminals which head the most rules, and the number of rules each heads in the dependency grammar and the corresponding $\overline{\mathsf{X}}$ grammar.

A second reason for the poor shrinkage arises from the number of rules which handle sentence-level constructs, headed by either $\mathsf{S} =$ or $\bar{.}$, which is effectively our start symbol. Both of these resist compression, the former because it is not in dependency grammar format, and the latter because nothing can follow the terminal period in a sentence. Together these account for half the rules of the grammar.

The above implies our results should be taken with a grain of salt. However, they are supported by evidence from the $\overline{\mathsf{NN}}$ non-terminal, which does not have any strong prejudice for $\overline{\mathsf{X}}$ form. There are 684 $\overline{\mathsf{NN}}$ rules in the unprocessed grammar, yielding 195 $\overline{\mathsf{NN}}$ and 184 $\overline{\overline{\mathsf{NN}}}$ rules. This is a reduction of about 45%, a much better and more plausible figure for a grammar that really is in a dependency grammar format, but still far from the theoretical maximum, $\sqrt{684}$ or about 26 rules for each category.

8.3 Conjunctions

Singer's constraints on $\overline{\mathsf{AND}}$ say little more than that it heads no interesting rules. There is no requirement that it appear in rules with the symmetry mandated by $\overline{\mathsf{X}}$, and there are many violations of this symmetry. Some of these rules have more structure than $\overline{\mathsf{X}}$ deems appropriate, such as specifiers or adjuncts. Others handle asymmetry which arises from the data itself, as in

> "Ames was a spy and proud of it."

	cross entropy	coverage %	lambda	size
standard	2.7006	99.7	0.7266	4176
symm $\overline{\mathsf{X}}$ rules	2.72587	99.72	0.642138	2890
asymm $\overline{\mathsf{X}}$ rules	2.71564	99.80	0.677837	2850

Figure 8.2: Performance results for $\overline{\mathsf{X}}$-style conjunctions

Here one conjunct is a noun ($\overline{\mathsf{NN}}$) and the other an adjective phrase ($\overline{\mathsf{ADJ}}$). If there is any symmetry, it exists at the semantic, rather than the syntactic level.

To evaluate the benefits of using $\overline{\mathsf{X}}$-style conjunction rules, we began by deleting all rules with $\overline{\mathsf{AND}}$ from the regular bias grammar. Since $\overline{\mathsf{AND}}$ rules are so tightly constrained under $\overline{\mathsf{X}}$, we chose not to induce them on the fly; rather, they were generated offline and added to the bias. We allow as few as one conjunct and as many as five. Following the $\overline{\mathsf{X}}$ template for generating conjunction rules, this generates 100 rules from 20 non-terminals. We also permit an optional pseudo-determiner "both", "either", or "neither" at the front of phrases with two or more conjuncts. This brings the total to 180 rules. Finally, we hand-coded 26 asymmetrical rules, based partially on the bias grammar, and partially on a spot check of the training data. These are listed in appendix D.

For our experiment, we uniformly initialized these rules, and added them to the bias grammar. We adjusted the constraints to prohibit learning of new conjunction rules, and carried out the normal rule acquisition and training.

As fig. 8.2 shows, these rules exact some penalty in performance, 0.025 bits/word without the asymmetrical rules, and 0.015 bits/word with them. As with specifier/complement format, the slide is significant, but not disastrous.

8.4 Trying It All Together . . . almost

In this section we describe the results of putting all of the above together, effectively converting Singer to an $\overline{\mathsf{X}}$ learner. While the principle is clear, there are a lot of details to take care of.

8.4.1 Bias grammar

To try and make a fair comparison, we based the bias grammar on the standard one in use for dependency grammars. We cut the rules into specifier/complement halves, eliminated old-style rules containing $\overline{\mathsf{AND}}$, and substituted in $\overline{\mathsf{X}}$-style rules, as above. Since $\mathsf{S} =$ is not part of $\overline{\mathsf{X}}$, and $\overline{\mathsf{X}}$ has enough parameters to cope with the same kind of phrases, we eliminated it. First we eliminated rules headed by $\mathsf{S} =$ by searching for $\overline{\overline{\mathsf{VB}}}$ or $\overline{\overline{\mathsf{AUX}}}$ on the right-hand side. (These occurred mutually exclusively in our grammar.) If either was found, the corresponding double-bar non-terminal was used to head the replacement rule. If neither appeared, one of $\overline{\mathsf{VBG}}$, $\overline{\mathsf{AUXG}}$, or $\overline{\mathsf{VBN}}$ was searched for and used. These were not all mutually exclusive; whichever appeared first was treated as the head[1]. Together, these non-terminals accounted for all rules headed by $\mathsf{S} =$. As $\mathsf{S} =$ rules were almost universally in $\overline{\mathsf{X}}$ form, replacing the head was a reasonable conversion.

For each rule in which S= appeared on the right-hand side, we created 5 new rules, headed by one of the five non-terminals named above. To compute the counts for these new rules, we first computed a weight for each of the 5 non-terminals, equal to the total count of all former S= rules now headed by this non-terminal, divided by the total count of all former S= rules. The new rules' counts were initialized by multiplying the weight of the substitute head and the count of the progenitor rule. To a first approximation, this preserves the string probabilities assigned by the grammar.

8.4.2 Constraints

The constraints were already in a form suitable for $\overline{\mathsf{X}}$, and required only a slightly different interpretation. Where before the data indicated the welcomeness of a non-terminal preceding or following the terminal, now the same data would indicate the welcomeness of a non-terminal in a specifier or complement rule.

[1] In retrospect, this was a mistake, as these rules were primarily in specifier format. While it would have been better to choose the last as the head, the number of such rules is so small in any case, as to make no difference.

The only alterations that had to be made to the numbers came from our elimination of S=. This required "extending" the constraints applied to $\overline{\mathsf{VB}}$, $\overline{\mathsf{AUX}}$, and so on with the constraint previously applied to S=. Extending a constraint sounds backwards, but ours are written in terms of what is permitted, so we were really transferring permissions previously conferred only on S= to the other non-terminals. Most of these permissions were already there, but the actual weights had to be modified to reflect the new division of labor. This process was analogous to the rule elimination and made use of the same relative count data for $\overline{\mathsf{VB}}$, $\overline{\mathsf{AUX}}$, etc.

8.4.3 The Rule Generator

Aside from adjunctive and conjunctive rules, the rule generator is very similar to that used for dependency grammars. Where the latter generates one rule, the former creates two, in specifier/complement format. Since conjunctive rules are handled by generating them offline, we need not worry about them. Adjuncts, however, are problematic.

For any rule of length n, there are $n-1$ possible adjunctive rules that could be generated along with it. Left unconstrained, this would generate between three and four times as many rules at each step, unless some other limit was reached. Since there are only $|NT|^2$ rules of the form $\overline{\overline{\mathsf{X}}}\rightarrow\overline{\overline{\mathsf{XY}}}$, another $|NT|^2$ for rules of the form $\overline{\overline{\mathsf{X}}}\rightarrow\overline{\overline{\mathsf{YX}}}$, and the same for rules with only single-bar non-terminals. In our case, with 20 non-terminals this is only 1600 rules all together, an entirely manageable figure. Unfortunately, these rules are a disaster for parsing, as they drive it to maximum expense and minimum information return. That is, the rules compete with each other for probability, and there is no structural advantage to using one over another. Any time one has $\overline{\mathsf{X}}$ followed by $\overline{\mathsf{Y}}$ in the chart, this could be dominated by either a $\overline{\mathsf{X}}$ or a $\overline{\mathsf{Y}}$. If the two possible adjunctive rules have equal probability, then the inside probability of the dominating $\overline{\mathsf{X}}$ and $\overline{\mathsf{Y}}$ will be exactly the same. Their outside probabilities may, of course, be different, but this sort of competition is both more expensive to compute and less effective at furnishing information for revising the rule probabilities.

Adjunctive rules look like good candidates for another set of constraints. As with our existing constraints, there are pretty good guesses for which non-terminals should be allowed to modify which others. Or, like conjunctive rules, one might even generate them offline, selecting some fixed set of candidates, and let the data sort it out. Both of these require substantial more hours of work by an expert; such experiments would provide another interesting data point, although they would not be more definitive. We decided to draw the line in how much expert-supplied information we would use, and we rejected generating adjunct rules.

Since adjunct rules were not used, we did not seek to apply the severe length restrictions that go hand-in-hand with them. Standard $\overline{\mathsf{X}}$ limits the number of specifiers (and complements) to one or two. Since our rules had to cope with modifiers as well, we placed no limit on their length besides the preferences used for our dependency grammars. This choice was also a deliberate attempt to keep our $\overline{\mathsf{X}}$ close to our dependency grammars, giving them similar flexibility in how they handle modifiers.

The rest of our decisions—splitting the bias, eliminating S=, massaging the constraints, and modifying the rule generator are all bound together. It simply won't do to have a rule generator constructing double-bar rules in specifier/complement format while the bias grammar uses some other form; the two will be unable to work together. Since $\overline{\mathsf{X}}$ provides enough parameters to handle sentence phrases, it seemed better to eliminate S= from both the bias and the generator, and give $\overline{\mathsf{X}}$ a chance to handle these in its own fashion. *Not* eliminating S= would have rendered $\overline{\mathsf{VB}}$, $\overline{\mathsf{AUX}}$, etc. superfluous if no rules were generated for them, or antagonistic to S= if they were. In the latter case, they would be duplicating the structure of S= rules, and hence competing with them. In short, we tried to make all of the alterations that appeared interdependent, and no more.

	cross entropy	coverage %	lambda	size
standard	2.7006	99.7	0.7266	4176
$\overline{\mathsf{X}}$	2.79235	99.26	0.3035	1626
dep gram - S= rules	2.71332	99.52	0.678	4361
doubled penalty	2.7919	99.26	0.304	1630

Figure 8.3: $\overline{\mathsf{X}}$ Performance results

8.4.4 Results

While we attempted to conserve the known good performance of our dependency grammars, figure 8.3 shows we failed. Figures for the grammar, labelled $\overline{\mathsf{X}}$ in the figure, show the cross entropy was .09 bits/word higher, which is disastrous. Our experiments with specifier/complement rules and conjunctions show that these presented no intrinsic problems. Assuming those penalties were approximatly cumulative, they would account for only .022 bits/word. The remainder must be the responsibility of the bias grammar, the constraints, the rule generator, or some interaction of all three.

There are two "superficial" explanations for the poor performance. First, the elimination of S= from the bias grammar and constraints may have penalized the grammar more than we estimated, despite our efforts to preserve the probabilities. By substituting in $\overline{\mathsf{VB}}$, $\overline{\mathsf{AUX}}$, et al, we permitted strings which might previously be impossible, since now any $\overline{\mathsf{AUX}}$ rule is permissible, whereas before only those headed by S= were. We can make a weak test of this theory in the same way we have tested others. We convert a finished, learned dependency grammar using the same process used to convert the bias grammar. The resulting grammar has penalties of .0128 bits/word, .2% coverage, and .0482 lambda. While these are significant penalties, it still brings the total cross entropy penalty from the various pieces to only half of the total performance loss seen.

Second, the rule length penalties which were tuned for dependency grammars may have failed for $\overline{\mathsf{X}}$ format. On average, one expects these rules to be shorter by half, but changing the penalty formula to $10^{[(2*\text{length})-1]}$ produces no substantial change, as documented in the "doubled penalty" row in figure 8.3. It is possible that there is some systematic difference in the length of specifier and complement rules, but we did not explore this.

There is no way to disentangle the bias, constraints and rule generator, and evaluate their performance independently. Hence, there is no way to determine which is responsible, individually, or in cooperation with some others, for the performance loss. The best we can say is that the overall performance loss appears to be a combination of small penalties for each aspect of $\overline{\mathsf{X}}$ together with an overall degradation from the combination and a lack of what has been termed "the TLC factor". That is, if we were committed to $\overline{\mathsf{X}}$ as our working formalism for the future, we could probably develop a combination of new parameters and better settings for old ones which would recover some or all of the missing performance. It looks unlikely that we will see much improvement, either in lower cross entropy, a much smaller grammar, or a simpler learner, so we have little motivation to adopt $\overline{\mathsf{X}}$.

There is one aspect of $\overline{\mathsf{X}}$ which we have neglected so far, but is worth mentioning before we drop the subject entirely. $\overline{\mathsf{X}}$ claims that there are similarities that cut across groups of rules headed by different non-terminals. This suggests a number of possibile strategies for broader generalization than we have used. E.g., rule preferences should apply across multiple categories. Moreover, a new rule could suggest similar rules headed by other non-terminals, and the presence or absence of these other rules might be used as part of the evaluation of candidate rules.

There are several difficulties standing in the way of realizing these possibilities. First, there will be more parameters to be set. The system must learn or be told which categories resemble which others; some quantifiable measure of similarity must be invented; some uses of similarity such as those above must be developed; and there are some technical hangups to be addressed. Some non-terminals span more than one syntactic category (e.g., $\overline{\overline{\mathsf{TO}}}$), and so provide weaker generalization possibilities. Other generalizations may be permitted or forbidden due to exceptions sanctioned by $\overline{\mathsf{X}}$ itslef.

Such generalizations seem best suited to a somewhat different system than our own. In particular, they fit almost perfectly into Hindle's requirements [22] for heuristics to generalize from existing rules for a lexicalized grammar. Our own grammars are not lexicalized, nor do they have enough non-terminals for us to expect a great deal of performance profit from the necessary development work.

Chapter 9

System Premises, Assumptions and Parameters

In this section we discuss the premises on which our scheme is based and the limitations which derive from them. The aim is to informally characterize both the conditions which must be met for our scheme to function and how much work is required to extend it in various directions. We begin with the premises and what these directly imply in terms of performance limitations. Next we turn to the requirements or assumptions we make about the input to our learner. Last we discuss re-targeting the learner to other languages.

9.1 System Premises

The essential premise of our system is that the target language has syntactic structure, so it can be modeled by a probabilistic context-free grammar whose rules reflect this structure. Recall that it is possible to construct grammar-based models that ignore syntactic structure and yet have minimal cross-entropy, regardless of whether the strings of the target language have any internal structure. The easiest way to do this is to write down a memorizing grammar, with one rule per sentence type. Memorizing grammars do not work well because they generalize poorly, and so perform poorly on test data. By aiming at modeling syntax, we

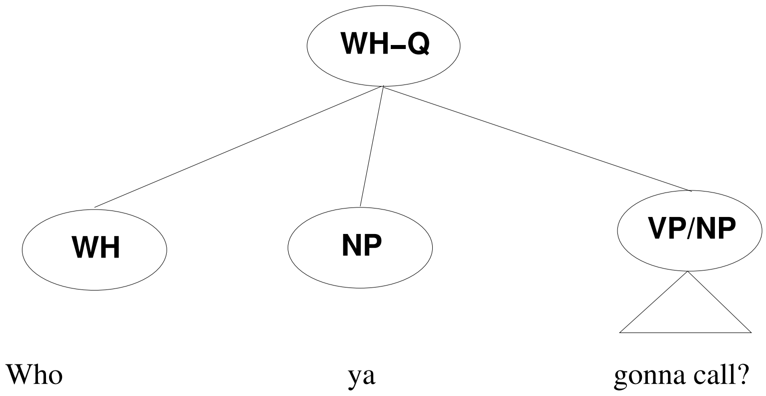

Figure 9.1: The standard strategy for handling wh-questions with slash categories.

take advantage of the fact that there are many syntactic structures per sentence, although they are not directly observable.

Since memorizing grammars are possible in the dependency grammar formalism, dependency grammars can model English if they are not required to capture syntax. It is, unfortunately, pretty clear that dependency grammars cannot do a perfect job modeling English syntactically. For example, one of the standard ways of handling wh-questions is to introduce the notion of "slash category", VP/NP, VP/PP, etc. Figure 9.1 shows an abbreviated example of a parse tree with slash-category non-terminals. Dependency grammars do not have enough non-terminals to include slash categories, so they cannot model them correctly. Instead, one must make do with a parse which does not take account of the fact that the verb phrase is missing its object, as in figure 9.2. The root non-terminal here is "$\overline{.}$" (read "period-bar"), which is used to head all sentences. As one can see, dependency grammars must use a single non-terminal, $\overline{\mathsf{VB}}$, to model both ordinary verb phrases (VP) and verb phrases missing a noun phrase ($\mathsf{VP/NP}$). The net effect of this is to superimpose the two distributions, which implies that dependency grammars will pay some penalty in terms of overall cross-entropy. The severity of the penalty will depend, in general, on how different the distribution is between rules headed by VP/NP and those headed by $\overline{VB}$.

Our belief is that the penalty is not severe, and dependency grammars are

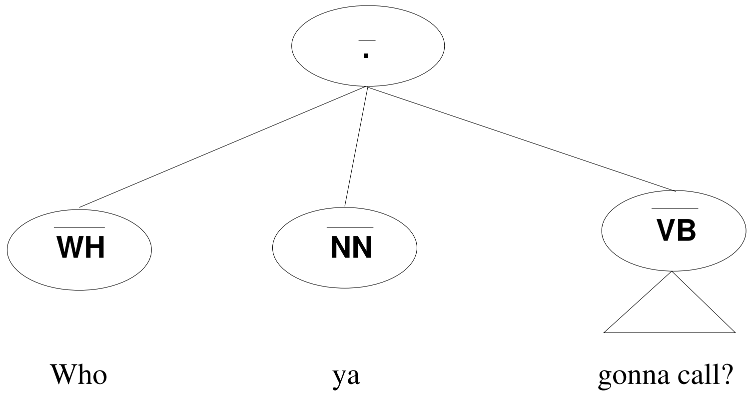

Figure 9.2: Handling wh-questions with dependency grammars.

adequate for most cases. As we showed above for embedded sentences, it is possible to extend the dependency grammar formalism in a controlled fashion, so as to address high-penalty shortcomings. We return to this when we discuss extensions in 9.3.

9.2 Input Assumptions

Modeling a language using a context-free grammar implicitly assumes that word order is important for the language's syntax. This is true for English and the Romance languages, French, Spanish, and Italian, but it is not true or less so for languages which rely on inflection to signal the syntactic role of words, e.g., German, Russian, and Latin. [1] The first assumption we make, then, is that the target language has some syntactic structure reflected in word order.

Given a suitable target language, we need to have this data tagged with part-of-speech information. This is a fairly mechanical issue, as automatic taggers for English are both publicly available and straightforward to construct, and taggers for other languages are beginning to appear [16,43–46]. These taggers typically have around 95-97% accuracy and use reasonably sized tag sets.

[1] That is, these languages logically permit many word orderings to reflect a single syntactic structure. Despite this license, a standardized word order is commonly used, perhaps enough to make some statistically significant inferences.

We say "reasonably sized" because gratuitously large tag sets imply a grammar that is more or less lexicalized, with the problems attendant on lexicalized grammars. Essentially, these boil down to a sparse data problem which is substantially worse than ours, and therefore requires alternative methods of generalization. (See, for example, [22].) We assume that the data represent at least most syntactic phenomena, which is not the case for lexicalized grammars.

A final soft requirement for the tagging process is that the tags generally ought to reflect meaningful syntactic categories. One might expect that datasets assembled with machine processing in mind would choose meaningful tags simply as a matter of course, but that does not so far seem to be the case. The Brown Corpus, for example, labels all occurrences of the word "this" as DET (determiner), even when it is clearly acting as a pronoun, as in

`Would you look at this?`

The Penn Treebank commits similar errors, e.g., labelling "to" as TO, whether used as a preposition or as an infinitive specifier.

These choices for tags are not so much wrong as unfortunate. Since "to" is not identified as a preposition, there must be two sets of rules for dealing with prepositions, one with all other prepositions, and one with the TO tag. This hurts generalization. Moreover, TO must head a set of rules which are a mix of tenseless verb phrases and noun phrases, which is overgeneralization. While our system can and does learn in spite of these handicaps, it would doubtless perform better without them.

9.3 Adapting the learner to other tasks

The learner we have built is aimed at the general English found in the Brown Corpus. For other target languages, some tuning will generally be necessary for optimal performance. The aim of this section is to point out what is likely to need adjustment and how these adjustments might be carried out.

The two "parameters" that most commonly require some adjustment are the bias grammar and its corresponding rule preferences. One reason for this is that different corpora commonly use different tag sets, and if no mapping can be established between the two, some massaging of rules and constraints will be necessary. It is also possible for the same corpus to be tagged more than once, using incompatible tag sets, in which case the same situation arises.

Generally, these adjustments should not be difficult to make. The bias grammar is, to begin with, just a skeleton, and quite broadly applicable. It is less than a hundred rules, written in a general PCFG format which is translated into the dependency grammar formalism. Usually only minor adjustments to this "meta-grammar" are necessary. The rule form preferences must also be adjusted to reflect a different tag set, but again this should also involve only localized changes, as certain terminal categories (noun, verb, and so on), are more or less universal. Grammars omitting these are unlikely to perform reasonably. Grammars adding thousands of non-terminals are likely to run into the lexicalization problems previously mentioned.

Domains with idiosyncratic English, such as computer manuals, children's stories or the like, are will also require adjustment of the bias grammar and constraints. If the idiosyncrasies are known in advance, then the bias may be adjusted immediately. If not, it is straightforward to run SINGER , evaluate the results with our analysis program, and then patch the grammar and/or constraints.

This tool computes the probability of each sentence according to both our grammar model and the competing trigram model, and prints out those sentences for which the grammar model's probability is much worse. If the learner has trouble with a specific terminal or syntactic structure, this is generally evident by inspecting the output. As a second source of information, we keep statistics over the terminals, indicating their frequency in troubled and untroubled sentences. Generally, this is adequate for uncovering localized problems, so it suffices for re-tuning to specific domains, curing problems with different tag sets, and the like.

Besides the bias and constraints, it may sometimes be necessary to adjust the formalism, generating rules which are not strictly in dependency grammar format. For example, we used our analysis tool to uncover the learner's shortcomings with respect to embedded sentences, and corrected the problem by extending the formalism. As in our example, such problems cannot in general be solved simply by tinkering with the grammar or the constraints. Rather, the rule generator must be modified, which involves writing code. Clearly this is not the sort of modification a user likes to face. There are, however, two ameliorating factors. First, for situations like ours, where the problem is specific, the modifications are straightforward and simple. Second, such modifications should seldom be required, since the formalism covers most of the syntax most of the time.

The scenarios above are concerned with re-targeting the learner to other English corpora, and it is assumed that these are not radically different from the general English corpus we have worked with. While this is all we envision our system working on, one can at least imagine deploying our scheme farther afield, on foreign languages or truly eccentric English. What would be required in that case? The primary requirements would be an original bias grammar and constraints, rather than a rewrite of the existing ones. Some alteration of the formalism might also be indicated, but this would be a matter for the analysis tool to show or deny. Building a bias grammar and constraints requires a competent linguist, familiar with the target language. While this requirement may seem onerous, it is nothing more than our original premise that linguistic bias is required for best performance. Absent such information, one may as well use a more generic learning algorithm such as Omohundro's [47,48], which uses only non-linguistic bias information. While this is unlikely to perform well for complex data, low bias learners are preferable when there is no useful insight available.

A careful reader will have noted that we do not suggest altering a number of parameters buried in our system, including the rule length parameters, and the threshold for pruning rules during generation and selection. While we can imagine theoretical scenarios which would mandate altering these values, it seems unlikely that any would arise in practice. For example, it seems unlikely that other corpora or even other natural languages have such systematically long rules that they would benefit from a longer length limit and/or a weaker penalty.

The thresholds governing rule pruning are in a similar situation, although their primary impact is on the speed of learning. If one raises the pruning threshold, removing all but the top one or two rules, this reduces the amount of learning that takes place on each sentence, which both drives up the frequency of learning events and encourages memorizing the data. Lowering the threshold decreases the number of learning events, but leads to poor cross entropy because many good rules are never generated. While minor benefits might be realized by altering these parameters to a small degree, more radical settings have generated only poor results for us.

Overall, we expect the only required tuning to take place in the bias and the constraints. This should be straightforward for a linguist familiar with the target material, together with modest assistance from our simple analysis tool. Our primary assumptions about the input are that it has some syntactic structure, and it is available in sufficient quantity for learning to take place.

Chapter 10

Future Directions

The next major landmark to achieve is to build a true grammatical *word* model, and not just a grammatical model for a language of part-of-speech tags. While there are several promising possibilities for developing further improvements in grammar learning, the effectiveness of our methods would be most compellingly demonstrated in a full model. We have already begun work in this area, with encouraging preliminary results. The first step towards a word model is extending our equations to define the probability of a word. Roughly speaking, the probability of a word w_i is conditioned on its part-of-speech category (its tag), the grammar rule which produced the terminal category, and the head word of the parent rule (a.k.a the governor). By "head word" we mean the English word which corresponds to the head tag of the parent rule, see figure 10.1. (Diagram "The submarine, which was in drydock last week, sank like a stone." here)

In this example, the head word for the rule $\overline{\mathsf{NN}}\rightarrow\overline{\mathsf{DET}}\mathsf{NN}\overline{\mathsf{WDT}}$ is "submarine". For rules headed by S =, we choose the main verb to be the head, so in this case the head word is "sank". Thus, the probability of "submarine" is conditioned in part on the appearance of "sank", a dependency which trigrams necessarily miss. (Actually, our probability equations are a little more complicated, and also allow us to capture the dependency of "drydock" on "submarine".) Prototyping runs with this system show that the word probabilities, $P(word|head-word)$ resemble bigram probabilities, at least to a first approximation, and also capture

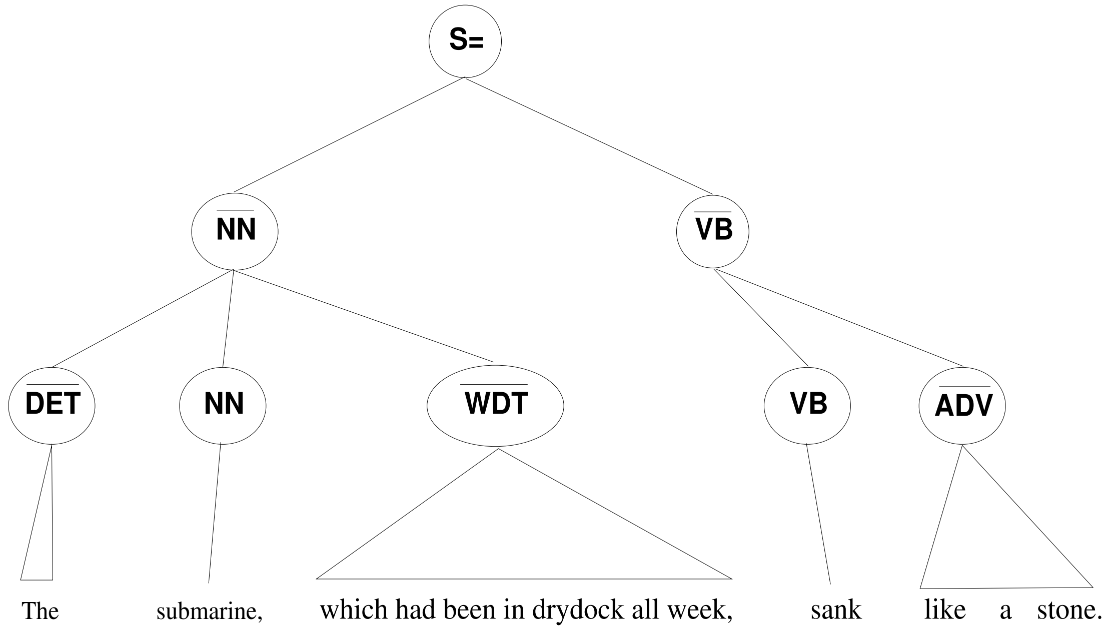

Figure 10.1: A poor sentence for trigrams.

dependencies, similar to the submarine example above, that would *not* be caught by either bigrams or trigrams.

This system is still undergoing shakedown tests on relatively small amounts of data (800,000 words). We have some 38 million words of text from the Wall Street Journal available for training, which should be enough to produce a language model beyond the current state of the art. Our hope is that the computation to learn the grammar and model can be made cheap enough, and the performance gain large enough, that trigram models can be laid to a well-deserved rest.

Chapter 11

Conclusions

In this dissertation we present a scheme for building a grammar-based language model. We draw our motivation both from the inadequacy of the trigram models from a linguistic point of view and from the challenge posed by the long-enduring success of such models. Since grammatical models are drawn from a much larger space than trigram models, they require some additional help for successful learning. We have made use of linguistic knowledge in several forms as the lever to make grammatical model learning practical. Our results show that one can learn a complex target grammar, with coverage $> 99\%$ and low cross entropy, without requiring excessive amounts of data or computation. The scheme is practical and performance of the learned model exceeds that of a trigram model, trained and evaluated over the same data. Finally, the stand-alone learning strategies reported may be used by grammar learners with other goals such as [24,49].

For linguists, our work implies that there is a useful contribution which symbolic linguistic knowledge can make to statistical models. Further, linguistic theories can be fleshed out and tested by statistical methods. Our evaluations of SINGER and particularly the $\overline{\mathsf{X}}$ variants show one way this might be done. For statistical NLP workers, our work means that there is hope for great improvement over the current models, in part by direct application of linguistic methods and in part by using linguistics to gain insight into and control over the models that are developed. While a 3,000 rule grammar such as ours is not immediately understandable, we can understand the parses this grammar assigns to particular sentences, and so we can analyze "problem" sentences—those with excessively low probability—in a way not possible for trigram models.

The amalgamation of traditional linguistic technology with statistical techniques is currently an area of great interest (cf. the proceedings of ACL '94 [50] and its associated symposium [51]). Our contribution has been to show that information from a traditionally symbolic field (linguistics) can be usefully applied to a traditionally statistical problem (language modeling). E.g., we have shown that linguistically motivated constraints can be used as a partial cure for overfitting. Others such as Magerman [25] have taken the converse step of bringing statistical methods (decision trees) to parsing. Both Magerman's and our results show improvements over the current state of the art. Together they constitute a fairly strong argument that symbolic and statistical approaches have complementary strengths, at least for natural language domains, and hybrid systems are preferable to the alternatives.

Appendix A

Terminals

This is the reduced set of tags used in our experiments with split-corpus learning.

, comma

. sentence delimiter, including "!", "?", etc.

ABX "both", "either", "neither"

ADJ adjective

ADV adverb

AND both coordinating conjunctions such as "and", "but", etc, and subordinating conjunctions such as "if" and "although"

AUX auxiliary verbs, "to be", "to have" and modals.

AUXG the "ing" forms of $\overline{\mathsf{AUX}}$ verbs

DET determiners such as "the" and "a" but also words such as "which" or "that", whether appearing as a determiner ("which ball?") or relative pronoun ("the ball that I threw")..

NN nouns, including proper nouns and plural nouns

PREP prepositions

PRON pronouns

TO the infinitive specifier, not the preposition

VB verbs

VBG the "ing" form verbs

VBN the past participle form of verbs, e.g. "baked" or "broken".

WDT "who", "which"

WRB "what", "where"

S= sentence phrase, for handling relative clauses and so on. This is a non-terminal that has no corresponding terminal.

S reserved start symbol for the grammar.

Appendix B

The Tag Map

This is the map used to reduce the Brown Corpus tag set down to our own. Where words had multiple tags (e.g. "he's", tagged PRON+AUX), we treated them as if they were several words, each with one tag. The left column shows the original Brown Corpus tag, and the right shows the corresponding tag in our set.

.	.
,	,
ABL	ADV
ABN	ADV
ABX	ABX
AP	ADJ
AP$	ADJ
AT	ART
BE	AUX
BED	AUX
BEDZ	AUX

BEG	AUXG
BEM	AUX
BEN	AUX
BER	AUX
BEZ	AUX
CC	AND
CD	NN
CD$	ADJ
CS	AND
DO	AUX
DOD	AUX
DOZ	AUX
DT	DET
DT$	ADJ
DTI	DET
DTS	DET
DTX	DET
EX	PRON
HV	AUX
HVD	AUX
HVG	AUXG
HVN	AUX
HVZ	AUX
IF	AND
IN	PREP

JJ	ADJ
JJR	ADJ
JJS	ADJ
JJT	ADJ
MD	AUX
NN	NN
NN$	ADJ
NNS	NN
NNS$	ADJ
NP	NN
NP$	ADJ
NPS	NN
NPS$	ADJ
NR	NN
NR$	ADJ
NRS	NN
OD	ADJ
PN	PRON
PN$	ADJ
PP$	ADJ
PP$$	PRON
PPL	PRON
PPLS	PRON
PPO	PRON
PPS	PRON

PPSS	PRON
QL	ADV
QLP	ADV
RB	ADV
RB$	ADJ
RBR	ADV
RBT	ADV
RN	ADV
RP	ADV
TO	TO
VB	VB
VBD	VB
VBG	VBG
VBN	VBN
VBZ	VB
WDT	WDT
WP$	WDT
WPO	WDT
WPS	WDT
WQL	WDT
WRB	WRB

Appendix C

Bias Grammar

This is the 600-rule version of the bias grammar. The raw bias grammar is trained once, pruned to 600 rules, and scaled so that the count of the start rule is 4000.

```
1.000000  S -> ._  (4000.000000)
1.000000  ,_ -> ,  (163.612565)
0.850782  ._ -> S= .  (3383.447226)
0.041168  ._ -> VB_ .  (163.720411)
0.037748  ._ -> NN_ .  (150.117039)
0.010294  ._ -> PREP_ .  (40.936494)
0.007570  ._ -> ADV_ .  (30.104712)
0.006683  ._ -> AUX PRON_ VB_ .  (26.578309)
0.004263  ._ -> AUX_ .  (16.954890)
0.003338  ._ -> WRB_ AUX PRON_ VB_ .  (13.275512)
0.003205  ._ -> VBN_ .  (12.744375)
0.002962  ._ -> ADJ_ .  (11.780105)
0.002567  ._ -> WDT_ .  (10.209736)
0.002095  ._ -> VBG_ .  (8.332192)
0.001899  ._ -> WDT AUX PRON_ VB_ .  (7.553207)
0.001513  ._ -> AUX NN_ VB_ .  (6.015224)
0.001358  ._ -> WDT AUX PRON_ VBG_ .  (5.399414)
0.001192  ._ -> WDT AUX NN_ PREP_ .  (4.741382)
0.001137  ._ -> AUX PRON_ VBN_ .  (4.522502)
```

```
0.001086  ._ -> AUX PRON_ AUX_ .  (4.319643)
0.001070  ._ -> PRON_ .  (4.254193)
0.001039  ._ -> AUX PRON_ VBG_ .  (4.133651)
0.001027  ._ -> AUX PRON_ NN_ .  (4.085648)
0.000744  ._ -> AUX NN_ AUX_ .  (2.958341)
0.000733  ._ -> WDT AUX PRON_ AUX_ .  (2.913121)
0.000635  ._ -> WRB_ AUX PRON_ AUX_ .  (2.523990)
0.000632  ._ -> WRB_ AUX NN_ VB_ .  (2.512306)
0.000578  ._ -> AUX PRON_ ADV_ VB_ .  (2.300170)
0.000551  ._ -> PREP_ AUX NN_ PREP_ .  (2.192097)
0.000549  ._ -> TO_ .  (2.184530)
0.000538  ._ -> AUX NN_ NN_ .  (2.140300)
0.000516  ._ -> AUX S= NN_ .  (2.051938)
0.000509  ._ -> AUX S= AUX_ .  (2.024562)
0.000507  ._ -> ADV_ AUX NN_ PREP_ .  (2.015453)
0.000462  ._ -> AUX NN_ PREP_ .  (1.835416)
0.000445  ._ -> WRB_ AUX NN_ AUX_ .  (1.768015)
0.000429  ._ -> AUX PRON_ S= .  (1.705636)
0.000418  ._ -> PREP_ AUX NN_ NN_ .  (1.664282)
0.000411  ._ -> AUX PRON_ ADJ_ .  (1.635728)
0.000393  ._ -> WDT AUX NN_ VB_ .  (1.561806)
0.000359  ._ -> AUX PRON_ PREP_ .  (1.427313)
0.000328  ._ -> AND_ AUX NN_ VB_ .  (1.304156)
0.000312  ._ -> AUX S= ADJ_ .  (1.242220)
0.000304  ._ -> AUX NN_ VBN_ .  (1.209303)
0.000303  ._ -> AUX DET_ VB_ .  (1.203306)
0.000273  ._ -> VB_ AND_ VB_ .  (1.084345)
0.000267  ._ -> AUX NN_ ADJ_ .  (1.063448)
0.000262  ._ -> PREP_ AUX NN_ VB_ .  (1.040564)
0.952961  ABX_ -> ABX  (20.892799)
0.047039  ABX_ -> ABX PREP_  (1.031285)
0.854640  ADJ_ -> ADJ  (3067.902183)
0.079526  ADJ_ -> ADJ ADJ_  (285.475611)
```

```
0.042713  ADJ_ -> ADV_ ADJ  (153.328258)
0.017750  ADJ_ -> ADJ AND_ ADJ_  (63.715979)
0.002583  ADJ_ -> ADV_ ADJ ADJ_  (9.272966)
0.001945  ADJ_ -> ADV_ ADJ AND_ ADJ_  (6.983366)
0.000842  ADJ_ -> ADJ ADJ_ AND_ ADJ_  (3.023479)
0.922863  ADV_ -> ADV  (1945.379731)
0.068726  ADV_ -> ADV ADV_  (144.873966)
0.007778  ADV_ -> ADV AND_ ADV_  (16.396410)
0.000633  ADV_ -> ADV ADV_ AND_ ADV_  (1.334187)
0.989026  AND_ -> AND  (1557.003951)
0.010974  AND_ -> AND AND_  (17.276154)
0.245753  AUX_ -> AUX VBN_  (710.859984)
0.148258  AUX_ -> AUX NN_  (428.848532)
0.112896  AUX_ -> AUX AUX_  (326.558925)
0.112386  AUX_ -> AUX VB_  (325.084670)
0.059081  AUX_ -> AUX ADJ_  (170.896779)
0.056837  AUX_ -> AUX VBG_  (164.404052)
0.036188  AUX_ -> AUX PREP_  (104.676776)
0.033472  AUX_ -> AUX  (96.819662)
0.026539  AUX_ -> AUX NN_ PREP_  (76.765750)
0.025874  AUX_ -> AUX ADJ_ PREP_  (74.842968)
0.014437  AUX_ -> AUX ADV_ NN_  (41.761528)
0.012313  AUX_ -> AUX TO_  (35.617212)
0.010821  AUX_ -> AUX ADV_ ADJ_  (31.299826)
0.009532  AUX_ -> AUX PRON_  (27.571989)
0.008806  AUX_ -> ADV_ AUX VBN_  (25.472751)
0.006795  AUX_ -> AUX S=  (19.656413)
0.005454  AUX_ -> ADV_ AUX NN_  (15.774700)
0.005292  AUX_ -> AUX NN_ ADV_  (15.307727)
0.005114  AUX_ -> AUX TO_ PREP_  (14.791433)
0.005074  AUX_ -> AUX ADV_ PREP_  (14.678019)
0.004629  AUX_ -> AUX ADV_ ADJ_ PREP_  (13.390785)
0.003966  AUX_ -> AUX PRON_ PREP_  (11.473281)
```

```
0.003363  AUX_ -> AUX AUXG_  (9.728162)
0.002710  AUX_ -> AUX WDT_  (7.838182)
0.002640  AUX_ -> AUX ADV_ NN_ PREP_  (7.635658)
0.002604  AUX_ -> ADV_ AUX VB_  (7.533560)
0.002512  AUX_ -> ADV_ AUX AUX_  (7.267448)
0.002467  AUX_ -> AUX VBN_ PREP_  (7.136570)
0.002297  AUX_ -> AUX DET_  (6.644143)
0.002187  AUX_ -> AUX ADV_ S=  (6.325614)
0.001796  AUX_ -> AUX ADJ_ ADV_  (5.195568)
0.001793  AUX_ -> ADV_ AUX ADJ_  (5.186403)
0.001584  AUX_ -> AUX ADV_ TO_  (4.580566)
0.001396  AUX_ -> AUX PREP_ ADV_  (4.038368)
0.001086  AUX_ -> AUX S= PREP_  (3.141511)
0.001079  AUX_ -> ADV_ AUX VBG_  (3.121222)
0.001042  AUX_ -> AUX PRON_ ADV_  (3.014915)
0.001022  AUX_ -> AUX ADJ_ AND_ VBN_  (2.955720)
0.000892  AUX_ -> ADV_ AUX PREP_  (2.580757)
0.000811  AUX_ -> AUX TO_ ADJ_  (2.346198)
0.000799  AUX_ -> AUX PREP_ AND_ AUX_  (2.312359)
0.000791  AUX_ -> ADV_ AUX TO_  (2.289396)
0.000743  AUX_ -> ADV_ AUX NN_ PREP_  (2.149149)
0.000647  AUX_ -> AUX ADJ_ AND_ VB_  (1.871497)
0.000624  AUX_ -> AUX ADV_ NN_ ADV_  (1.804689)
0.000602  AUX_ -> AUX TO_ ADV_  (1.740595)
0.000578  AUX_ -> AUX NN_ AND_ VB_  (1.672896)
0.000571  AUX_ -> AUX ADJ_ AND_ AUX_  (1.651820)
0.000564  AUX_ -> ADV_ AUX PRON_  (1.631185)
0.000514  AUX_ -> AUX ADV_ PRON_  (1.486489)
0.000509  AUX_ -> ADV_ AUX ADJ_ PREP_  (1.471725)
0.000455  AUX_ -> AUX S= ADV_  (1.316644)
0.000445  AUX_ -> AUX ADJ_ PREP_ ADV_  (1.286707)
0.000443  AUX_ -> AUX PREP_ AND_ VBG_  (1.281611)
0.000416  AUX_ -> AUX DET_ ADV_  (1.203510)
```

```
0.000412  AUX_ -> AUX NN_ PREP_ ADV_  (1.192013)
0.000410  AUX_ -> AUX TO_ PREP_ AND_ VB_  (1.184866)
0.000403  AUX_ -> AUX NN_ AND_ AUX_  (1.166267)
0.000398  AUX_ -> AUX VBN_ AND_ VBN_  (1.150256)
0.000383  AUX_ -> AUX VBG_ PREP_  (1.108641)
0.000379  AUX_ -> AUX WDT_ PREP_  (1.094865)
0.000347  AUX_ -> AUX NN_ AND_ VBG_  (1.005023)
0.537499  AUXG_ -> AUXG VBN_  (13.637280)
0.252313  AUXG_ -> AUXG NN_  (6.401624)
0.061680  AUXG_ -> ADV_ AUXG VBN_  (1.564936)
0.053333  AUXG_ -> AUXG ADJ_  (1.353145)
0.045539  AUXG_ -> AUXG NN_ PREP_  (1.155396)
1.000000  ART_ -> ART  (3463.678010)
0.931763  DET_ -> DET  (620.208166)
0.038170  DET_ -> DET PREP_  (25.407006)
0.012952  DET_ -> DET WDT_  (8.621428)
0.009414  DET_ -> DET VBN_  (6.266263)
0.006915  DET_ -> DET VBG_  (4.602757)
0.266791  NN_ -> NN  (2629.988843)
0.154548  NN_ -> ART_ NN  (1523.514716)
0.124072  NN_ -> ADJ_ NN  (1223.084231)
0.073972  NN_ -> NN_ NN  (729.207333)
0.059353  NN_ -> ART_ ADJ_ NN  (585.092574)
0.058692  NN_ -> ART_ NN PREP_  (578.575856)
0.047918  NN_ -> NN PREP_  (472.368153)
0.026805  NN_ -> DET_ NN  (264.239930)
0.024507  NN_ -> ADJ_ NN PREP_  (241.587639)
0.022753  NN_ -> ART_ ADJ_ NN PREP_  (224.295884)
0.017115  NN_ -> ART_ NN_ NN  (168.713508)
0.013953  NN_ -> NN AND_ NN_  (137.551257)
0.012625  NN_ -> NN_ NN PREP_  (124.454631)
0.007250  NN_ -> ADJ_ NN_ NN  (71.465751)
0.006594  NN_ -> ADJ_ NN AND_ NN_  (64.999956)
```

```
0.006175  NN_ -> ART_ NN AND_ NN_  (60.871741)
0.005173  NN_ -> DET_ ADJ_ NN  (50.994636)
0.005069  NN_ -> DET_ NN PREP_  (49.971379)
0.004085  NN_ -> NN_ NN AND_ NN_  (40.266181)
0.003657  NN_ -> ART_ ADJ_ NN_ NN  (36.047498)
0.003423  NN_ -> ART_ NN WDT_  (33.747029)
0.003090  NN_ -> ART_ NN_ NN PREP_  (30.456931)
0.003032  NN_ -> NN WDT_  (29.886853)
0.002848  NN_ -> ART_ NN PRON  (28.073880)
0.002435  NN_ -> ART_ NN PREP_ AND_ NN_  (24.001067)
0.002391  NN_ -> ART_ NN VBN_  (23.568134)
0.002037  NN_ -> NN VBN_  (20.084236)
0.001980  NN_ -> ART_ ADJ_ NN AND_ NN_  (19.521700)
0.001847  NN_ -> NN PREP_ AND_ NN_  (18.203868)
0.001728  NN_ -> NN VBG_  (17.031441)
0.001584  NN_ -> NN PRON  (15.612782)
0.001557  NN_ -> ART_ NN VBG_  (15.344916)
0.001556  NN_ -> ADJ_ NN WDT_  (15.343075)
0.001495  NN_ -> DET_ NN_ NN  (14.738586)
0.001365  NN_ -> ART_ ADJ_ NN WDT_  (13.455502)
0.001227  NN_ -> ABX_ NN  (12.091878)
0.001121  NN_ -> ART_ ADJ_ NN PREP_ AND_ NN_  (11.048484)
0.001115  NN_ -> DET_ ADJ_ NN PREP_  (10.987614)
0.001102  NN_ -> NN_ NN WDT_  (10.862464)
0.001071  NN_ -> NN_ NN VBN_  (10.562078)
0.001042  NN_ -> ADJ_ NN PREP_ AND_ NN_  (10.273853)
0.001017  NN_ -> ART_ ADJ_ NN VBN_  (10.027190)
0.000963  NN_ -> ART_ ADJ_ NN PRON  (9.490008)
0.000954  NN_ -> ADJ_ NN_ NN PREP_  (9.407248)
0.000863  NN_ -> ART_ NN PREP_ WDT_  (8.508396)
0.000852  NN_ -> ADJ_ NN VBN_  (8.398738)
0.000744  NN_ -> ART_ NN_ NN AND_ NN_  (7.331793)
0.000619  NN_ -> NN_ NN PRON  (6.100195)
```

```
0.000614  NN_ -> ART_ ADJ_ NN VBG_  (6.048192)
0.000589  NN_ -> ADJ_ NN PRON  (5.805236)
0.000585  NN_ -> ADJ_ NN VBG_  (5.769396)
0.000575  NN_ -> DET_ NN PRON  (5.670105)
0.000532  NN_ -> NN_ NN VBG_  (5.249150)
0.000507  NN_ -> DET_ NN AND_ NN_  (4.999477)
0.000501  NN_ -> DET_ NN WDT_  (4.941945)
0.000493  NN_ -> ART_ NN PREP_ VBN_  (4.857616)
0.000466  NN_ -> ART_ ADJ_ NN_ NN PREP_  (4.591102)
0.000459  NN_ -> NN PREP_ VBN_  (4.525229)
0.000455  NN_ -> NN PREP_ WDT_  (4.482735)
0.000450  NN_ -> NN_ NN PREP_ AND_ NN_  (4.433074)
0.000444  NN_ -> ART_ NN_ NN VBN_  (4.376251)
0.000408  NN_ -> ADJ_ NN_ NN AND_ NN_  (4.023287)
0.000376  NN_ -> ART_ NN PREP_ PRON  (3.701975)
0.000353  NN_ -> ART_ NN PREP_ VBG_  (3.477406)
0.000348  NN_ -> ART_ NN_ NN WDT_  (3.426794)
0.000290  NN_ -> ABX_ NN AND_ NN_  (2.863047)
0.000254  NN_ -> ADJ_ NN PREP_ WDT_  (2.504124)
0.000247  NN_ -> ART_ ADJ_ NN PREP_ WDT_  (2.436405)
0.000243  NN_ -> NN PREP_ VBG_  (2.397970)
0.000236  NN_ -> ART_ NN_ NN VBG_  (2.323114)
0.000219  NN_ -> DET_ ADJ_ NN_ NN  (2.155314)
0.000209  NN_ -> DET_ NN VBN_  (2.063213)
0.000203  NN_ -> DET_ NN VBG_  (1.998027)
0.000186  NN_ -> NN VBN_ AND_ NN_  (1.833021)
0.000170  NN_ -> NN_ NN PREP_ VBN_  (1.679255)
0.000164  NN_ -> DET_ NN_ NN PREP_  (1.613387)
0.000157  NN_ -> ART_ ADJ_ NN_ NN WDT_  (1.544059)
0.000152  NN_ -> ART_ ADJ_ NN_ NN AND_ NN_  (1.499829)
0.000152  NN_ -> NN PREP_ PRON  (1.499564)
0.000148  NN_ -> ADJ_ NN PREP_ VBN_  (1.462058)
0.000144  NN_ -> ABX_ NN PREP_  (1.415459)
```

```
0.000139  NN_ -> ART_ NN_ NN PREP_ AND_ NN_  (1.368582)
0.000133  NN_ -> DET_ NN PREP_ AND_ NN_  (1.310484)
0.000131  NN_ -> ART_ NN WDT_ AND_ NN_  (1.294966)
0.000127  NN_ -> ART_ ADJ_ NN PREP_ VBN_  (1.255405)
0.000127  NN_ -> ART_ ADJ_ NN PREP_ VBG_  (1.254757)
0.000122  NN_ -> ART_ ADJ_ NN PREP_ PRON  (1.197772)
0.000114  NN_ -> ADJ_ NN PREP_ VBG_  (1.123225)
0.000112  NN_ -> ART_ ADJ_ NN_ NN VBN_  (1.108684)
0.000105  NN_ -> DET_ ADJ_ NN PRON  (1.038345)
0.000103  NN_ -> ART_ NN_ NN PRON  (1.010557)
0.804072  PREP_ -> PREP NN_  (3236.915352)
0.047588  PREP_ -> PREP PRON_  (191.573484)
0.035272  PREP_ -> PREP VBG_  (141.993188)
0.028065  PREP_ -> ADV_ PREP NN_  (112.979201)
0.016955  PREP_ -> PREP S=  (68.253861)
0.010596  PREP_ -> PREP DET_  (42.654307)
0.010156  PREP_ -> PREP WDT_  (40.886480)
0.009873  PREP_ -> PREP NN_ PREP_  (39.743505)
0.008149  PREP_ -> PREP ADJ_  (32.804592)
0.006176  PREP_ -> PREP VBN_  (24.863389)
0.005984  PREP_ -> PREP NN_ AND_ PREP_  (24.090096)
0.004558  PREP_ -> PREP PRON_ PREP_  (18.349758)
0.002523  PREP_ -> ADV_ PREP PRON_  (10.156643)
0.001794  PREP_ -> PREP AUXG_  (7.220925)
0.001680  PREP_ -> PREP ADJ_ PREP_  (6.761607)
0.001098  PREP_ -> ADV_ PREP S=  (4.421476)
0.000799  PREP_ -> PREP VBG_ PREP_  (3.216459)
0.000577  PREP_ -> ADV_ PREP VBG_  (2.324077)
0.000565  PREP_ -> PREP S= PREP_  (2.273000)
0.000389  PREP_ -> ABX_ PREP NN_  (1.566642)
0.000386  PREP_ -> ADV_ PREP NN_ AND_ PREP_  (1.555206)
0.000375  PREP_ -> PREP S= AND_ PREP_  (1.510311)
0.000359  PREP_ -> ADV_ PREP NN_ PREP_  (1.443853)
```

```
0.000339  PREP_ -> PREP DET_ PREP_  (1.363403)
0.000321  PREP_ -> PREP VBG_ AND_ PREP_  (1.290832)
0.000314  PREP_ -> ADV_ PREP ADJ_  (1.262686)
0.000272  PREP_ -> ADV_ PREP WDT_  (1.096741)
0.984705  PRON_ -> PRON  (2543.890476)
0.008216  PRON_ -> PRON ADJ  (21.226414)
0.004209  PRON_ -> PRON AND_ NN_  (10.872817)
0.001139  PRON_ -> PRON ADJ AND_ NN_  (2.943684)
0.000911  PRON_ -> PRON AND_ PRON_  (2.354591)
0.000438  PRON_ -> PRON AND_ S=  (1.132619)
0.809313  TO_ -> TO VB_  (358.311518)
0.190687  TO_ -> TO AUX_  (84.424084)
0.204965  VB_ -> VB NN_  (646.322605)
0.140202  VB_ -> VB  (442.103497)
0.115206  VB_ -> VB PREP_  (363.283017)
0.051620  VB_ -> VB PRON_  (162.774483)
0.042882  VB_ -> VB ADV_  (135.222133)
0.035899  VB_ -> VB NN_ PREP_  (113.202040)
0.026489  VB_ -> VB S=  (83.527180)
0.023368  VB_ -> VB TO_  (73.688140)
0.021494  VB_ -> VB ADV_ PREP_  (67.778681)
0.019746  VB_ -> VB PRON_ PREP_  (62.265795)
0.016751  VB_ -> VB ADV_ NN_  (52.821177)
0.015954  VB_ -> ADV_ VB NN_  (50.308700)
0.015526  VB_ -> VB NN_ AND_ VB_  (48.957967)
0.013697  VB_ -> VB NN_ ADV_  (43.190798)
0.012284  VB_ -> VB ADJ_  (38.734913)
0.012092  VB_ -> VB NN_ TO_  (38.128558)
0.012086  VB_ -> VB AND_ VB_  (38.111609)
0.010080  VB_ -> VB PREP_ AND_ VB_  (31.786691)
0.009777  VB_ -> VB PRON_ ADV_  (30.828732)
0.008587  VB_ -> ADV_ VB PREP_  (27.077753)
0.008567  VB_ -> ADV_ VB  (27.015291)
```

```
0.007813  VB_ -> VB NN_ NN_  (24.637973)
0.007575  VB_ -> VB VBG_  (23.885400)
0.007173  VB_ -> VB PRON_ NN_  (22.618830)
0.005725  VB_ -> VB ADV_ AND_ VB_  (18.053223)
0.005619  VB_ -> VB S= PREP_  (17.717274)
0.005262  VB_ -> VB ADV_ TO_  (16.592241)
0.005049  VB_ -> VB PRON_ TO_  (15.920148)
0.004928  VB_ -> VB PREP_ ADV_  (15.541063)
0.004909  VB_ -> VB ADJ_ PREP_  (15.480897)
0.004772  VB_ -> VB VBN_  (15.048079)
0.004685  VB_ -> VB TO_ NN_  (14.771808)
0.004359  VB_ -> VB TO_ PREP_  (13.745515)
0.003592  VB_ -> VB WDT_  (11.325411)
0.003419  VB_ -> ADV_ VB NN_ PREP_  (10.780487)
0.003025  VB_ -> VB NN_ ADJ_  (9.538377)
0.002910  VB_ -> VB ADV_ ADV_  (9.175189)
0.002782  VB_ -> ADV_ VB PRON_  (8.771806)
0.002768  VB_ -> VB ADV_ PREP_ AND_ VB_  (8.726856)
0.002675  VB_ -> ADV_ VB ADV_  (8.436678)
0.002665  VB_ -> VB ADV_ NN_ AND_ VB_  (8.403324)
0.002614  VB_ -> VB DET_  (8.243824)
0.002585  VB_ -> VB S= ADV_  (8.151412)
0.002314  VB_ -> VB NN_ S=  (7.295698)
0.002260  VB_ -> VB NN_ PREP_ AND_ VB_  (7.127949)
0.002206  VB_ -> VB NN_ TO_ PREP_  (6.957362)
0.002194  VB_ -> VB ADV_ NN_ PREP_  (6.918289)
0.002186  VB_ -> ADV_ VB TO_  (6.892343)
0.002064  VB_ -> VB PRON_ S=  (6.509418)
0.002009  VB_ -> VB TO_ ADV_  (6.333579)
0.001967  VB_ -> ADV_ VB S=  (6.203612)
0.001945  VB_ -> VB ADV_ ADJ_  (6.132654)
0.001941  VB_ -> VB VBG_ PREP_  (6.119214)
0.001790  VB_ -> VB PRON_ AND_ VB_  (5.644336)
```

```
0.001744  VB_ -> VB PRON_ ADV_ PREP_  (5.500036)
0.001570  VB_ -> VB NN_ PRON_  (4.951698)
0.001477  VB_ -> VB S= NN_  (4.657366)
0.001349  VB_ -> VB PRON_ PREP_ AND_ VB_  (4.253927)
0.001293  VB_ -> VB NN_ AND_ VBG_  (4.076257)
0.001247  VB_ -> VB NN_ ADV_ PREP_  (3.933000)
0.001241  VB_ -> VB NN_ AND_ AUX_  (3.912335)
0.001221  VB_ -> VB NN_ NN_ PREP_  (3.850469)
0.001189  VB_ -> VB PRON_ WDT_  (3.748895)
0.001173  VB_ -> ADV_ VB ADJ_  (3.698840)
0.001135  VB_ -> VB NN_ ADV_ AND_ VB_  (3.579886)
0.001071  VB_ -> VB PRON_ PRON_  (3.376583)
0.001026  VB_ -> VB VBN_ PREP_  (3.235140)
0.001020  VB_ -> VB ADV_ PREP_ ADV_  (3.214968)
0.000949  VB_ -> VB S= AND_ VB_  (2.993576)
0.000931  VB_ -> VB ADV_ WDT_  (2.935164)
0.000917  VB_ -> ADV_ VB NN_ AND_ VB_  (2.891873)
0.000909  VB_ -> ADV_ VB ADV_ PREP_  (2.867224)
0.000902  VB_ -> ADV_ VB ADV_ NN_  (2.844044)
0.000880  VB_ -> ADV_ VB PRON_ PREP_  (2.776025)
0.000857  VB_ -> VB TO_ TO_  (2.702793)
0.000835  VB_ -> VB ADV_ NN_ NN_  (2.633086)
0.000817  VB_ -> ADV_ VB VBG_  (2.576283)
0.000804  VB_ -> VB ADV_ NN_ ADV_  (2.535664)
0.000792  VB_ -> ADV_ VB NN_ NN_  (2.497986)
0.000777  VB_ -> ADV_ VB WDT_  (2.450237)
0.000740  VB_ -> VB PRON_ ADJ_  (2.334473)
0.000733  VB_ -> VB NN_ AND_ VBN_  (2.311515)
0.000727  VB_ -> VB DET_ PREP_  (2.293745)
0.000726  VB_ -> VB ADJ_ PRON_  (2.290576)
0.000712  VB_ -> VB PREP_ AND_ VBG_  (2.243753)
0.000692  VB_ -> VB PRON_ ADV_ AND_ VB_  (2.182797)
0.000675  VB_ -> ADV_ VB AND_ VB_  (2.129689)
```

```
0.000674  VB_ -> VB TO_ PRON_  (2.125323)
0.000669  VB_ -> VB PRON_ NN_ AND_ VB_  (2.109457)
0.000668  VB_ -> ADV_ VB NN_ ADV_  (2.105041)
0.000666  VB_ -> ADV_ VB PREP_ AND_ VB_  (2.101563)
0.000664  VB_ -> VB NN_ NN_ ADV_  (2.092351)
0.000641  VB_ -> VB WDT_ PREP_  (2.022109)
0.000640  VB_ -> VB NN_ ADV_ NN_  (2.018476)
0.000636  VB_ -> VB PRON_ NN_ PREP_  (2.005166)
0.000617  VB_ -> ADV_ VB DET_  (1.946874)
0.000615  VB_ -> VB ADV_ TO_ PREP_  (1.938909)
0.000614  VB_ -> VB ADJ_ AND_ VB_  (1.935285)
0.000598  VB_ -> VB AND_ AUX_  (1.886136)
0.000584  VB_ -> VB NN_ S= PREP_  (1.842152)
0.000570  VB_ -> VB ADV_ S=  (1.796146)
0.000562  VB_ -> VB PREP_ AND_ VBN_  (1.773071)
0.000560  VB_ -> VB VBG_ AND_ VB_  (1.765187)
0.000551  VB_ -> VB ADV_ PRON_  (1.737883)
0.000541  VB_ -> VB PREP_ AND_ AUX_  (1.706242)
0.000526  VB_ -> VB TO_ AND_ VB_  (1.659580)
0.000505  VB_ -> ADV_ VB PRON_ ADV_  (1.591236)
0.000504  VB_ -> VB ADJ_ ADV_  (1.588287)
0.000499  VB_ -> VB TO_ NN_ PREP_  (1.572965)
0.000494  VB_ -> VB PRON_ TO_ PREP_  (1.558266)
0.000470  VB_ -> VB TO_ S=  (1.481714)
0.000453  VB_ -> VB NN_ ADV_ ADV_  (1.427571)
0.000451  VB_ -> VB ADV_ ADV_ PREP_  (1.422360)
0.000447  VB_ -> VB VBG_ AND_ VBG_  (1.408425)
0.000442  VB_ -> VB VBG_ ADV_  (1.395106)
0.000423  VB_ -> VB NN_ PREP_ ADV_  (1.335360)
0.000415  VB_ -> ADV_ VB ADJ_ PREP_  (1.308898)
0.000410  VB_ -> ADV_ VB PRON_ NN_  (1.292099)
0.000396  VB_ -> VB ADV_ AND_ AUX_  (1.248795)
0.000395  VB_ -> VB PRON_ ADV_ NN_  (1.246471)
```

```
0.000387  VB_ -> VB TO_ ADJ_  (1.219857)
0.000366  VB_ -> VB WDT_ PRON_  (1.154589)
0.000364  VB_ -> VB PRON_ PREP_ ADV_  (1.147351)
0.000355  VB_ -> VB S= PRON_  (1.120817)
0.000328  VB_ -> VB VBN_ AND_ VB_  (1.033043)
0.278251  VBG_ -> VBG NN_  (150.270721)
0.207060  VBG_ -> VBG  (111.824036)
0.105969  VBG_ -> VBG PREP_  (57.229272)
0.044926  VBG_ -> VBG TO_  (24.262365)
0.039443  VBG_ -> VBG ADV_  (21.301200)
0.033312  VBG_ -> VBG NN_ PREP_  (17.990227)
0.031134  VBG_ -> VBG PRON_  (16.813903)
0.025672  VBG_ -> ADJ_ VBG NN_  (13.864551)
0.017015  VBG_ -> ADV_ VBG NN_  (9.189081)
0.015192  VBG_ -> VBG ADV_ PREP_  (8.204626)
0.014970  VBG_ -> VBG ADV_ NN_  (8.084428)
0.014014  VBG_ -> ADV_ VBG  (7.568407)
0.011090  VBG_ -> VBG NN_ TO_  (5.989025)
0.010922  VBG_ -> VBG ADJ_  (5.898393)
0.009730  VBG_ -> VBG S=  (5.254999)
0.009564  VBG_ -> ADV_ VBG PREP_  (5.164878)
0.008837  VBG_ -> VBG PRON_ PREP_  (4.772568)
0.008687  VBG_ -> VBG NN_ NN_  (4.691684)
0.007710  VBG_ -> VBG AND_ VBG_  (4.163926)
0.007700  VBG_ -> VBG NN_ ADV_  (4.158427)
0.006420  VBG_ -> VBG PREP_ ADV_  (3.467158)
0.005314  VBG_ -> VBG PRON_ ADV_  (2.869619)
0.004849  VBG_ -> VBG NN_ ADJ_  (2.618473)
0.004768  VBG_ -> VBG TO_ PREP_  (2.574926)
0.004762  VBG_ -> ADJ_ VBG  (2.571758)
0.004032  VBG_ -> VBG ADV_ ADV_  (2.177746)
0.003312  VBG_ -> VBG NN_ AND_ VBG_  (1.788908)
0.002960  VBG_ -> VBG ADV_ ADJ_  (1.598321)
```

```
0.002916  VBG_ -> ADV_ VBG NN_ PREP_  (1.574677)
0.002895  VBG_ -> VBG NN_ S=  (1.563582)
0.002857  VBG_ -> VBG ADJ_ PREP_  (1.543052)
0.002852  VBG_ -> ADV_ VBG ADV_  (1.540388)
0.002802  VBG_ -> VBG PRON_ TO_  (1.513130)
0.002752  VBG_ -> VBG ADV_ AND_ VBG_  (1.486487)
0.002719  VBG_ -> ADV_ VBG TO_  (1.468587)
0.002667  VBG_ -> ADV_ VBG ADV_ PREP_  (1.440570)
0.002537  VBG_ -> VBG VBG_  (1.369963)
0.002521  VBG_ -> VBG VBN_  (1.361741)
0.002324  VBG_ -> VBG ADV_ NN_ PREP_  (1.254974)
0.002056  VBG_ -> VBG TO_ ADV_  (1.110576)
0.002014  VBG_ -> VBG S= PREP_  (1.087507)
0.001925  VBG_ -> VBG ADV_ TO_  (1.039546)
0.001867  VBG_ -> VBG S= NN_  (1.008193)
0.271966  VBN_ -> VBN PREP_  (285.361912)
0.214561  VBN_ -> VBN  (225.129242)
0.120390  VBN_ -> VBN NN_  (126.320232)
0.048387  VBN_ -> VBN TO_  (50.769984)
0.039573  VBN_ -> ADV_ VBN  (41.521942)
0.037809  VBN_ -> VBN ADV_  (39.671560)
0.036197  VBN_ -> ADV_ VBN PREP_  (37.979582)
0.023605  VBN_ -> VBN ADV_ PREP_  (24.767433)
0.016286  VBN_ -> VBN NN_ PREP_  (17.088600)
0.014503  VBN_ -> ADV_ VBN NN_  (15.217675)
0.013687  VBN_ -> VBN AND_ VBN_  (14.361271)
0.012492  VBN_ -> VBN PRON_  (13.106826)
0.010116  VBN_ -> VBN S=  (10.614153)
0.009705  VBN_ -> VBN TO_ PREP_  (10.182974)
0.007477  VBN_ -> VBN ADJ_  (7.844986)
0.006618  VBN_ -> VBN PRON_ PREP_  (6.943522)
0.006361  VBN_ -> VBN PREP_ ADV_  (6.673812)
0.006273  VBN_ -> VBN ADV_ NN_  (6.581881)
```

```
0.005952  VBN_ -> VBN NN_ ADV_  (6.244970)
0.005312  VBN_ -> VBN ADJ_ PREP_  (5.573932)
0.004659  VBN_ -> VBN NN_ NN_  (4.888464)
0.004629  VBN_ -> VBN NN_ TO_  (4.857516)
0.004561  VBN_ -> VBN PREP_ AND_ VBN_  (4.785883)
0.004054  VBN_ -> VBN ADV_ ADV_  (4.253269)
0.004043  VBN_ -> ADV_ VBN ADV_  (4.242157)
0.004025  VBN_ -> ADV_ VBN TO_  (4.223738)
0.003509  VBN_ -> VBN VBG_  (3.681965)
0.003341  VBN_ -> VBN PRON_ NN_  (3.505726)
0.002596  VBN_ -> VBN PRON_ ADV_  (2.723812)
0.002564  VBN_ -> VBN ADV_ AND_ VBN_  (2.690415)
0.002343  VBN_ -> VBN NN_ AND_ VB_  (2.457916)
0.002269  VBN_ -> VBN S= PREP_  (2.380596)
0.002219  VBN_ -> VBN PREP_ AND_ VB_  (2.328401)
0.002067  VBN_ -> VBN ADV_ TO_  (2.168923)
0.001996  VBN_ -> ADV_ VBN NN_ PREP_  (2.093892)
0.001735  VBN_ -> VBN AND_ VB_  (1.820900)
0.001676  VBN_ -> VBN VBN_  (1.758142)
0.001559  VBN_ -> VBN NN_ ADJ_  (1.636173)
0.001514  VBN_ -> VBN NN_ AND_ AUX_  (1.588355)
0.001477  VBN_ -> ADV_ VBN PREP_ AND_ VBN_  (1.550239)
0.001423  VBN_ -> VBN TO_ ADV_  (1.493227)
0.001404  VBN_ -> VBN PRON_ ADV_ NN_  (1.473580)
0.001372  VBN_ -> VBN PREP_ AND_ VBG_  (1.439653)
0.001365  VBN_ -> ADV_ VBN ADV_ PREP_  (1.432731)
0.001354  VBN_ -> VBN NN_ AND_ VBN_  (1.420821)
0.001354  VBN_ -> VBN TO_ NN_  (1.420691)
0.001302  VBN_ -> VBN NN_ S=  (1.366184)
0.001267  VBN_ -> VBN PREP_ AND_ AUX_  (1.329187)
0.001258  VBN_ -> VBN PRON_ TO_  (1.320032)
0.001185  VBN_ -> ADV_ VBN ADJ_ PREP_  (1.243632)
0.001178  VBN_ -> ADV_ VBN TO_ PREP_  (1.236154)
```

```
0.001133  VBN_ -> VBN ADV_ AND_ VB_  (1.188473)
0.001109  VBN_ -> VBN ADV_ ADJ_ PREP_  (1.163702)
0.001087  VBN_ -> ADV_ VBN NN_ NN_  (1.140176)
0.001081  VBN_ -> ADV_ VBN S=  (1.134263)
0.001021  VBN_ -> ADV_ VBN AND_ VBN_  (1.071777)
0.000965  VBN_ -> ADV_ VBN PRON_  (1.012925)
0.000960  VBN_ -> VBN PRON_ PRON_  (1.007181)
0.279857  WDT_ -> WDT VB_  (86.508741)
0.272637  WDT_ -> WDT S=  (84.277121)
0.228063  WDT_ -> WDT AUX_  (70.498255)
0.219443  WDT_ -> WDT  (67.833799)
0.991018  WRB_ -> WRB  (108.311518)
0.206436  S= -> NN_ AUX_  (901.303308)
0.171538  S= -> NN_ VB_  (748.938235)
0.147324  S= -> PRON_ AUX_  (643.218661)
0.147076  S= -> PRON_ VB_  (642.134303)
0.020925  S= -> DET_ AUX_  (91.357902)
0.019965  S= -> NN_ AUX_ AND_ S=  (87.165643)
0.019110  S= -> PRON_ AUX_ AND_ S=  (83.432709)
0.019083  S= -> NN_ VB_ AND_ S=  (83.315350)
0.017268  S= -> PRON_ VB_ AND_ S=  (75.390914)
0.014969  S= -> ADV_ NN_ AUX_  (65.355482)
0.012246  S= -> AND_ PRON_ AUX_  (53.465915)
0.011127  S= -> AND_ NN_ AUX_  (48.581305)
0.010921  S= -> S= AUX_  (47.679912)
0.009239  S= -> PREP_ NN_ AUX_  (40.337516)
0.009203  S= -> ADV_ PRON_ AUX_  (40.181420)
0.008650  S= -> ADV_ PRON_ VB_  (37.765405)
0.008442  S= -> S= VB_  (36.859998)
0.008399  S= -> PREP_ PRON_ AUX_  (36.671085)
0.008167  S= -> PREP_ NN_ VB_  (35.658612)
0.007866  S= -> ADV_ NN_ VB_  (34.342682)
0.007716  S= -> AND_ PRON_ VB_  (33.687290)
```

```
0.006158  S= -> DET_ VB_  (26.886121)
0.006034  S= -> WDT_ AUX_  (26.346460)
0.005365  S= -> AND_ NN_ VB_  (23.422362)
0.005088  S= -> NN_ VBN_  (22.214974)
0.005030  S= -> PREP_ PRON_ VB_  (21.959898)
0.004402  S= -> PRON_ AUX_ WRB_ S=  (19.217654)
0.004043  S= -> WDT_ VB_  (17.651681)
0.003584  S= -> PRON_ VB_ WRB_ S=  (15.649747)
0.003278  S= -> NN_ AUX_ WRB_ S=  (14.309874)
0.003126  S= -> VBG_ AUX_  (13.646604)
0.002948  S= -> TO_ AUX_  (12.871553)
0.002904  S= -> NN_ VB_ WRB_ S=  (12.676719)
0.002739  S= -> NN_ VBG_  (11.958899)
0.002308  S= -> VBG_ VB_  (10.078899)
0.001933  S= -> AND_ DET_ AUX_  (8.441301)
0.001900  S= -> AND_ PRON_ AUX_ AND_ S=  (8.296956)
0.001843  S= -> VBN_ AUX_  (8.047509)
0.001817  S= -> DET_ AUX_ AND_ S=  (7.933862)
0.001634  S= -> DET_ AUX_ WRB_ S=  (7.134074)
0.001493  S= -> ADV_ PRON_ VB_ AND_ S=  (6.516995)
0.001365  S= -> DET_ VB_ AND_ S=  (5.960789)
0.001354  S= -> AND_ PRON_ VB_ AND_ S=  (5.912528)
0.001218  S= -> ADJ_ AUX_  (5.317339)
0.001126  S= -> PRON_ VBN_  (4.917610)
0.001123  S= -> S= VBN_  (4.905010)
0.001119  S= -> AND_ WDT_ AUX_  (4.887013)
0.001070  S= -> ADV_ NN_ VB_ AND_ S=  (4.672971)
0.001063  S= -> ADV_ S= VB_  (4.642970)
0.001037  S= -> AND_ NN_ VB_ AND_ S=  (4.529568)
0.001009  S= -> AND_ NN_ AUX_ AND_ S=  (4.406383)
0.000973  S= -> AND_ ADV_ PRON_ AUX_  (4.249975)
0.000956  S= -> ADV_ PRON_ AUX_ AND_ S=  (4.175256)
0.000939  S= -> PRON_ VBG_  (4.101234)
```

```
0.000938  S= -> AND_ S= AUX_  (4.095177)
0.000934  S= -> AND_ PREP_ NN_ AUX_  (4.076502)
0.000830  S= -> AND_ ADV_ PRON_ VB_  (3.623087)
0.000828  S= -> S= VBG_  (3.613271)
0.000816  S= -> ADV_ S= AUX_  (3.564335)
0.000811  S= -> PREP_ TO_ AUX_  (3.541875)
0.000801  S= -> TO_ VB_  (3.495710)
0.000769  S= -> AND_ ADV_ NN_ AUX_  (3.355833)
0.000751  S= -> ADJ_ VBN_  (3.278496)
0.000735  S= -> AND_ PREP_ PRON_ AUX_  (3.209794)
0.000720  S= -> ADV_ DET_ AUX_  (3.144172)
0.000694  S= -> AND_ ADV_ NN_ VB_  (3.029937)
0.000671  S= -> ADV_ NN_ AUX_ AND_ S=  (2.930517)
0.000657  S= -> VBN_ VB_  (2.867646)
0.000615  S= -> ADV_ PREP_ NN_ AUX_  (2.683418)
0.000590  S= -> S= AUX_ AND_ S=  (2.573838)
0.000579  S= -> ADV_ PREP_ PRON_ AUX_  (2.527061)
0.000569  S= -> PREP_ NN_ AUX_ AND_ S=  (2.484695)
0.000565  S= -> WDT_ VBN_  (2.468628)
0.000561  S= -> PREP_ TO_ VBN_  (2.447556)
0.000545  S= -> S= VB_ AND_ S=  (2.379371)
0.000541  S= -> AND_ WDT_ VB_  (2.360353)
0.000523  S= -> PREP_ NN_ VB_ AND_ S=  (2.283950)
0.000523  S= -> ADV_ PREP_ PRON_ VB_  (2.281621)
0.000520  S= -> AND_ PREP_ PRON_ VB_  (2.271700)
0.000507  S= -> PREP_ TO_ VB_  (2.215414)
0.000483  S= -> PREP_ PRON_ VB_ AND_ S=  (2.106694)
0.000481  S= -> WDT_ VBG_  (2.101193)
0.000450  S= -> AND_ S= VB_  (1.964195)
0.000438  S= -> ADJ_ VB_  (1.910952)
0.000427  S= -> AND_ PREP_ NN_ VB_  (1.865315)
0.000423  S= -> ADV_ NN_ VBN_  (1.846369)
0.000393  S= -> PREP_ S= VB_  (1.716967)
```

```
0.000385  S= -> PREP_ NN_ VBN_  (1.681162)
0.000383  S= -> PREP_ S= AUX_  (1.670261)
0.000377  S= -> AND_ DET_ VB_  (1.644009)
0.000374  S= -> ADV_ WDT_ AUX_  (1.634292)
0.000310  S= -> PREP_ WDT_ VB_  (1.352052)
0.000301  S= -> NN_ AUXG_  (1.313147)
0.000299  S= -> PREP_ DET_ AUX_  (1.307497)
0.000298  S= -> ADJ_ AUX_ AND_ S=  (1.301118)
0.000292  S= -> PREP_ PRON_ AUX_ AND_ S=  (1.272818)
0.000290  S= -> WDT_ VB_ AND_ S=  (1.268201)
0.000287  S= -> ADV_ ADJ_ AUX_  (1.251243)
0.000271  S= -> DET_ VBN_  (1.182672)
0.000268  S= -> WDT_ AUX_ AND_ S=  (1.171175)
0.000256  S= -> NN_ VBN_ AND_ S=  (1.117305)
0.000249  S= -> PREP_ S= VBN_  (1.087931)
0.000246  S= -> AUXG_ AUX_  (1.073939)
0.000236  S= -> TO_ VBN_  (1.029051)
0.000236  S= -> ADV_ WDT_ VB_  (1.028672)
0.000233  S= -> ADV_ PREP_ NN_ VB_  (1.016112)
```

Appendix D

Special conjunction rules for $\overline{\mathbf{X}}$

These are additional the rules that do not fit the usual pattern of

$$\overline{\overline{\mathsf{X}}}\rightarrow(\overline{\overline{\mathsf{BTH}}})\ (\overline{\overline{\mathsf{X}}})^{*}\ \mathsf{AND}\ \overline{\overline{\mathsf{X}}}$$

$$\begin{array}{rcl}
\overline{\overline{\mathsf{PRON}}} & \rightarrow & \overline{\overline{\mathsf{PRON}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{NN}}} & \rightarrow & \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{PRON}}} \\
\overline{\overline{\mathsf{DET}}} & \rightarrow & \overline{\overline{\mathsf{DET}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{ADJ}}} & \rightarrow & \overline{\overline{\mathsf{ADJ}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VBN}}} \\
\overline{\overline{\mathsf{ADJ}}} & \rightarrow & \overline{\overline{\mathsf{ADJ}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{ADJ}}} & \rightarrow & \overline{\overline{\mathsf{ADJ}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VBG}}} \\
\overline{\overline{\mathsf{VBN}}} & \rightarrow & \overline{\overline{\mathsf{VBN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{ADJ}}} \\
\overline{\overline{\mathsf{NN}}} & \rightarrow & \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{ADJ}}} \\
\overline{\overline{\mathsf{VBG}}} & \rightarrow & \overline{\overline{\mathsf{VBG}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{ADJ}}} \\
\overline{\overline{\mathsf{VBN}}} & \rightarrow & \overline{\overline{\mathsf{VBN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{NN}}} & \rightarrow & \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{S}}} \\
\overline{\overline{\mathsf{WDT}}} & \rightarrow & \overline{\overline{\mathsf{WDT}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{S}}} & \rightarrow & \overline{\overline{\mathsf{S}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VB}}} \\
\overline{\overline{\mathsf{VB}}} & \rightarrow & \overline{\overline{\mathsf{VB}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{S}}}
\end{array}$$

$$
\begin{aligned}
\overline{\overline{\mathsf{S}}} &\rightarrow \overline{\overline{\mathsf{S}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{AUX}}} \\
\overline{\overline{\mathsf{AUX}}} &\rightarrow \overline{\overline{\mathsf{AUX}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{S}}} \\
\overline{\overline{\mathsf{ADV}}} &\rightarrow \overline{\overline{\mathsf{ADV}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{AUXG}}} \\
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{DET}}} \\
\overline{\overline{\mathsf{DET}}} &\rightarrow \overline{\overline{\mathsf{DET}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VBG}}} \\
\overline{\overline{\mathsf{VBG}}} &\rightarrow \overline{\overline{\mathsf{VBG}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{NN}}} &\rightarrow \overline{\overline{\mathsf{NN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VBN}}} \\
\overline{\overline{\mathsf{VBN}}} &\rightarrow \overline{\overline{\mathsf{VBN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{NN}}} \\
\overline{\overline{\mathsf{VB}}} &\rightarrow \overline{\overline{\mathsf{VBN}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{AUX}}} \\
\overline{\overline{\mathsf{AUX}}} &\rightarrow \overline{\overline{\mathsf{AUX}}}\ \ \mathsf{AND}\ \ \overline{\overline{\mathsf{VB}}}
\end{aligned}
$$

Bibliography

[1] A. Alshawi & et al., "Interim report on the sri core language engine.," Technical Report CCSRC-5, 1988.

[2] L. Hirschman, "Multi-site data collection for a spoken language corpus," *Proceedings of the February 1992 DARPA Speech and Natural Language Workshop* (1992).

[3] Kenneth W. Church & William A. Gale, "A Comparison of the Enhanced Good-Turing and Deleted Estimation Methods for Estimating Probabilities of English Bigrams," *Computers, Speech, and Language* 5 (1991).

[4] Yves Schabes, "Stochastic Lexicalized Tree-Adjoining Grammars," *Proc. of COLING-92* (1992).

[5] Fernando Pereira & Yves Schabes, "Inside-outside Reestimation from Partially Bracketed Corpora," in *Proceedings of ACL '92*, 1992, 128–135.

[6] Paul Placeway, Richard Schwartz, Pascale Fung & Long Nguyen, "The Estimation of powerful language models from small and large corpora," *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing* II (1993), 33–36.

[7] G. Sampson, "Evidence against the (un)grammaticality distinction," *Proceedings of the 7th Int. Conf. on English Language Research on Computerized Corpora*, Amsterdam (1987).

[8] Stuart Geman, Elie Bienenstock & Rene Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation* 4 (1992), 1–58.

[9] E. Mark Gold, "Language Identification in the Limit," *Information and Control* 10 (1967), 447–474.

[10] Dana Angluin, "On the Complexity of Minimum Inference of Regular Sets," *Information and Control* 39 (1978), 337–350.

[11] E. M. Gold, "Complexity of Automaton Identification from Given Sets," *Information and Control* 37 (1978), 302–320.

[12] Leonard Pitt & Manfred K. Warmuth, "The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial," *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, New York (1989).

[13] M. Kearns & L. G. Valiant, "Cryptographic limitations on learning Boolean formulae and finite automata," *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, New York (1989).

[14] Craig M. Cook, Azriel Rosenfeld & Alan R. Aronson, "Grammatical inference by hill climbing," *Information Sciences* 10 (1976), 59–80.

[15] J. Gerard Wolff, "Language Acquisition, Data Acquisition, and Generalization," *Language & Communication* 2 (1982), 57–89.

[16] Frederick Jelinek, "Markov source modeling of text generation," in *The Impact of Processing Techniques on Communications*, J. K. Skwirzinski, ed., Nijhoff, Dordrecht, 1985.

[17] J. K. Baker, "Trainable grammars for speech recognition," *Proceedings of the Spring Conference of the Acoustical Society of America*, Cambridge, MA (1979).

[18] J. Baker, "Trainable Grammars for Speech Recognition," in *Speech Communication Papers for the 97th Meeting of the Acoustic Society of America*, D. Klatt and J. Wolf, ed., ASA, 1982, 547–550.

[19] K. Lari & S. Young, "The estimation of stochastic context-free grammars using the Inside-Outside Algorithm," *Computer Speech and Language Processing* 4 (1990), 35–56.

[20] Michael R. Brent, "From Grammar to Lexicon: Unsupervised Learning of Lexical Syntax," *Computational Linguistics* 19 (1993), 243–262.

[21] R. C. Berwick & S. Pilato, "Learning Syntax by Automata Induction," *Machine Learning* 2 (1987), 9–38.

[22] Don Hindle, "Parsing a Probabilistic Dependency Grammar," *Proceedings of the AAAI-92 Fall Symposium: Probabilistic Approaches to Natural Language* (1992).

[23] Y. Schabes & R. Waters, "Stochastic lexicalized context-free grammar," in *Proceedings of the August 1993 International Workshop on Parsing Technologies*, 1993.

[24] Ted Briscoe & Nick Waegner, "Robust Stochastic Parsing Using the Inside-Outside Algorithm," in *Workshop notes for the Statistically-Based NLP Techniques Workshop*, 1992, 39–53.

[25] David Magerman, "Natural Language Parsing as Statistical Pattern Recognition," Stanford University, PhD thesis, 1994.

[26] C. L. Baker, *English Syntax*, MIT Press, Cambridge, MA, 1989.

[27] D. I. Slobin, "Cognitive prerequisites for the acquisition of grammar," in *Studies of Child Language Development*, C. A. Ferguson & D. I. Slobin, eds., Holt, New York, 1973.

[28] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of the IEEE* 64 (1976), 532–556.

[29] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai & Robert L. Mercer, "Class-based n-gram models of natural language," 1990.

[30] Lalit R. Bahl, Peter F. Brown, Peter V. deSouza & Robert L. Mercer, "A Tree-Based Statistical Language Model for Natural Language Speech Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37 (1989), 1001–1008.

[31] Glenn Carroll & Eugene Charniak, "Two Experiments on Learning Probabilistic Dependency Grammars from Corpora," Brown University Dept. of Computer Science, Brown University Technical Report No. CS-92-16, 1992.

[32] R. Jackendoff, $\overline{X}$ *syntax: A study in phrase structure.*, MIT Press, Cambridge, MA, 1977.

[33] John Lafferty, Daniel Sleator & Davy Temperley, "Grammatical Trigrams: A Probabilistic Model of Link Grammar," *Proceedings of the AAAI-92 Fall Symposium: Probabilistic Approaches to Natural Language* (1992).

[34] John Carroll & Ted Briscoe, "Probabilistic Normalization and Unpacking of Packed Parse Forests for Unification-based Grammars," in *Workshop notes for the Statistically-Based NLP Techniques Workshop*, 1992, 33–38.

[35] W. Nelson Francis & Henry Kučera, *Frequency Analysis of English Usage: Lexicon and Grammar*, Houghton Mifflin, Boston, 1982.

[36] David H. Wolpert, "Stacked Generalization," Los Alamas National Labs, Tech Report LA-UR-90-3460, 1990.

[37] G.E. Schulz, "Comparison of predicted and experimentally determined structure of adenyl kinase," *Nature* 250 (1974), 140–142.

[38] L. E. Baum, "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes," *Inequalities* III (1972), 1–8.

[39] David M. Magerman & Carl Weir, "Efficiency, robustness and accuracy in Picky chart parsing," in *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992, 40–47.

[40] Ezra Black, Fred Jelinek, John Lafferty, David Magerman, Robert Mercer & Salim Roukos, "Towards history-based grammars: using richer models for probabilistic parsing," in *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993, 31–37.

[41] Ted Dunning, "Accurate Methods for the Statistics of Surprise and Coincidence," *Computational Linguistics* 19 (1993), 61–74.

[42] Glenn Carroll & Eugene Charniak, "Learning Probabilistic Dependency Grammars from Labelled Text," *Proceedings of the AAAI Fall Symposium* (1992).

[43] Kenneth Ward Church, "A stochastic parts program and noun phrase parser for unrestricted text," in *Second Conference on Applied Natural Language Processing*, ACL, 1988, 136–143.

[44] Doug Cutting, Julian Kupiec, Jan Pedersen & Penelope Sibun, "A Practical Part-of-Speech Tagger," *Proceedings of the Third Conference on Applied Natural Language Processing* (1992).

[45] Ralph Weischedel, Marie Meteer, Richard Schwartz, Lance Ramshaw & Jeff Palmucci, "Coping with ambiguity and unknown words through probabilistic models," *Computational Linguistics* 19 (1993), 359–382.

[46] Steven J. DeRose, "Grammatical category disambiguation by statistical optimization," *Computational Linguistics* 14 (1988), 31–39.

[47] Stephen Omohundro, "Best-First Model Merging for Dynamic Learning and Recognition," *Advances in Neural Information Processing Systems* 4 (1992).

[48] Stephen Omohundro & Andreas Stolcke, "Hidden Markov Model Induction by Bayesian Model Merging," *Advances in Neural Information Processing Systems* 5 (1993).

[49] Yasubumi Sakakibara, Michael Brown, Rebecca C. Underwood, I. Saira Mian & David Haussler, "Stochastic Context-Free Grammars for Modeling RNA," UCSC, Technical Report UCSC-CRL-93-16, 1993.

[50] *Proceedings of the ACL Conference* (1994).

[51] "The Balancing Act," *Proceedings of the Workshop on Combining Symbolic and Statistical Approaches to Language* (1994).