Automatic Acquisition of Language Models for Speech Recognition

by

Michael Kyle McCandless

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1992

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author . . . .
Department of Electrical Engineering and Computer Science
May 12, 1994

Certified by . . . .
James R. Glass
Research Scientist
Thesis Supervisor

Accepted by . . . .
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Automatic Acquisition of Language Models for Speech Recognition

by

Michael Kyle McCandless

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis focuses on the automatic acquisition of language structure and the subsequent use of the learned language structure to improve the performance of a speech recognition system. First, we develop a grammar inference process which is able to learn a grammar describing a large set of training sentences. The process of acquiring this grammar is one of generalization so that the resulting grammar predicts likely sentences beyond those contained in the training set. From the grammar we construct a novel probabilistic language model called the phrase class n-gram model (PCNG), which is a natural generalization of the word class n-gram model [11] to phrase classes. This model utilizes the grammar in such a way that it maintains full coverage of any test set while at the same time reducing the complexity, or number of parameters, of the resulting predictive model. Positive results are shown in terms of perplexity of the acquired phrase class n-gram models and in terms of reduction of word error rates for a speech recognition system.

Thesis Supervisor: James R. Glass
Title: Research Scientist

Automatic Acquisition of Language Models for Speech Recognition

by

Michael Kyle McCandless

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis focuses on the automatic acquisition of language structure and the subsequent use of the learned language structure to improve the performance of a speech recognition system. First, we develop a grammar inference process which is able to learn a grammar describing a large set of training sentences. The process of acquiring this grammar is one of generalization so that the resulting grammar predicts likely sentences beyond those contained in the training set. From the grammar we construct a novel probabilistic language model called the phrase class n-gram model (PCNG), which is a natural generalization of the word class n-gram model [11] to phrase classes. This model utilizes the grammar in such a way that it maintains full coverage of any test set while at the same time reducing the complexity, or number of parameters, of the resulting predictive model. Positive results are shown in terms of perplexity of the acquired phrase class n-gram models and in terms of reduction of word error rates for a speech recognition system.

Thesis Supervisor: James R. Glass
Title: Research Scientist

Acknowledgments

I am indebted to many people who made this research possible. Foremost, I would like to thank Jim Glass, my thesis supervisor, for his guidance and long term vision, and patience with my impatience. Without him, this thesis would not have been possible.

I wish to thank everyone in the Spoken Language Systems group: first of all Victor Zue, for making all of this research possible and providing the environment and resources which have allowed me to explore my ideas; Stephanie Seneff for providing very stimulating and rewarding conversations about language modeling, as well as critical feedback on my thesis; Dave Goddeau for his insight and witty comments; Mike Phillips for his invaluable help with our recognizer; Eric Brill for patiently listening as I spoke my mind about finite state models; Joe Polifroni and Christine Pao for keeping the computers up and running and the disks empty; and Victoria Palay and Sally Lee for taking care of all sorts of administrative details. I would also like to thank all of the other students in our group for support and feedback and challenging discussions. Without the efforts of these people, this research could not have reached this point.

I would also like to thank all four of my parents, Mom, Dad, Vic and Helen, as well as my brothers Greg, Tim, and Cory and my sister Melanie, for supporting me and providing *everything* to bring me to where I now stand.

This research was supported by ARPA under Contract N00014-89-J-1332 monitored through the Office of Naval Research, in part by a research contract from NYNEX Science and Technology and through a grant from Apple Computer, Inc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we describe an algorithm which is able to automatically learn something about the structure of a language, given a large set of example "training" sentences. Automatic learning is attractive for many reasons. First, we would like to understand what sorts of language patterns a learning algorithm learns well and what patterns it has difficulty with. This process can give us insight into the nature of our own language facilities. Second, if an algorithm is able to learn some structure about language this acquired knowledge can be used to improve language recognition systems, e.g. speech recognition. At the same time it can save resources which must presently be employed to write grammars to model language structure.

This thesis will explore two themes. The first is a question of the properties of automatic learning of language. To address this we develop a grammar inference process which generalizes from the training set to predict the likely target language. This process of generalization is rewarding to watch — the algorithm makes intuitively satisfying choices. To better evaluate the output of this generalization phase we construct probabilistic models from the acquired grammars. By computing the cross-entropy of these acquired models and the true distribution of the language we are able to better evaluate the limits of the grammar inference process.

The second theme is the issue of practicality. If we have a system which is able to learn something about the structure of language we would like to use the acquired knowledge to improve language recognition applications. We therefore investigate practical issues about the language models, including the computational resources they consume and the effect on word error rates of a speech recognition system.

1.1 Related Efforts

Recent advances in technology have enabled the storage of very large text databases, which in turn has allowed previously infeasible (because of limits in both time and space) data-driven approaches to be more realistic. As a result, there have been a number of recent efforts towards empirically deriving some structure of human languages.

Most of the approaches have dealt only with word classes, not trying to acquire any phrase structure. Some of these approaches utilize a word class n-gram model [11] to evaluate the output, while others simply rely on linguistic knowledge to decide if the system is learning "correct" word classes. Brill and Marcus [9] used bottom up clustering to assign words to classes, using divergence as a similarity metric. Brown et al. [11] use minimal loss of mutual information as the criterion for merging word classes hierarchically. Jardino and Adda [24] used simulated annealing to divide words into classes in a top-down manner. On a similar vein of automatic acquisition, Brill [6] developed a rule based system which automatically acquires rules to decide how to tag a word with its linguistic part of speech, given the word's context.

The efforts to acquire phrase structure are fewer, and almost all of them are linguistically motivated: is it possible to automatically discover the "correct" way to **bracket** a sentence. Sentence **bracketing** is a first step in parsing the sentence – it assigns the phrasal structure to the sentence. The second step is to then tag each of the phrases with names denoting phrasal equivalence classes. Brill et al. [8] tested a metric based on mutual information to automatically parse sentences, starting from the parts of speech of the words. Brill [7] developed a system which learns a decision tree to bracket sentences. Pereira and Schabes [35] developed a modified version of the inside/outside algorithm [2, 29] to automatically parse test sentences, given a very small amount of training data. Bod [4] uses a monte-carlo based approach to bracket sentences.

The emphasis in these approaches is to "correctly" parse a sentence in the linguistic sense — i.e., not to decide whether a given sentence is legal or not, but to assume the sentence is legal, and find the correct parse for it. They are not trying to assign probabilities to test sentences, only to correctly derive their structure, which makes it difficult to extend them to language models for speech recognition.

1.2 Thesis Outline

All of the experiments described in this thesis are conducted within the ATIS domain, which is a common evaluation domain in the ARPA speech understanding community [23]. ATIS contains flight information queries solicited from users who solved problem scenarios. For example, these are some typical sentences: "what is the earliest flight from boston to pittsburgh tomorrow afternoon", or "could you tell me if american airlines flight two oh one serves breakfast".

All of the results presented in this thesis are trained on the same training set and tested on the same test set. The training contains 9711 sentences, and the test set contains 1595 sentences. These two sets are entirely independent. The lexicon used to represent these sets has 841 words, including one "unknown word" to which all words not in the lexicon are mapped.

The chapters in this thesis are structured as follows. Chapter 2 describes relevant background information for this thesis: formal language theory, language modeling, including current approaches and means of evaluation, and speech recognition. Chapter 3 describes the inference process which produces a stream of **acquired** grammars from the training set of sentences. In this chapter we will show example acquired grammars, and evaluate the grammars by measuring test set coverage and other qualities of the grammars. These acquired grammars are then used to develop a novel probabilistic language model called the Phrase Class n-gram model, which is described in Chapter 4. In this chapter we will further evaluate the acquired grammars by plotting the perplexity and complexity of the resulting PCNG models. Using perplexity we can select the "best" grammar from the sequence and measure useful quantities for these grammars as we test different inference runs. Chapter 5 will describe how the PCNG model can be integrated with a speech recognition system, and will examine some computational implications and the impact on word error rate. Chapter 6 will conclude the thesis and describe some directions for future work.

Chapter 2

Background

2.1 Introduction

This chapter provides some background information to place the results of this thesis in context. Formal language theory will be described, including theoretical notions of language, natural languages and sub-languages, and probabilistic frameworks for languages. The speech recognition problem will be formulated in a Bayesian manner, which is the basis of present day speech recognition systems. This will place the field of language modeling in context.

We will address some issues behind language modeling, including the different ways of evaluating language models – both subjectively and objectively. Then we will give an overview of three broad classes of present day approaches towards language modeling – n-gram based models, decision tree models, and structured models.

2.2 Formal Language Theory

The definitions and formal properties of languages have been studied in great detail by the theoretical community. In this section we review some of these definitions and describe those properties that are relevant to this thesis.

A **lexicon** is a set of all possible words, also called terminal symbols, that a person may speak. A **sentence** is just a finite sequence of words chosen from the lexicon; there are a countably infinite number of sentences. A **language** is a subset of all sentences, and may be either finite or infinite. There are an uncountably infinite number of languages because the total number of languages is the number of subsets of a countably infinite number of sentences. A **sub-language** is a subset of a particular language.

Languages can be grouped into different classes of complexity. For example, the **finite** languages are languages containing only a finite number of sentences. The class of **regular** languages, which contains all finite languages as a subclass, consists of all languages which can be accepted by a finite state machine *or*, alternatively, denoted by a regular expression. Regular languages may contain an infinite number of sentences – for example $(01)^*$ is a regular language containing $\{\Lambda, 01, 0101, 010101, 01010101, ...\}$ where Λ is the empty sentence (one containing zero words).

Another class of languages is the **context-free** languages, so called because they are the set of languages which can be described, or denoted, by a **context free grammar** (CFG) G. The class of context-free languages properly contains the regular languages. For example, the language $a^N b^N$ (i.e., $\{\Lambda, ab, aabb, aaabbb, aaaabbbb, ...\}$, or those strings starting with a certain number of a's and ending with the same number of b's) is an example CFL which is not regular.

A CFG consists of a finite number of **rewrite rules**, or just rules for short, the lexicon of words, and an implicit lexicon of **non-terminal** symbols. As a matter of convention, words are spelled with lowercase letters, and non-terminals are spelled with uppercase letters. A rule is of the form $N \Rightarrow X$, where N is a single non-terminal, and X is a string of one or more non-terminals and terminals. Rules that share same non-terminal on the left may be abbreviated: $N \Rightarrow X|Y$ is the shorthand for the two rules $N \Rightarrow X$ and $N \Rightarrow Y$. A distinguished non-terminal, S, is the **start** symbol, from which all sentences in the language are derived. Figure 2-1 shows a simple CFG which accepts the language of zero or more a's followed by one or more b's.

The grammar uniquely denotes a language, referred to as $\mathbf{L(G)}$, by specifying implicitly every sentence in the language. This implicit specification is by means

$$S \Rightarrow AB|B$$

$$A \Rightarrow a|Aa$$

$$B \Rightarrow b|Bb$$

Figure 2-1: Example context free grammar

of derivation: if the grammar can **derive** a sentence then that sentence is in the language denoted by the grammar. This derivation begins with the start symbol S, and proceeds iteratively by replacing non-terminals in the string, one at a time, until the string contains only terminals. Thus, if there is a rule $S \Rightarrow AB$, then S can be replaced with AB. We can then similarly replace A or B. There can be a number of choices as to which non-terminal to replace at any given step, as there may be more than one non-terminal in the partially reduced string, or more than one rule for each non-terminal. By varying which replacements are chosen, all sentences in the language may be derived from the start symbol S. This is referred to as using the grammar generatively because, with this process, all sentences in the language denoted by the grammar can be generated. It is because the context in which the non-terminal occurs is ignored during this derivation process that this class of languages is referred to as context-free. For example, to use the grammar in Figure 2-1 to derive $aaabb$, one possible derivation is $S \Rightarrow AB \Rightarrow AaB \Rightarrow AaaB \Rightarrow aaaB \Rightarrow aaaBb \Rightarrow aaabb$.

The grammar may also be used in the other direction: given a sentence, find the sequence of reductions which, starting with S, would result in the provided sentence. This process is referred to as **parsing**, and there are well known search algorithms to implement it [1, 15]. The output can then be represented as a parse tree, as shown in Figure 2-2.

It is possible for a given sentence to have more than one derivation with respect to a grammar. In such cases the grammar is said to be **ambiguous**. In many natural language applications, ambiguity is difficult to avoid, and presents problems in using the grammars to model natural languages.

Because a language can be very much larger than the grammar being used to

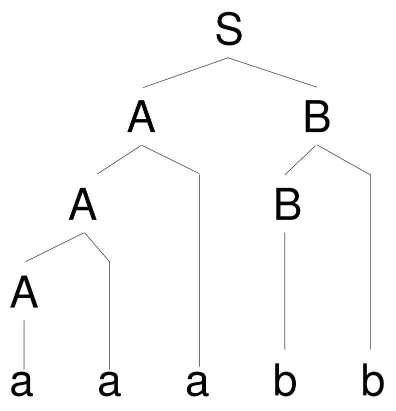

Figure 2-2: Parse tree for *aaabb*

denote it (even infinite), grammars are a compact means of denoting, or referring to, a language. One difficulty, however, is that for a given context free language there are many CFG's that will accept exactly that language — the correspondence between CFG's and context-free languages is a many-to-one mapping.

2.3 Natural Languages

Natural languages, and sub-languages of natural languages, are languages spoken by humans. Because a given natural language, for example English, is a language in the formal sense, it falls somewhere in the hierarchy of the formal languages. However, natural languages are *very* complex and have a number of properties which are beyond the power of context free languages. A simple example is number agreement, which is the requirement that the subject and verb of a sentence must agree on their number feature. The sentence "the girl selling cookies is a good friend of mine" is an example of such constraint. The subject "the girl" must agree with the verb "is" in number. Had the subject been "the girls" instead, the verb would have to be "are". These kinds of so-called "long distance" constraints can be arbitrarily far away in the sentence, and do not fit very well into the context free framework.

If we assume that we can *approximate* a natural language or sub-language as a CFL, then we can write a grammar trying to describe those sentences in the language.

In attempting to write such a grammar, we would like to have some means to evaluate how well we have done. The grammar can fail in two different manners. The first type of error, referred to as **under-generation**, occurs when the grammar fails to accept valid sentences of the sub-language we are trying to model. The second type of error is **over-generation**, which means our grammar actually derives many sentences which are not actually valid in the language.

The under-generation problem can be measured by computing what percentage of the sentences in an independent test set of "example" sentences can be parsed by the grammar. This is typically referred to as the **coverage** of the grammar on a particular test set. High coverage is desirable as this means the grammar does not suffer much from under-generation.

The problem of over-generation, however, is far more difficult to measure. We would like to measure how many of the sentences accepted by a grammar are not valid sentences, but this is not directly possible. This presents a difficult problem in developing grammars.

2.3.1 Spontaneous *vs.* Read Languages

Another difficulty with natural languages is the stark difference between proper sentences of the language, as one would read in a newspaper or magazine article, and spontaneous sentences which are typically spoken in a dialog between two people. The former version of the language is typically referred to as the **read** language, and the second as the **spontaneous** language. The grammatical rules of the language decide which sentences are well formed and which are not, but these formal rules are often broken or at least relaxed during spontaneous speech. Thus, for a given natural language, the spontaneous version can differ substantially from the read version.

2.4 Probabilistic Languages

The formal notion of a language as described in Section 2.2 needs to be extended somewhat for our purposes. Formally, a particular sentence is either in a given language or not — it is a strictly binary property for all sentences. But natural languages actually exhibit stronger patterns than this because some sentences are more likely or more frequent than others. We will therefore augment the formal notion of a language to associate probabilities with every possible sentence. If a particular sentence is not in the language, it has probability 0. We will refer to this distribution for a language as the **underlying** probability distribution.

We will model a language as a **stochastic source** which outputs sentences one word at a time in proportion to the underlying distribution on sentences. This word source could be thought of as first choosing a sentence, according to its distribution on all sentences, and then outputting that sentence one word at a time. A special marker, $\langle\rangle$, is used to denote the boundary between two sentences. There is, in general, substantial probabilistic dependence among the words.

We will assume that these sentences are probabilistically independent of one another (i.e., ignoring all dialog effects). To generate two sentences, then, the language source would generate the first, then generate the boundary marker $\langle\rangle$, then generate the second, then generate a second boundary marker. Because sentences are independent, the order in which those two sentences are generated does not affect the probability that they are generated. We could also "turn on" the source and have it generate one sentence, then turn it off and on again and have it generate the second, with the same probability.

We further assume that this probability distribution is **time-invariant**, which means that the probability distribution over all sentences does not change with time. Also, we assume the source generating sentences is **ergodic**, which simply means that if we were to run the source for a little while, the sentences that it produces during that time would look very much like the sentences it would produce if we ran it for a very long time.

These assumptions imply that we can collect many sentences from this source to create various independent sets of sentences for experimentation and that each such set produced will look somewhat similar. This source is typically actual human subjects who, when carefully coerced, will emit the sentences of the language we are trying to model. Our goal in attempting to model a target language is to build a model which comes as close as possible to the true underlying distribution for the language.

One interesting implication of this probabilistic formalism is that it is entirely reasonable for a language source to assign some non-zero probability to *every* possible sentence. The implication is that every sentence is actually possible, but just extremely unlikely. For example, consider the *plausible* but unlikely sentence "now i am going to say something totally random: orange eagle bagels exacerbate uncle fill". Some of the language models we will examine actually approximate a language source with such a distribution!

2.5 Speech Recognition

Speech recognition is fundamentally a classification problem. We observe small perturbations in air pressure varying with time, typically referred to as acoustic evidence A. We then wish to classify A into one of many equivalence classes representing the actual sequence of words spoken, W. To establish these equivalence classes we choose the sequence of words whose estimated posterior probability is maximal, given that the acoustic waveform A was observed:

$$W^{\star} = \arg\max_{W} \hat{P}(W|A)$$

where $W = w_1 ... w_n$ is a sentence consisting of a sequence of n words, and $\hat{P}(W|A)$ is our estimate, according to our models of the speech production process, of the probability that the words W were spoken given that we observed the acoustic evidence A.

By Baye's Law, we may rewrite $W^{\star}$:

$$W^{\star} = \arg\max_{W} \hat{P}(A|W)\frac{\hat{P}(W)}{\hat{P}(A)}$$

While computing this maximization, $\hat{P}(A)$ is constant, and can therefore be ignored.

Acoustic models are used to estimate $\hat{P}(A|W)$, and they must take into account the fundamental properties of speech production: what the basic phonetic units of speech sound like, the constraints on word pronunciations, contextual effects, etc. A language model is used to estimate $\hat{P}(W)$, and it must take into account the structure and constraints of the language to assign probabilities.

Once we have constructed models to estimate these two probabilities, a recognizer must consider all possible choices of W, in some efficient manner, checking $\hat{P}(W)$ to see if it is maximal. This process is referred to as the **search phase** of speech recognition, and it is typically compute intensive. The space which is being searched is the **sentence space**, which contains all possible sentences that the person may have uttered. The role of the language model is to, a priori, *constrain* this very large sentence search space to a more manageable one.

2.6 Language Modeling

2.6.1 Introduction

The role of a language model in a speech recognition system is to approximate the true underlying probability distribution of the target language. This approximation serves to reduce the number of possible sentences that needs to be searched in recognizing a given utterance. To formalize this role we abstract a language model as a device which associates with every sentence W a probability $\hat{P}(W)$, such that:

$$\sum_{W \in word^*} \hat{P}(W) = 1$$

The difference between this formal abstraction of a language model and the stochastic source model for the language is very small. For a language, the stochastic source generates sentences randomly according to the *hidden* underlying distribution. A language model performs the same function, just in the other direction — a sentence is provided to it, and it outputs the probability of that sentence. Both a language and a language model have a probability distribution over all sentences. Our goal in constructing language models is to have these two distributions be as close as possible.

Within this rather general abstraction of a language model there are many different approaches that have been taken to assign probabilities to sentences. The beauty of this abstraction is that all approaches to language modeling can be cast in this framework and therefore certain measures for evaluating language models, which depend only on the rules of this abstraction, can be used across all approaches.

Since it is clearly not practical for language models to actually store explicitly the probability of each possible sentence, especially when the language is infinite, all language models assume some sort of structure about the language they are trying to model, and then parameterize their probability estimates according to this structural assumption. The models will differ drastically on these structural assumptions. The role of these structural assumptions is to reduce the total number of parameters which must be stored, internally, and to enable the model to compute a probability for every possible input sentence.

Language models are usually trained on a large **training set** of example sentences from the language so that they can obtain robust estimates of their internal parameters. The required size of the training set is in proportion to the number of parameters within the model that need to be estimated. Thus, we will include in the abstraction of a language model the **model complexity**, or the number of internal parameters used by the model to compute its sentence distribution. The more complex a model, the more parameters it must estimate internally. The tradeoff between model complexity and performance must be carefully examined when choosing a particular language model.

2.6.2 Word by Word Causal Formulation

The model can internally compute the probability of a sentence in any number of ways. One popular formulation, for its ease of integration with recognition systems, uses the chain rule of probability to represent the probability of an individual word given **only** words preceding it. This form is a causal word by word framework:

$$\hat{P}(S) = \hat{P}(w_1, ..., w_n) = \prod_{i=1}^{n} \hat{P}(w_i | w_1, ..., w_{i-1})$$

where $S = w_1 w_2 ... w_n$ is a sentence.

The chain rule of probability is fully general, meaning *any* language model can *in theory* be reformed so that it is able to estimate the probability of each word, sequentially, given the previous words. But *in practice* many language models must do a lot of work to compute this reformulation.

2.7 What Language?

The notion of a **sub-language** of a natural language is important and extremely relevant to this thesis. A sub-language is a subset of the full natural language, and is a language in its own right, but constrained to a certain subject matter. For example, if we were to record an infinite number of sentences spoken by future travelers to their travel agents, those sentences would be a sub-language of the English language.

Until now the language we are working with has been deliberately left undefined, but at this point it is necessary to describe the language in a little more detail. The actual language being modeled is a function of what *task* is being handled by the speech recognition system. Typically speech recognition systems are constructed to handle very limited domains of queries (e.g., understanding air travel information, playing games of chess, simple geographical navigation in limited areas). Once the domain is known or defined, sample sentences are collected from likely users of the system (called experimental **subjects**). These sample sentences, which are divided into training and testing sets for evaluation of the language models and speech recognition systems, define a lexicon of words. Thus, the lexicon is tailored to the task at hand and will contain words that are appropriate to that task.

Typically the lexicon will be chosen so that those words that occurred more often than a certain threshold will be included. All other words (i.e., rare words) are automatically mapped to a special unknown word also included in the lexicon.

2.8 Evaluating Language Models

Language models can be evaluated in several different ways. Typically, for any of the following evaluations, the training and testing sets are fixed when comparing two different language models in order to avoid the extra variable of empirical noise. By far the most prevalent means is to compute an information theoretic quantity, called **perplexity**. Perplexity is a measure of the distance between a language model's distribution $\hat{P}(W)$ on all sentences and the true underlying distribution $P(W)$, in an information theoretic sense. It reflects how well the language model predicts a particular set of test sentences in the language. Perplexity is discussed in detail in the next section and in Appendix A.

Another means of evaluation is to compute how the language model impacts the word error rate of a particular speech recognition system. This evaluation is more relevant, but requires more work than perplexity, since two different models must be recoded so as to interface to the same recognition system. Furthermore, this evaluation will vary from one recognizer to another. There is a rough correlation between the reduction of perplexity and decrease in word error rate, but this connection is not formal.

A third method for evaluating language models is to turn the model around and use it to *generate* random sentences according to the model's probability distribution. This form of evaluation is really a subjective one — it is often very amusing and surprising to see what sentences the model feels are reasonable or likely. If the model is a good one, then the random sentences it generates will tend to be plausible sentences. If the model generates a large number of unreasonable sentences, this means the model has allocated some probability mass to unreasonable sentences, necessarily at the expense of reasonable sentences. If the model is perfect, i.e. $\hat{P}(W) = P(W)$ for all word sequences W, then the sentences produced by this random sampling will be reasonable sentences.

2.8.1 Perplexity

Perplexity [25, 37] is an information-theoretic measure for evaluating how well a language model predicts a particular test set. It is an excellent metric for comparing two language models because it is entirely independent of how each language model functions internally, and because it is mathematically very simple to compute. It is often referred to as the average "branching factor" of the language model with respect to a particular test set, meaning how many words on average can follow at any point in a test sentence. For a given vocabulary size, a language model with lower perplexity is modeling the language more accurately, which will generally correlate with lower word error rates during speech recognition.

Perplexity is derived from the average log probability that the language model assigns to each word in the test set:

$$\hat{H} = -\frac{1}{N} \times \sum_{i=1}^{N} log_2 \hat{P}(w_i|w_1, ..., w_{i-1})$$

where $w_1, ..., w_N$ are all words of the test set constructed by listing the sentences of the test set end to end, separated by a sentence boundary marker. The perplexity is then $2^{\hat{H}}$. $\hat{H}$ may be interpreted as the average number of bits of information needed to compress each word in the test set given that the language model is providing us with information.

While we assume a particular test set was produced by the same underlying source distribution of the training set, perplexity does empirically vary from one test set to another. For this reason two language models are typically compared based on the perplexity of the *same* test set.

The quantity $\hat{H}$ is actually just the empirical cross entropy between the distribution predicted by the language model and the distribution exhibited by the test set. Assuming the training and testing sets were drawn according to the same distribution, $\hat{H}$ will be very close to the cross entropy between the language model's distribution and the true underlying distribution of the language. Because cross entropy is lower bounded by the true entropy H of the language, perplexity is lower bounded by 2^H.

Thus, as we construct better and better language models, the models become closer and closer to the fundamental entropy of the language, beyond which we cannot they cannot be improved. Perplexity is an information-theoretic "distance metric" to measure how far a language model's prediction is from the true underlying distribution of the language.

Some language models employ complex means to compute the probability of a given sentence. It is often the case for such language models that the total probability mass which they allot is *less than* 1, and thus their distribution is not valid. Such models are said to have a **normalization** problem because their distribution could be made to sum to 1 with appropriate normalizing of all sentence probabilities. For these models determining the appropriate normalization is very difficult. However, as long as such models can guarantee that the sum over their distribution is *less than* 1 (rather than greater than), perplexity is still a meaningful quantity and is interpreted as an upper bound on the true perplexity if the language model were normalized correctly. Because the goal with language modeling is to *minimize* perplexity, an upper bound on the true perplexity is useful.

Appendix A derives some information theoretic bounds relating perplexity to the true entropy of the language.

2.9 n-gram Language Models

A very common approach to language modeling in present day speech recognition systems is the n-gram approach, so called because the parameters it stores internally are the frequencies of all unique sequences of n words (called n-grams) that occurred in the training set. This model directly predicts the causal word by word probabilities $\hat{P}(w_i|w_{i-1}, ..., w_1)$ for every possible context $w_{i-1}, ..., w_1$, which makes it amenable to integration with speech recognition systems. In addition, there are very efficient run-time tree structures for representing n-gram models. This approach to language modeling is very simplistic as it does not make any effort to model the structure of natural language, but it is the most competitive approach to date, in terms of perplexity and recognition word error rates.

The n-gram models take the simple approach of considering all possible word **contexts**, or words preceding the word to be predicted, that might ever occur. They collect counts of all such n-grams from the training data. Because training data are necessarily finite, it is not possible to compute probabilities for *every* possible context (of which there are infinitely many).

One very powerful way around this problem, which is the approach taken by n-gram models, is to collapse all possible word contexts into *chosen* equivalence classes, and then estimate one probability vector for each such equivalence class. The language model must therefore store a partitioning function, π, which maps any context into its equivalence classes, and must also store probability vectors for each of these equivalence classes:

$$\hat{P}(w_i|w_1 ..., w_{i-1}) \approx \hat{P}(w_i|\pi(w_1, ..., w_{i-1}))$$

Typically π is *chosen* at the outset by hand, and then the probability vectors are estimated for each equivalence class from the training set. By mapping many contexts into a single equivalence class, the model assumes that the true distributions of the words following these contexts are very similar. Simple n-gram models choose π to map any two contexts to the same equivalence class if they share the same last $n-1$ words:

$$\pi_1(w_1, ..., w_{i-1}) =< w_{i-n+1}, ..., w_{i-1} >$$
$$\hat{P}(w_i|w_1, ..., w_{i-1}) \approx \hat{P}(w_i|w_{i-n+1}, ..., w_{i-1})$$

Thus, the assumption with simple n-gram models is that the most substantial constraint offered by any given context comes from the nearest $n-1$ words. It is well known that natural languages have many constraints that are *not* so local, but empirically the local context does seem to provide substantial constraint.

The parameters for an n-gram language model are derived from the frequency counts of n-grams in the training set:

$$\dot{P}(w_i|w_{i-n+1}, ..., w_{i-1}) = \frac{C(w_{i-N+1}, ..., w_i)}{C(w_{i-N+1}, ..., w_{i-1})}$$

where $C(W)$ is the number of times the word sequence W was observed in the training set. The quantity $\dot{P}$ will be used to derive the model probabilities $\hat{P}$.

2.9.1 Sparse Data

One difficulty for all models, which is especially pertinent to n-gram models, is that of insufficient training data given the model complexity. For any language model, we must have enough training data to robustly estimate the parameters of the model. The n-gram models are particularly sensitive in this regard because they make no effort to reduce the number of parameters to estimate.

This problem is often referred to as the "sparse data" problem. With the n-gram model, the problem is manifested by the fact that many of the n-grams that occur in a new test set have never been seen in the training set, even for the bigram language model ($n = 2$). By the straight maximum likelihood estimates as shown above, these word sequences would be assigned zero probability. To deal with this difficulty various approaches, collectively referred to as **smoothing**, have been developed. The general approach is to *reallocate* some small amount of probability mass from n-grams that have been seen in the training data to those that have not been seen. The specific approaches differ only in where this probability mass is taken from and how much is taken.

One general approach to smoothing is referred to as **interpolation** [25], where the estimated probability vectors of n-gram model are smoothed with the probability vectors of the $(n-1)$-gram model:

$$\begin{aligned} \lambda &= \frac{C(w_1, ..., w_{n-1})}{C(w_1, ..., w_{n-1}) + K_s} \\ \hat{P}_n(w_i \mid w_{i-n+1}, ..., w_{i-1}) &= \lambda \times \dot{P}_n(w_i \mid w_{i-n+1}, ..., w_{i-1}) \\ &\quad + (1-\lambda) \times \hat{P}_{n-1}(w_i \mid w_{i-n+2}, ..., w_{i-1}) \end{aligned}$$

where K_s is a global smoothing parameter, and λ is a smoothing constant which varies from context to context but depends **only** on the count of that context. In general, the smoothing parameter λ can be chosen as above by optimizing K_s empirically, or through deleted interpolation techniques [25].

Typically the parameter K_s is optimized so as to minimize perplexity on an independent test set. This smoothing process is recursive so that each level of the n-gram model is smoothing with the vectors from the $(n-1)$-gram model.

At the unigram level, we smooth the distribution by "pretending" we saw all words at least a minimum count floor (C_{min}) number of times. This guarantees that every possible word sequence is assigned some non-zero probability. We state without proof that this model is a valid one – i.e., all probability mass sums to precisely 1.

The interpolation approach is very simple in that it uniformly transfers probability mass from all n-grams. Other approaches try to steal more probability mass from infrequent n-grams, for example back-off smoothing [27], which only removes probability mass from those n-grams that occurred fewer than a certain cutoff number of times. There are various other approaches, such as nonlinear discounting [33], and Good/Turing [21, 13].

When one uses an n-gram language model to generate random sentences, as shown in Tables 2.1, 2.2 and 2.3, very few of the resulting sentences could be considered valid. This indicates that n-gram models leave substantial room for improvement. The sentences for the trigram model are curious because locally the words are somewhat reasonable, but in the long term the sentence completely loses focus.

2.9.2 Word Class n-gram Models

A slight modification of the n-gram model is that of the word class n-gram model [11]. This model first assigns words to word classes, $M(w)$, and then maps contexts to equivalence classes based on these word classes:

$$\pi_2(w_1, ..., w_{i-1}) = < M(w_{i-n+1}), ..., M(w_{i-1}) >$$

Word class n-gram language models can be competitive because they can have far fewer parameters, thus making better use of the training data. There are fewer contexts to store, since many word n-gram contexts will map to a single class n-gram context.

2.10 Decision Tree Models

Another approach to language modeling is the decision tree language model [5, 10]. These models function like the n-gram model in that they represent a function to classify any possible context into a finite number of equivalence classes. But they differ on how the function is chosen. In particular, a binary decision tree is created from the training set. This decision tree asks questions at each node of a particular context, thus mapping every context which comes in at the root to a particular leaf of the tree. The leaves represent the equivalence classes.

The interesting property of decision trees is that they can have access to a number of questions and choose those questions which empirically function the best. A context can thus be classified based on very long distance information as well as local information.

The performance of these models is very close to that of the n-gram models, but they take considerable computation to build.

```
midnight lockheed sure boston
i+ve
atlanta downtown to live
the
fly
american early philadelphia u about philadelphia from atlanta thirty right
four t_w_a to i american
sort flight available land provides you six of midnight which
of aircraft fare east
the the stopover city the coach the one_way frequent in traveling flights
to show please later coast eight u_a those does guess specific august the
reserve on are to used that+s_fine these
does me fly noon like me mid unrestricted continue i provides ways
stop looks boston the three sorry cheapest how fare the
dallas_fort_worth
lands what
least tuesday denver how return the ticket does march
to
two on flight really boston me like
and like six would these fifty
```

Table 2.1: Random Unigram Sentences

```
show me the flight earliest flight from denver
how many flights that flight leaves around is the eastern denver
i want a first_class
show me a reservation the last flight from baltimore for the first
i would like to fly from dallas
i get from pittsburgh
which just small
in denver on october
i would like to san_francisco
is flight flying
what flights from boston to san_francisco
how long can you book a hundred dollars
i would like to denver to boston and boston
make ground transportation is the cheapest
are the next week on a_a eleven ten
first_class
how many airlines from boston on may thirtieth
what is the city of three p_m
how many from atlanta leaving san_francisco
what about twelve and baltimore
```

Table 2.2: Random Bigram Sentences

```
what type of aircraft
what is the fare on flight two seventy two
show me the flights i+ve boston to san_francisco on monday
what is the cheapest one_way
okay on flight number seven thirty six
what airline leaves earliest
which airlines from philadelphia to dallas
i+d like to leave at nine eight
what airline
how much does it cost
how many stops does delta flight five eleven o+clock p_m that go from
what a_m
one_way
is eastern from denver before noon
earliest flight from dallas
i need to philadelphia
describe to baltimore on wednesday from boston
i+d like to depart before five o+clock p_m
end scenario c_m_u thirty four eighty seven please
which flight do these flights leave after four p_m and lunch and <unk>
```

Table 2.3: Random Trigram Sentences

2.11 Structured Language Models

The other major class of approaches to language modeling is those that assign structure to the sequence of words, and utilize that structure to compute probabilities. These approaches typically require an expert to initially encode the structure, typically in the form of a grammar. In general, they have the advantage over n-gram approaches because there are *far* fewer parameters to estimate for these models, which means far less training data is required to robustly estimate their parameters.

However, structured models require some amount of computation to determine the structure of a particular test sentence, which is costly in comparison to the efficient n-gram mechanism. And, this computation could fail to find the structure of the sentence when the test sentence is agrammatical. With such sentences structured models have a very difficult time assigning probability. This problem is referred to as the robustness problem.

When discussing perplexity results for such structural language models, typically the test set *coverage* is first quoted, which is what percentage of the test set parses according to the grammar, and then the perplexity of the language model is quoted for that particular subset of the test set. This makes objective comparisons of such models difficult since the perplexity varies with the particular subset of the test set that parses.

Some extensions have been made to these systems to try to deal in some manner with the robustness problem. For example, both the TINA and PLR models, described below, accommodate such exceptions. But, the emphasis of these *robust parsing* techniques is to extract the correct meaning of these sentences in rather than to provide a reasonable estimate of the probability of the sentence.

2.11.1 SCFG

One structural approach is the stochastic context free grammar (SCFG) [26], which mirrors the context-freeness of the grammar by assigning independent probabilities to each rule. These probabilities are estimated by examining the parse trees of all of the training sentences.

A sentence is then assigned a probability by multiplying the probabilities of the rules used in parsing the sentence. If the sentence is ambiguous (has two or more parses) the probability of the sentence is then the sum of the probabilities of each of the parses. This process generates a valid probability distribution, since the sum of the probabilities of all sentences which parse will be 1.0, and all sentences which do not parse have zero probability.

SCFG's have been well studied by the theoretical community, and well-known algorithms have been developed to manipulate SCFG's, and to compute useful quantities for them [32, 26]. A SCFG may be converted into the corresponding n-gram model [43]. The inside/outside algorithm is used to optimize the probabilities such that the total probability of the training set is optimized [2, 29]. Given a sentence prefix, it is possible to predict the probabilities of all possible next words which may follow according to the SCFG.

2.11.2 TINA

TINA [38] is a structured probabilistic language model developed explicitly for both understanding and ease of integration with a speech recognition system. From a hand-written input CFG, TINA constructs a probabilistic recursive transition network which records probabilities on the arcs of the network. The probability of a sentence, if it parses, is the product of all probabilities along the arcs. Thus, the probabilities are conditional bigram probabilities, at all levels of the recursive network.

For each unique non-terminal NT in the input grammar, TINA constructs a small bigram network that joins all rules that have the particular left-hand-side non-terminal. This process is actually a form of generalization as it allows more sentences to be accepted than were accepted by the original grammar. Probabilities are no longer explicitly rule-dependent as in the SCFG formalism, but instead are dependent in a bigram-like manner on the unit immediately preceding a given unit.

TINA parses a test sentence using a best first strategy whereby the partial parse which looks most promising is chosen for extension. This has the effect of creating an extremely efficient parser, when the model is trained well, and was motivated by the notion that humans do not have nearly as much difficulty with ambiguous parses as computers do. TINA also has a trace mechanism and a feature-unification procedure, which complicates the probabilistic model and makes it more difficult to estimate the perplexity.

TINA, like SCFG, only assigns probability to those sentences which it accept. It therefore has the same difficulties with robustness as other structural approaches. Recent work has been done to address this problem, mainly by resorting to standard bigram models for words that fall outside of parsable sequences [39].

2.11.3 PLR

PLR [19] is another approach to which generates a probabilistic language model from a CFG as input. It is a probabilistic extension to the efficient LR parsing algorithm [1], and is designed to achieve full coverage of a test set by implementing error recovery schemes when a sentence fails to parse.

The LR model takes a CFG as input and uses this CFG to construct a *deterministic* parser. This parser looks at each word in the input sentence in sequence, and decides to either *shift* the word onto the top of the stack, or *reduce* the top of the stack to a particular non-terminal. The PLR model extends this by predicting word probabilities as a function of which units are on the top of the stack. At every iteration, the stack represents the partially reduced portion of the test sentence processed so far.

One limitation of this approach is that it requires that the input CFG fall within a special class of CFG's known as the *LR* grammars. If the grammar is in this class, it is possible to construct a deterministic LR parser. An advantage of this model is that it is already formulated as a word by word predictive model, thus making integration to speech recognition systems easier. It is also extremely efficient as, at any given point, the LR machine knows exactly which action, shift or reduce, applies.

Chapter 3

Grammar Inference

3.1 Overview

The process of automatically acquiring a language model is divided into two steps. First, we try to learn a CFG which describes the training set and also generalizes beyond it to the target language. Second, we use the learned CFG to construct a probabilistic language model. The first step, which is described in this chapter, is implemented using a form of grammar inference. It is the only step which does any "learning". As output it produces a stream of grammars, each grammar more general than those listed before it. The second step takes each learned grammar and trains a probabilistic model using the training set. We can measure the overall performance of these two steps by evaluating the final acquired language model according to the known objective measures for language models.

In this chapter we describe an algorithm to implement the first of the above steps. This algorithm is a form of grammar inference [16], which is an iterative process that tries to learn a grammar to describe an example set of training sentences. From the finite training set the inference process produces a stream of *learned* grammars G_0, G_1, G_2, ..., where each grammar is successively more general than the ones before it. Typically the initial grammars are quite specific in predicting only the training set, but through generalization the grammars eventually become too general. One extreme grammar is the final grammar, whose language will be something close to all possible sentences. Somewhere in between these two extremes is the "best" grammar.

The overall inference process is actually quite simple, but the details are somewhat complex. The process begins with a very explicit grammar which precisely encodes the training set. It then iteratively simplifies, or transforms, that grammar. Each transformation generalizes the language accepted by the grammar such that each language is a superset of the language accepted by the previous grammar. An example of this generalization could be as follows. Throughout the training set it may be apparent that the words "boston" and "denver" occur interchangeably, according to some objective measure. Furthermore, there is a sentence "show me the cheapest flights leaving boston before six p m" in the training set. The inference system could *generalize* from this to also predict the sentence "show me the cheapest flights leaving *denver* before six p m". This notion is at the very heart of the inference process.

Figure 3-1 shows a block diagram describing the general inference process. The **merging component** is the process that generalizes each grammar by one iteration. It selects units to merge, which may be either a word, a non-terminal, or a phrase from the **phrase set**. The phrase set stores the top S phrases which can be considered for merging at each iteration, where S is a parameter of the inference process.

3.2 Formal Machine Learning

The machine learning community has explored the theoretical limitations and properties of algorithms which can learn a language from example sentences of the language, either positive or negative or both. Typically some sort of formalism for learning is chosen, whereby a learner is presented with examples one at a time and must then formulate a hypothesis over time that may be used to classify future examples. The learner constructs the hypothesis that it believes is the correct one from the examples, and the performance of the learner is measured objectively based on how well it classifies future examples.

One such formalism of learning, proposed by Solomonof [42], is a strict sense of learning called **learning in the limit**. This formalism requires that the learner

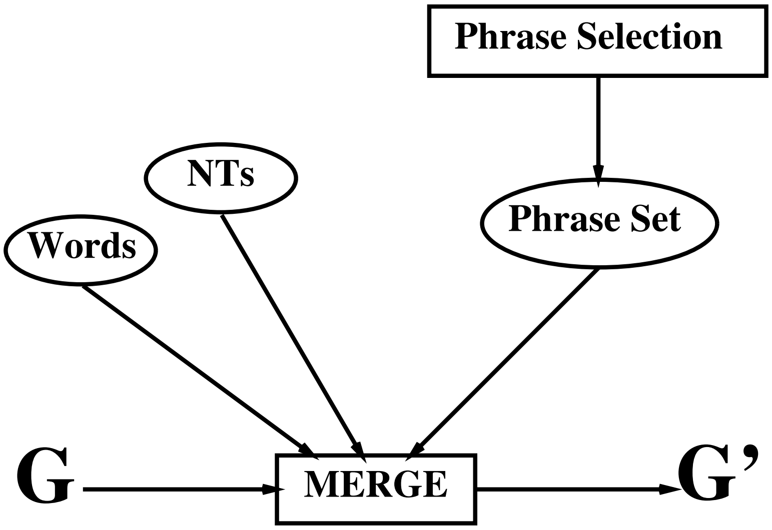

Figure 3-1: Schematic diagram of the inference process
This shows the function of each component. The merging unit actually transforms the grammar by selecting two similar units to merge. These units can either be word or non-terminals or phrases from the phrase set.

eventually learn the concept or language *precisely*. Thus, the learner is presented with many examples, positive or negative, and in any possible order. It is then said to be able to learn a concept class if for any concept in that class, and after some bounded number of examples, it will correctly classify all future examples. Many efforts have been made within this formalism to show that various concept classes are or are not learnable. Gold [20] has shown that not even the regular languages, let alone the context free languages, can be learned within this framework.

The learning formalism established by Solomonof is very strict in that it demands precise learning of the concept. A more relaxed and practical formalism was proposed recently by Valiant [44], called PAC-learning, which stands for "probably approximately correct learning". Instead of requiring the learner to classify all future examples, it simply imposes a bound on the probability of misclassification. Thus, the learner is allowed to make mistakes, as long as it does not make too many. Many efforts have been undertaken to show that certain classes of concepts are or are not "PAC-learnable" or efficiently (polynomial time and polynomial in the requirement of the number of examples that must be seen) PAC-learnable.

Each of these learning models also encompasses what sort of evidence is presented to the learner, along with what sorts of concept classes are being worked with. Typically the evidence (examples) can be only positive, or both positive and negative. Learning from only positive data is difficult because the process never knows if the concept it has acquired tends to over-generalize.

The results from this community are of interest to the linguistic community because it is widely held that children are exposed to very little negative evidence as they learn natural languages [31, 34]. This constrains the linguistic theories of human language as they must be consistent with this evidence.

For our purposes, we measure learning with far less stringent requirements. We measure the quality of the resulting language models and speech recognition systems which use these language models. The results of the machine learning community are interesting only in the abstract. From our point of view, if an algorithm is able to produce practical results through some learning framework, that algorithm is useful.

3.3 Grammar Structure

We can look at a CFG as serving two different functions. First, it performs **classing**, which is specifying which grammar **units** (words and non-terminals and phrases) of the grammar fall into the same class. The grammar accomplishes this by creating rules which allow a given non-terminal to derive each of the units grouped into the same class. For example, a grammar will group similar words into the same class – days of the week, city names, month names. It can also do so with phrases. For example, it could group the phrases "on tuesday evening" and "before two o'clock" together. Therefore, any non-terminal in the grammar can be viewed as denoting a class of similar words and phrases according to what the grammar allows that non-terminal to derive.

The second role that a grammar serves is to assign structure to the language it derives by specifying which sequences of words and non-terminals should be bracketed as phrases. For example, the grammar will specify that "to CITY", "from CITY", and "the least expensive flight" are phrases in the language. This function of the grammar is referred to as **structure building**.

The two functions – classing and structure building – are very much intertwined with one another. In building structure, the grammar specifies phrases containing non-terminals which denote classes. In grouping different units into classes the grammar groups the structured units as well as single words and non-terminals into classes.

Given this interpretation of a CFG, a natural decomposition of the grammar inference algorithm is into two different components: one that searches for appropriate classes, and one that searches for structures to build. Somehow these two separate functionalities of the inference process must then be made to function gracefully together.

3.4 Occam's Razor

The word inference means to predict a general rule given specific example applications of the rule. When applied to context-free grammars, grammar inference is the process of predicting a general CFG to describe a set of example sentences. The process generalizes from a training set, called the *positive sample*, to produce grammars which accept not only the training set but sentences beyond the training set. Inference systems can also be provided a **negative sample**, which contains examples of elements which are *not* example applications of the general rule. We do not have access to such data in our grammar inference, as such a notion does not really make sense – how would we find prototypical negative example sentences? Thus the inference process described here works only from positive data.

The inference process is guided by finding a relatively simple grammar to describe the training set. This principle is commonly referred to as **Occam's Razor**, or "the simplest explanation is best", and embodies the rather strong philosophical statement that the world we live in tends to favor simpler things. In this spirit, inference begins with a *relatively* complex description of the training set and then proceeds to iteratively generalize and simplify this description. In this context, the word *complex* means a large model, in terms of its physical size (the total size of all rules in the grammar, for example).

3.5 Inference Definitions

We assume that there is an underlying language, L_t, which is the actual target language that we are trying to learn. Furthermore, we assume that all sentences in the independent training and testing sets are in L_t and that these sets were drawn according to the probability distribution associated with the language L_t.

The grammar inference system consists of the pair $\langle G_0, T \rangle$, where G_0 is the starting grammar derived directly from the training set, and T is a function which takes a grammar G as input and outputs a set of grammar transformations which could apply to G. G_0 is a large grammar whose corresponding language is *exactly* the training set. Any one of the transformations $t_i \in T(G)$ for G, can be chosen to transform G into G': $G' = t_i(G)$. These transformations represent the means of generalization that are available to the inference system at each step, and typically simplify the grammar by reducing its complexity or size. They have the property that they *only* transform the grammar such that $L(G) \subseteq L(G')$):

$$\forall t_i \in T(G) : L(t_i(G)) \supseteq L(G)$$

The initial grammar G_0, and the transformation function $T(G)$ entirely define the inference system. The set of all reachable grammars, by some sequence of transformations starting from G_0, defines the **grammar space** $\hat{G}$ of the inference process. Correspondingly, the **language space** $\hat{L}$ is the set of all possible languages this system can acquire. If the target language $L_t \in \hat{L}$ then the inference process may, with an appropriate search, find the correct language using a series of transformations starting with the initial grammar. If $L_t \notin \hat{L}$ then the inference algorithm can at best approximate the target language. The mapping from $\hat{G}$ to $\hat{L}$ is a many to one mapping since there are many grammars for each language. The language space contains only languages which accept all sentences in the training set.

Because the grammar space can be exponentially large, the means employed to search the space for a good grammar is a greedy search. At each iteration, the single transformation which appears to be the *best*, according to a defined metric, is chosen and applied to the current grammar. This process is done iteratively until either no more transformations can apply to the grammar, or a stopping criterion has been reached. The resulting output of this process is a sequence of grammars G_0, G_1, ..., each slightly more general than the previous one. By varying the criterion used to choose the single best transformation from each set $T(G_i)$, and by varying what transformations are included in this set, we can produce different streams of grammars as output.

Figure 3-2 shows a graphical depiction of the relationship between the various sets of sentences involved in the inference process. These bubbles are drawn in sentence space; thus each bubble represents a particular subset of all sentences, or a formal language. We have finite training and testing sets, which are assumed to be drawn independently according to the underlying fixed probability distribution on sentences. These two sets will overlap with some non-zero probability. There is as well the actual target language which we are trying to learn during the inference process. The target language includes every sentence in the training and testing sets, but certainly contains sentences which are not in either set. It is the goal of the inference process to generalize from the training set to the target language.

Finally, there is the language which we have learned after some number of iterations of the inference process. As explained earlier, the learned language will *always* be a superset of the training set. The language learned by inference process can suffer from both under-generation and over-generation, as shown in Figure 3-2.

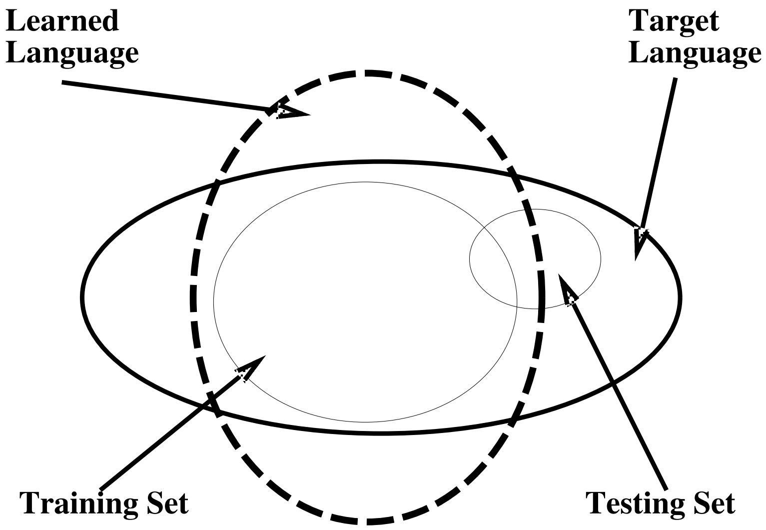

Figure 3-2: Inference language relationships
This figure shows the relationship between training and testing sets and the target and learned languages during the inference process.

3.6 Initial Grammar G_0

For the unital grammar G_0, we chose a grammar which contains exactly one rule for each sentence in the training set, of the form $S \Rightarrow s_i$ where s_i is the ith sentence in the training set. This grammar is extremely large since it does not make any attempt to share common structures between the sentences of the language. The initial language, L_0, is therefore exactly the training set.

This group of rules, collectively referred to as the **sentence rules**, are present in all subsequent grammars derived by the inference process. However, each rule within the sentence rules can be altered with each iteration of the inference process. In addition, new rules will be created and added to the grammar. These rules are referred to as the **acquired rules**.

One nice property of this starting point is that we could easily give the system a head start by providing it with an initial hand written grammar, thus saving some computation time and space.

3.7 Merging

The set of grammar transformations $T(G)$ for a given grammar G contains all transformations which perform a **merge** of two units in the current grammar G. A merge takes two **units** u_i and u_j which are currently present in the grammar and modifies the grammar such that these two units are interchangeable in every context both in the new grammar *and* in the language accepted by the new grammar. A unit can be either a single word (e.g., "denver"), or a single non-terminal in G representing the class created by a previous merge (e.g., "NT_5"), or a sequence of words and non-terminals (e.g., "please show NT_4"). For example, the phrase "washington d c" and the word "boston" may be selected as similar units to merge.

The grammar transformation is mirrored in the language accepted by the grammar. As a result of merging "washington d c" and "boston", any sentence whose partially reduced form is of the form "α washington d c β", where α and β can be zero or more words or non-terminals, allows the corresponding sentence "α boston β" to also be included in the language. Symmetrically, substituting "washington d c" for "boston" allows new sentences to be added to the language as well.

Formally, a merge must make a number of changes to the grammar G. A fresh non-terminal, for example NT_0, is chosen. The grammar is most un-creative in its selection of fresh non-terminal names — it simply chooses NT_i for $i = 0, 1, 2$, etc. Two new rules are added to G:

$$NT_0 \Rightarrow \text{washington d c}$$
$$NT_0 \Rightarrow \text{boston}$$

Next, wherever "washington d c" or "boston" occur in the present grammar, that occurrence is replaced with NT_0. This replacement is done for *both* sentence and acquired rules. It has the effect of allowing "washington d c" to appear wherever "boston" previously appeared, and vice-versa. There are some further transparent modifications that are done to the grammar to remove unnecessary non-terminals and to otherwise "clean" the grammar to make it easier to read. The notation used throughout the rest of this thesis to denote the merging of two units u_i and u_j to create NT_k is $(u_i, u_j) \rightarrow NT_k$.

As a result of this replacement everywhere in the present grammar, there may be several rules that have been reduced to exactly the same form on their right hand sides. In particular, any two rules which previously differed *only* by "washington d c" and "boston" will now be equal. For example, if these rules:

$NT_4 \Rightarrow$ from washington d c

$NT_4 \Rightarrow$ from boston

were already in the grammar, they will be duplicate rules as a result of this merge. Since it is unnecessary to have duplicate rules in a CFG, such rules are discarded (leaving one copy). This elimination of duplicate rules actually reduces the size of the grammar substantially, even though the grammar size actually increases slightly at first by adding the two new rules.

Appendix B describes a detailed example of a sequence of merges.

3.7.1 Infinite Merges

The exception to the merging process as described above are situations when the merge would introduce recursion into the grammar, thus generating an infinite language. One example of this is trying to merge two units where one unit is contained in the other: "please show me" and "show me". In the naive merging algorithm described above, this would create the rules:

$NT_0 \Rightarrow$ please NT_0

$NT_0 \Rightarrow$ show me

because the unit "show me" appeared in the rule just added, and we want to make both units interchangeable *in all contexts*. This would have the effect of allowing any number of "please's" to precede the phrase "show me". Clearly exactly zero or one "please's" ought to be allowed, so we chose to disallow these **infinite** merges. Whenever such a merge is proposed, we still allow the merge but we carefully update the grammar so as to avoid introducing recursion. Thus, the merge would add instead:

$NT_0 \Rightarrow$ please show me

$$NT_0 \Rightarrow \text{show me}$$

Another way to describe such situations is that there is an optional word which may be inserted or not. In the example above, that word is "please". Natural language tends to use many such optional words, so properly handling such cases is important.

3.8 Phrases

Some means must be provided to allow the inference process to acquire structure, or **phrases**. A phrase is a unit consisting of two or more terminals or non-terminals in sequence (e.g., "show me").

Ideally, every possible phrase, perhaps limited to only those word sequences that actually occur frequently enough in the training set, should be considered. But this is computationally unrealistic because the cost of introducing phrases grows with the square of the number of phrases. There are too many possible phrases to consider every one of them exhaustively.

Some criterion is needed to select a subset of all possible phrases which look "likely". To achieve this, we define a simple objective metric, shown in Equation 3.1, to measure the quality of the the phrase "u_1 u_2". The units u_1 and u_2 could be a word, a non-terminal, or *even* another phrase. In this manner longer phrases than length two can be considered even though the concatenation process is binary. This metric is based on mutual information, which has been used it other grammar inference approaches as a means to acquire structure [8].

$$Pr(u_1, u_2) \times log(\frac{Pr(u_2|u_1)}{Pr(u_2)}) \tag{3.1}$$

This score is high when two units are "sticky", meaning the probability that u_2 follows u_1 is much higher than the simple frequency of u_2. The additional weight of the frequency of the bigram is added to favor phrases which occur frequently.

Using this metric, we select the top S best phrases whose frequency count satisfies the count threshold ($\geq M_c$), and store these phrases in a **phrase set**. Table 3.1 shows

Phrase	Score $\times 10^{-2}$
show me	4.06
from boston	3.28
from boston to	3.27
boston to	2.79
flights from	2.71
i+d like	2.51
is the	2.51
what is the	2.82
what is	2.51
to san-francisco	2.45
would like	2.13
i would like	2.07
i would	2.04
flight from	1.64
like to	1.63
i'd like to	1.65
the cheapest	1.55
me the	1.52
show me the	1.99
ground transportation	1.41

Table 3.1: Example phrases and weighted mutual information scores.

an example phrase set of size 20. Once a phrase is selected, its distributional counts are collected by looping through all of the training sentences searching for occurrences of the phrase. The merging process is then free to choose any of these phrases, or a single word or non-terminal, as it searches for the most similar pair of active units.

The merging process is entirely independent from the phrase set. Only the merging process actually changes the grammar. The phrases which are added to and removed from the phrase set have no direct effect on the grammar. Only when the merging process chooses a phrase to merge is the grammar altered to include a phrase. The phrase set must be updated after every merge because certain phrases may now be obsolete (if they contained one of the merged units), while other newly competitive phrases may need to be added.

3.9 Choosing Merges

At each iteration the system must choose one merge from among many candidates as the actual merge to apply. Because a merge takes two previously distinct units and merges them together such that they are interchangeable everywhere, it seems reasonable to choose as the criterion for merging a metric which measures how similar or interchangeable the units really seem to be in the training set.

This similarity metric will be based on **distributional information** – i.e., the contexts in which a given unit occurs and the frequency of occurrence in each context. The context that is considered could be very general, but in the interest of computational resources, we limit it to simple left and right bigram units. Every possible unit u_i has two context vectors associated with it – one listing the counts of units k which ever occurred to the left of this unit ($L_i[k]$) and one listing counts of units which occurred to the right of it ($R_i[k]$), in the training data. From these count vectors, smoothed probability vectors are estimated representing the probabilities $\hat{P}(\text{left context } k|u_i)$ and $\hat{P}(\text{right context } k|u_i)$. This smoothed distribution is computed in exactly the same way as the interpolated n-gram distribution as described in Section 2.9.

There are various possible choices for "distance metrics" once we have these context probability vectors for all units. We chose as our distance metric divergence [28], which is a natural information theoretic distance between probability vectors. Divergence is based on the relative entropy $D(p||q)$ between two probability vectors p and q, each of length N:

$$D(p||q) = \sum_{i=1}^{N} p(i) \times log \frac{p(i)}{q(i)} \tag{3.2}$$

Divergence is then the symmetric sum $D(p||q) + D(q||p)$.

In order to be able to use a distributional distance metric such as divergence we must maintain the distributional counts of all candidate units for merging. At the start of the inference process, these count vectors need to be set properly by looking through all of the training sentences. After each merge is completed, the distributional count vectors for *all* units must be updated accordingly. For the newly created merged class, the new count vectors are simply the *sum* of the individual count vectors of the two units before they were merged. For all other units which occurred in a context involving one of the merged units (i.e., preceded by or followed by one of the merged units), their distributional counts are altered as well.

3.10 Minimum Count Threshold

The time and space complexity of this inference process is sensitive to the number of candidate units that must be considered for merging. To limit this number, and thus allow the process to run more expeditiously, we impose a minimum count limit M_c on any unit which may be considered for merging. If a unit has occurred fewer than M_c times in the training set, it cannot be considered as a candidate for merging. Another motivation for this is to ensure that the contextual probability estimates are *robust* so that a comparison of two units based on the divergence metric is reliable. The minimum count threshold ensures that we are merging units whose distributional information is *trustworthy*.

For all inference experiments completed in the rest of this thesis, M_c will be 5 unless otherwise stated. The inference run with either $M_c = 5$ or $S = 100$ or both is referred to as the **standard inference run**, and will be the default inference run when these parameters are not explicitly specified. These values were chosen because they represent near optimal choices as determined in the next chapter.

3.11 Active Grammar Units

The **grammar units** are defined as the *single* words and non-terminals in the grammar which have not yet been merged. This includes any word which has not yet been merged, and any non-terminal which was created by a previous merge and *not* merged since. The parameter N will be used to denote the number of these units.

A subset of these grammar units, called the **active grammar units**, or active units, are those units whose count satisfies the minimum count threshold M_c and are thus candidates for merging. The number of these units is defined as N_a, where $N_a \leq N$. During the merging process, $N = N_a + K$ for some constant K, because the non-active grammar units, of which there are K, will remain unmerged during the entire inference process.

The two parameters N_a and N are parameters of interest for several reasons. The computational complexity of the inference process depends strongly on both of them. At the start of the inference process, N is precisely the number of *words* in the lexicon, and N_a is the number of words whose count is $\geq M_c$. N_a is very sensitive to the value of M_c. Figure 3-3 shows how many words occur $\geq M_c$ times in the training set, as a function of M_c. Thus, the choice of M_c has tremendous impact on the computational complexity of the inference process.

The evolution of N as the inference process iterates is a means of comparing how much classing versus structure building is being accomplished by the acquired grammar. For a given merge, N can change by 1, 0, or -1 depending on the units. If both merged units are a single word or single non-terminal, the merge collapses two previously distinct active units into one active unit, thus reducing N (and N_a) by 1. If one merged unit is a single word or non-terminal, but the other is a phrase from the phrase set, the number of units in the grammar remains unchanged, because the merge process has simply grouped the phrase with the class of a currently active unit. Finally, if both units are phrases, then the merge process has spontaneously created a new active unit represented by NT_k, containing only those two phrases, which increases N by 1. This evolution of N is dependent on various parameters of the inference process, but is also very dependent on the underlying nature of the language: does the language structure favor more classes or more phrases ?

Figure 3-4 shows the evolution of N for an inference run when $M_c = 5$ and $S = 100$. The dotted line in this figure shows the steepest possible curve for N_a for reference, which is the case when the grammar is learning *only* word classes. Thus, the deviations from that reference curve and the curve shown occur because the inference process is choosing to merge some phrases. Before any merges, $N_a = 512$, i.e., there are 512 words in the lexicon whose counts satisfy the minimum count threshold. At

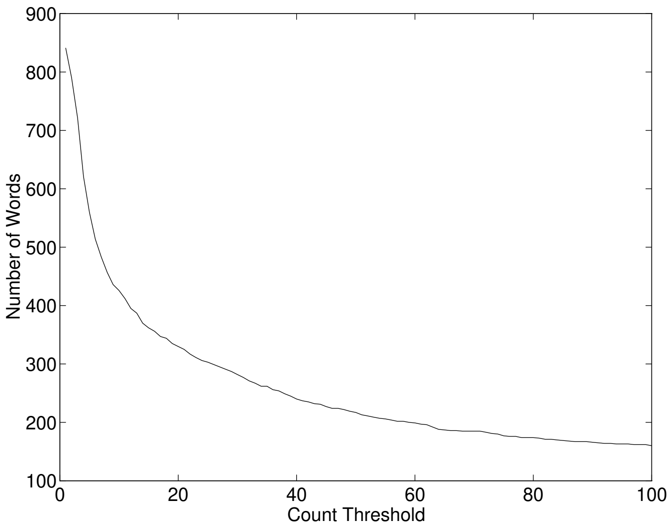

Figure 3-3: Word count distribution
Number of words which occur more than a certain threshold number of times in the training set. This number drops off quickly as the cutoff increases.

first, the inference algorithm very much favors grouping words into classes. But then, as it progresses, it builds more structure by merging phrases together.

3.12 Computational Complexity

The actual running time of the inference process is quite sensitive to the parameter M_c. The current implementation of the system, which is written in C, takes on the order of eight hours on a single Sun SparcStation 10 to compute a sufficient number of merges. In this section we compute the running time and space complexity of the various components of the system.

3.12.1 Space

For every active unit plus every phrase in the phrase set, we must maintain vectors of the distributional counts of each unit. These vectors are represented sparsely and thus take space proportional to how many units actually occur as the context for a

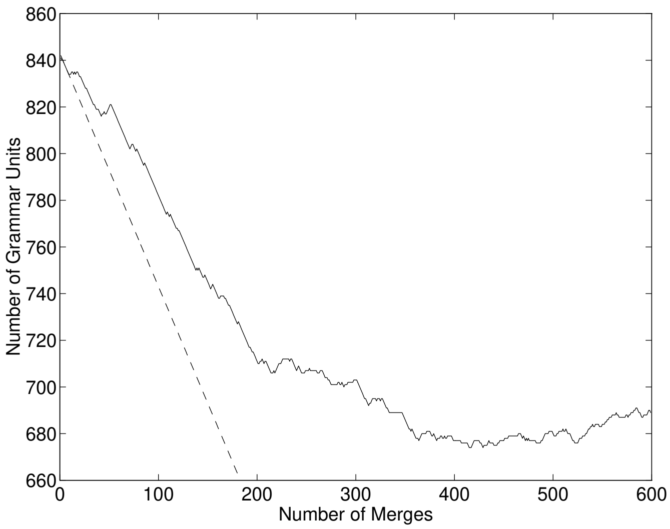

Figure 3-4: Active grammar units
Evolution of the number of active grammar units N_a for the standard inference run. The dotted line shows the steepest possible curve as reference, which is the word-class-only case.

given unit, in the

worst case $O(N)$. There are $N_a + S$ units whose counts must be stored, so the total space is at worst $O(N(N_a + S))$.

In addition to this, the training sentences have to be maintained in their partially reduced state to be able to collect context counts for the phrase set construction, taking $O(T)$ space. Finally, the grammar needs to also be maintained. The bulk of the space in the grammar is consumed by the sentence rules. We can actually save this space by using the partially reduced training sentences to serve this purpose as well. Additional space is used by the acquired rules in the grammar, but that is very small, on the order of the number of merge iterations.

Thus, the total space complexity of this inference algorithm is $O(N(N_a+S)+T)$. If the training set is very large, this complexity will be dominated by T, otherwise it will be dominated by the number of active units in the grammar or the number of words in the lexicon. Again, this in turn is very sensitive to the minimum count threshold M_c.

3.12.2 Merging: Time

A merge involves three computational steps. First, the pair of most similar active units and phrases must be identified. Second, the grammar must be updated to reflect the merge. And third, the distributional counts must be recomputed. Searching for the two most similar units to merge requires $O((N_a+S)^2)$ unit pairs to be considered. Computing divergence involves looping through the distributional vectors for each unit, which can involve N units. The total time to find the best pair of units is thus $O(N(N_a+S)^2)$, which is quite sensitive to N_a and N.

Fortunately a simple improvement can be implemented in selecting merges by maintaining a queue of the top few hundred best merges at each iteration. At the beginning of the inference process, we first perform the *full* computation to fill the queue initially, taking time $O(N(N_a+S)^2)$. Then, when a merge is performed, the divergence is only recomputed between pairs of active units whose distributional counts were *altered* as a result of this merge. While in the worst case there can still be $O((N_a+S)^2)$ unit pairs to compare, in practice it is quite a bit less and this **merge queue** saves substantial time.

Updating the grammar requires searching for any instance of u_i or u_j in the grammar and replacing it with NT_k. This can be done in time proportional to the total number of words T in the training set, which is $O(T)$.

Updating the distributional counts is a matter of first adding the count vectors of the two merged units, which takes $O(N)$ time. Then we must look through the count vectors of all other active units adding together the context counts of the two units wherever they occur in another unit's context. The count vectors are stored using a sparse representation, thus to look up each unit takes $O(logN)$ time using a binary search. Since there are N_a units, the total time is $O(N+N_a logN)$.

Adding all of this up, a single iteration of merging takes time:

$$O(N(N_a+S)^2+N+N_a logN+T)$$

$$= O(N(N_a + S)^2 + T)$$

3.12.3 Phrases: Time

Filling the phrase queue requires selecting S phrases. Each selection requires considering all pairs of active units to concatenate, of which there are $O((N_a + S)^2)$. The weighted mutual information calculation for each pair takes $O(1)$ time. Once a phrase is selected the distributional counts for that phrase must be collected. This requires looping through the entire training set of sentences searching for all occurrences of the phrase. This takes $O(T)$ time, where T is the total number of words in the training set. Adding a phrase to the phrase set therefore takes $O(T + (N_a + S)^2)$ time.

At the start of inference, the phrase set is filled by finding the best S phrases, taking total time $O(S(T + (N_a + S)^2))$. Then, after each merge we must update the phrase set. In the worst case this could involve replacing all S phrases, which would result in a worst case running time of $O(S(T + (N_a + S)^2))$ per iteration, but typically the actual number of phrases replaced is quite a bit less.

3.13 Discussion

In this section we describe some interesting issues and properties of the inference process.

3.13.1 Phrase Ordering

The fact that each merge is *also* applied to previously acquired rules deserves special attention. Although the inference process may have merged a large phrase with some other unit, that phrase may be further decomposed when sub-phrases within it are merged. For example, the phrase “show me all flights” can be merged with “show me the flights”. A later step could merge “show me” with “please list”, thus replacing the occurrence of “show me” in the two original rules. This notion is interesting because it allows the grammar to acquire the eventual phrase structure in a somewhat arbitrary order. At one extreme, it could at first learn very large particular examples of phrases, in a flat manner, and *then* learn the phrase structure within these phrases. At the other extreme, it could learn these phrases in a bottom up manner, always learning smaller phrases before learning larger phrases using these smaller ones.

The same observation cannot be made about the merging component. For merging there is a very definite dependency on the ordering. Once two classes are merged together there is no way to reach those classes in a future merge.

3.13.2 Language Space

While the grammars acquired by this system are context-free grammars, the set of all possible languages which can be learned, called the **language space**, is strictly smaller than the set of context-free languages. The grammar transformation process does not allow a merge to introduce recursion into the CFG rules. Thus, the resulting language is necessarily finite in the number of sentences it contains, and the language space contains only finite languages.

The language is transformed concomitantly with the grammar transformations, thus producing a stream of output languages L_0, L_1, L_2, ..., one for each grammar, which monotonically increase in their size. Figure 3-5 shows an *upper bound* on the language size of a particular inference run. The language size actually increases at a doubly exponential rate with the number of merges; thus the y axis is plotted on a $log(log(\text{language size}))$ scale.

3.13.3 Parsing and Ambiguity

In building the grammar through iterations of merges, the original training sentences are maintained as sentence rules in the grammar. With each merge these training sentences are further reduced according to the newly acquired rules. Because there is no room for ambiguity as these sentences are represented in the grammar, it is very unlikely for an acquired grammar to be ambiguous in how it parses a particular sentence. This is just a byproduct of how merges are done without allowing any room

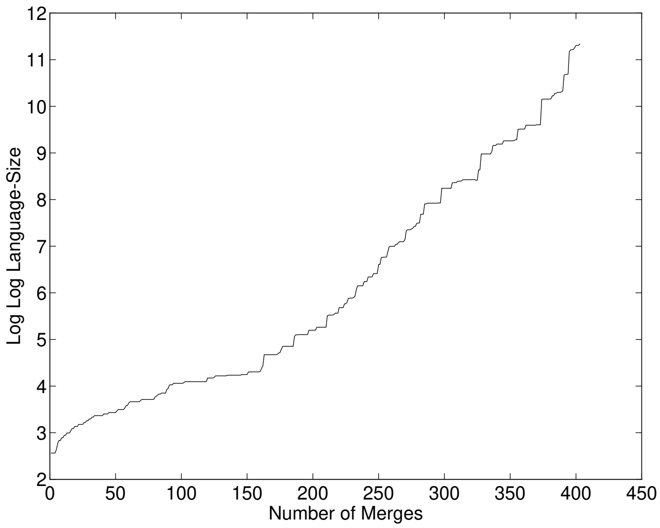

Figure 3-5: Language size

Upper bound on the number of sentences in the acquired language during the standard inference run. The language size is plotted on a log log scale on the y axis, and the number of merges is shown on the x axis. At $x = 0$ the language size is exactly the number of unique sentences in the training set.

for maintaining different possible parses of the training sentences.

This property is very useful as it allows us to develop a very simple parser to take a new test sentence and reduce it to its unique reduced form. This parser simply finds all rules which can apply to the current test sentence, and applies the rule which was acquired *earliest* during the inference process, and then reiterates. In this manner the test sentence will be parsed exactly as it would have been parsed if it were a training sentence.

This parser can be used to see if a test sentence is in the language using the following procedure: the test sentence is first reduced as far as possible using the *acquired* rules of the grammar. When this is done, we see if the resulting reduced form matches any of the reduced *sentence* rules stored in the grammar. If so, the test sentence is in the language of the grammar.

While the nature of this inference process discourages acquiring an ambiguous grammar, there can be sequences of merges that do result in an ambiguous grammar. Fortunately, this seems empirically not to happen very often. Unfortunately, there is no algorithm to determine whether an arbitrary CFG is ambiguous [30] — this decision problem has been proven *undecidable* by the theoretical community. Thus, we simply hope that the grammars are not too ambiguous since this ambiguity will cause problems when we use the grammar to develop a probabilistic language model, as discussed in the next Chapter.

3.14 Results

There are numerous ways to examine the results of the inference process. Probably the most immediately rewarding subjective result is the actual grammars that are acquired. Figures 3-6, 3-7, and 3-8 show the grammar acquired after 200 merges with the standard inference run. The classes and phrase structure represented in this grammar seem quite reasonable, and it seems like the system is doing a good job learning the language. For example, NT_{29} from the grammar is especially interesting. It derives many phrases which ask for flight information, for example "i would like a

northwest	kind	washington d-c	sunday	kosher	ninth
lufthansa	type	washington	friday	vegetarian	eighth
midway	types	dallas-fort-worth	board	passengers	third
eastern	kinds	oakland	wednesdays	people	seventh
continental	sort	boston	saturday	now	fifth
american	today	san-francisco	tuesday	hello	fourth
delta	tomorrow	baltimore	monday	hi	second
united	stop in	philadelphia	thursday	evening	sixth
these	arrive in	pittsburgh	wednesday	morning	reserve
those	departing	denver	c-o	afternoon	arrange
weekday	leaving	atlanta	e-a	end	buy
weekend	breakfast	dallas	d-l	beginning	rent
please list	dinner	b-w-i	u-a	begin	aircraft
list	lunch	chicago		ending	airplane

Table 3.2: Example acquired classes
Example word and phrase classes as acquired by the standard inference run.

flight", "can you give me all the flights", and "what flights are available" all derive from this non-terminal.

Another way to look at the grammar is to simply enumerate the actual words and word phrases which fall into the same class. This is only practical for small classes because the number of units in each class can become very large even with a few depths of recursion in the grammar. Table 3.2 shows some examples of words and phrases which are derived from the same grammar.

For more objective evaluations, we can examine the test set coverage and the net grammar size, as a function of the number of merges and the two parameters M_c and S. We are unfortunately limited in this because we have no way of knowing at what point each inference run is overgeneralizing. It is nonetheless interesting to see how these quantities progress.

We can also watch how the distortion, or divergence, for each merge varies. Initially we expect the merged units to be very close, and then to become further apart over time.

Another parameter we can watch is N, the number of units in the grammar. How N evolves with the merges reflects to what extent the acquired grammars are classing

$NT_0 \Rightarrow$ end | beginning | begin | ending
$NT_1 \Rightarrow$ kind | type | types | kinds | sort
$NT_2 \Rightarrow$ i'd | i would
$NT_3 \Rightarrow$ washington d-c | washington | dallas-fort-worth | oakland | boston | san-francisco | baltimore | philadelphia | pittsburgh | denver | atlanta | dallas | b-w-i | chicago
$NT_4 \Rightarrow$ wanted | wish
$NT_5 \Rightarrow$ NT_{21} is | what's | what is | what are
$NT_6 \Rightarrow$ NT_8 NT_3
$NT_7 \Rightarrow$ largest | smallest
$NT_8 \Rightarrow$ NT_3 to
$NT_9 \Rightarrow$ kosher | vegetarian
$NT_{10} \Rightarrow$ i NT_{60} | NT_2 like
$NT_{11} \Rightarrow$ give | tell
$NT_{12} \Rightarrow$ last | earliest | latest
$NT_{13} \Rightarrow$ now | hello | hi
$NT_{14} \Rightarrow$ california | colorado | georgia
$NT_{15} \Rightarrow$ q-w | q-x | y
$NT_{16} \Rightarrow$ nineteenth | thirtieth | twelfth
$NT_{17} \Rightarrow$ ninth | eighth | third | seventh | fifth | fourth | second | sixth
$NT_{18} \Rightarrow$ northwest | lufthansa | midway | eastern | continental | american | delta | united
$NT_{19} \Rightarrow$ reserve | arrange | buy | rent
$NT_{20} \Rightarrow$ sunday | friday | board | wednesdays | saturday | tuesday | monday | thursday | wednesday
$NT_{21} \Rightarrow$ what time | when | where | how NT_{22}
$NT_{22} \Rightarrow$ long | much
$NT_{23} \Rightarrow$ can | could
$NT_{24} \Rightarrow$ please NT_{11} | NT_{11}
$NT_{25} \Rightarrow$ show | NT_{23} you NT_{24} | NT_{24}
$NT_{26} \Rightarrow$ these | those
$NT_{27} \Rightarrow$ from NT_6 | between NT_3 and NT_3
$NT_{28} \Rightarrow$ stopping | with a stopover
$NT_{29} \Rightarrow$ NT_5 the flights | what flights are available | NT_5 NT_{33} | NT_{10} a flight | NT_{25} NT_{34} | NT_{10} NT_{50}
$NT_{30} \Rightarrow$ passengers | people
$NT_{31} \Rightarrow$ cheapest | least expensive
$NT_{32} \Rightarrow$ NT_{44} flight | NT_{12} flight | NT_{44} fare
$NT_{33} \Rightarrow$ the NT_{32}

Figure 3-6: Acquired grammar, part 1
Grammar acquired after 200 merges during the standard run (figure 1 of 3).

$NT_{34} \Rightarrow NT_{100}$ the flights | NT_{100} all flights
$NT_{35} \Rightarrow$ weekday | weekend
$NT_{36} \Rightarrow$ breakfast | dinner | lunch
$NT_{37} \Rightarrow$ itinerary | costs | prices
$NT_{38} \Rightarrow$ connections | enroute
$NT_{39} \Rightarrow$ spend | visit
$NT_{40} \Rightarrow$ price | name | shortest
$NT_{41} \Rightarrow$ live | stay
$NT_{42} \Rightarrow$ scratch | sure
$NT_{43} \Rightarrow$ the NT_{51}
$NT_{44} \Rightarrow NT_{31}$ one-way | NT_{31}
$NT_{45} \Rightarrow$ t-w-a | u-s-air | NT_{18} airlines
$NT_{46} \Rightarrow$ depart | leave
$NT_{47} \Rightarrow$ arrivals | departures
$NT_{48} \Rightarrow$ aircraft | airplane
$NT_{49} \Rightarrow$ arrival | departure
$NT_{50} \Rightarrow$ to NT_{94}
$NT_{51} \Rightarrow$ evening | morning | afternoon
$NT_{52} \Rightarrow$ fourteenth | thirteenth | twentieth | seventeenth | sixteenth | eleventh | tenth | fifteenth | eighteenth | twenty NT_{17}
$NT_{53} \Rightarrow NT_{28}$ in | through | via
$NT_{54} \Rightarrow$ thirteen | completed | hold
$NT_{55} \Rightarrow$ connect | originating
$NT_{56} \Rightarrow$ international | logan
$NT_{57} \Rightarrow$ one | nine | eight | five | six | seven | four | three | two
$NT_{58} \Rightarrow$ makes | include | under
$NT_{59} \Rightarrow$ zero | fifty | forty | oh
$NT_{60} \Rightarrow$ need | want
$NT_{61} \Rightarrow$ b | c
$NT_{62} \Rightarrow NT_{18}$ flight | NT_{45} flight
$NT_{63} \Rightarrow NT_{1}$ of NT_{48} is | NT_{48} is
$NT_{64} \Rightarrow$ its | reservations
$NT_{65} \Rightarrow$ select | standard

Figure 3-7: Acquired grammar, part 2
Grammar acquired after 200 merges of the standard inference run (figure 2 of 3).

$NT_{66} \Rightarrow$ atlanta's | dulles
$NT_{67} \Rightarrow$ describe | explain
$NT_{68} \Rightarrow$ okay | yes
$NT_{69} \Rightarrow$ stopovers | thanks
$NT_{70} \Rightarrow$ c-o | e-a | d-l | u-a
$NT_{71} \Rightarrow$ a-m | p-m
$NT_{72} \Rightarrow$ ninety | sixty | eighty | seventy
$NT_{73} \Rightarrow NT_{57}\ NT_{57} \mid NT_{57}\ NT_{59} \mid NT_{57}\ NT_{72}$
$NT_{74} \Rightarrow$ this flight | $NT_{76}\ NT_{57}$
$NT_{75} \Rightarrow$ approximately | thousand
$NT_{76} \Rightarrow NT_{62}\ NT_{73}$ | flight NT_{73}
$NT_{77} \Rightarrow$ after | before
$NT_{78} \Rightarrow NT_{77}\ NT_{57}\ NT_{71} \mid NT_{77}\ NT_{88}$
$NT_{79} \Rightarrow$ stop in | arrive in
$NT_{80} \Rightarrow$ start | starting
$NT_{81} \Rightarrow$ today | tomorrow
$NT_{82} \Rightarrow$ transport | transportation
$NT_{83} \Rightarrow$ companies | rentals
$NT_{84} \Rightarrow$ a-t-l | midnight
$NT_{85} \Rightarrow$ eleven | ten
$NT_{86} \Rightarrow$ please list | list
$NT_{87} \Rightarrow NT_{86}$ all | which of NT_{26}
$NT_{88} \Rightarrow$ noon | $NT_{85}\ NT_{71}$
$NT_{89} \Rightarrow$ delta's | them
$NT_{90} \Rightarrow$ may | april | february | december | september | august | november | july | june | march
$NT_{91} \Rightarrow NT_{90}\ NT_{52} \mid NT_{90}\ NT_{17}$
$NT_{92} \Rightarrow$ airfares | choice
$NT_{93} \Rightarrow$ departing | leaving
$NT_{94} \Rightarrow$ fly | go
$NT_{95} \Rightarrow$ equipment | why
$NT_{96} \Rightarrow$ departs | leaves
$NT_{97} \Rightarrow$ nonstop | transcontinental
$NT_{98} \Rightarrow$ display | inflight
$NT_{99} \Rightarrow$ section | tickets
$NT_{100} \Rightarrow$ me all | me
$NT_{101} \Rightarrow NT_{27}$ on $NT_{91} \mid NT_{27}$
$NT_{102} \Rightarrow$ rate | route

Figure 3-8: Acquired grammar, part 3
Grammar acquired after 200 merges of the standard inference run (figure 3 of 3).

versus building structure. This will be very relevant in the next chapter.

3.14.1 Test Set Coverage

As the inference process generalizes, we expect the coverage on the test set to increase, because more sentences are being added to each language. However, the inference process also over-generalizes at the same time, thus causing over-generation of the acquired grammar. We would like to select a grammar that achieves reasonable coverage of the test set while minimizing over-generation.

Figure 3-9 shows the evolution of test set coverage for the first 600 merges of the inference process with $M_c = 5$ and $S = 100$. The coverage starts at 10% because 10% of the test sentences appear verbatim in the training set. For the first 50 merges the coverage seems to rise *very* quickly, and then rises at a more or less steady rate for the rest of the merges. This pattern is typical for all coverage traces as M_c and S vary. In Figure 3-9 the system eventually reaches about 35% coverage after 600 merges. If we consider merges beyond 600, this coverage does not improve too much. Furthermore, we have strong reason to believe (from the next chapter) that already by 600 merges the acquired grammar is over-generalizing substantially.

While this result is somewhat disappointing from the perspective of learning the precise target language, coverage is more an academic issue because very good probabilistic models can be developed based on these grammars. In the next chapter we will explore such models.

The test set coverage depends on both parameters M_c and S of the inference process. Any test sentence which contains a word whose count is *below* M_c does not have much of a chance of being included in the acquired language because this word will never be merged during the inference process. Such a sentence can only parse if there is a training sentence which also contains that word and which otherwise reduces to the same form as the test sentence. Thus, as a function of M_c, the number of test sentences which *do not* contain any word whose count was below M_c is roughly an upper bound on the achievable test set coverage. Figure 3-10 shows this upper bound on test set coverage as a function of M_c. In this respect, we wish to make M_c

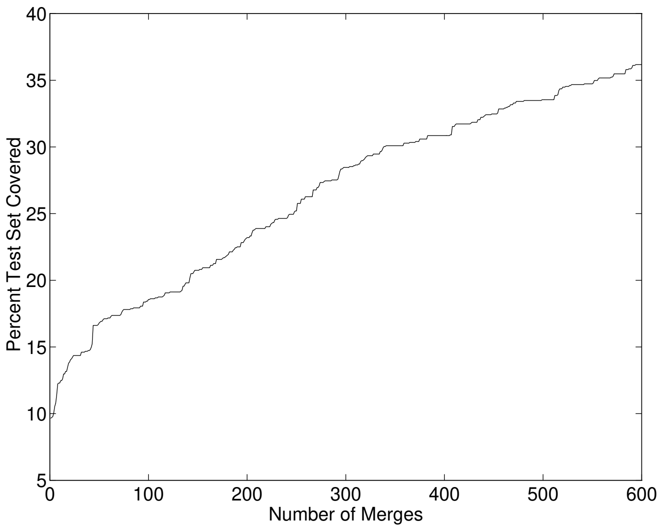

Figure 3-9: Test set coverage
Evolution of test set coverage for the standard inference run. Test set coverage increases substantially, which means the grammars are indeed generalizing correctly beyond only the training sentences.

as low as possible to have hope of covering a substantial portion of the test set.

On the flip side, decreasing M_c means that the estimates of the distributional probabilities for units is less reliable. We would expect this to have a detrimental effect on how well the system learns the language, because units are being merged based on unreliable estimates of their similarity. Decreasing M_c also causes N_a to increase, thus there are more merges to consider, and plots like Figure 3-9 are not directly comparable.

The test set coverage will also depend on S. If $S = 0$, the system is only able to learn word classes. Figure 3-11 shows the evolution of test set coverage with the number of merges for $M_c = 5$ and $S = 0$. Since language does seem to have some natural word classes, this will increase coverage to a point. But once the natural word classes have been learned, future merges can only over-generalize tremendously. This causes the coverage to increase tremendously as well, but at the expense of over-generation. That is what generates the sharp knee in Figure 3-11.

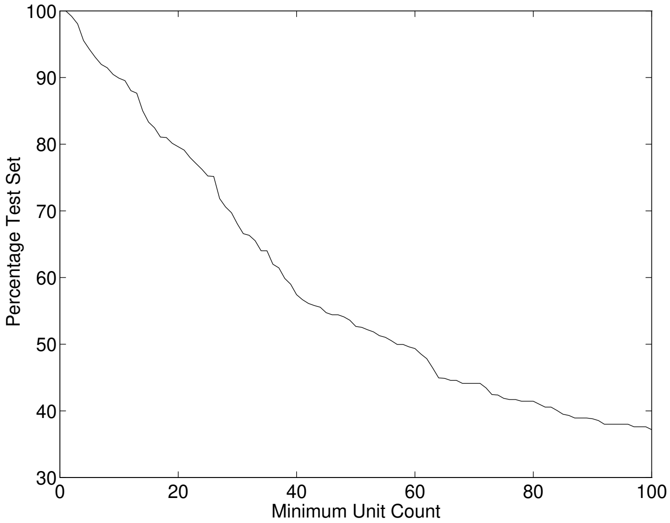

Figure 3-10: Test set coverage limit

Percentage of test sentences *not* containing a word whose count is below the count cutoff shown on the x axis. This quantity is of interest because it is an upper bound on achievable test coverage when we choose M_c.

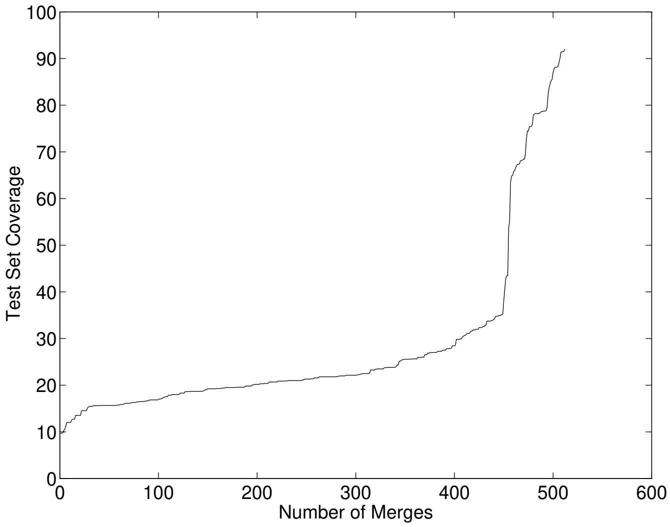

Figure 3-11: Word-class-only test set coverage

Evolution of the test set coverage for word-class-only inference with a count threshold of 5 ($S = 0$, $M_c = 5$). The coverage increases very sharply towards the end as words are nearly collapsed to one class.

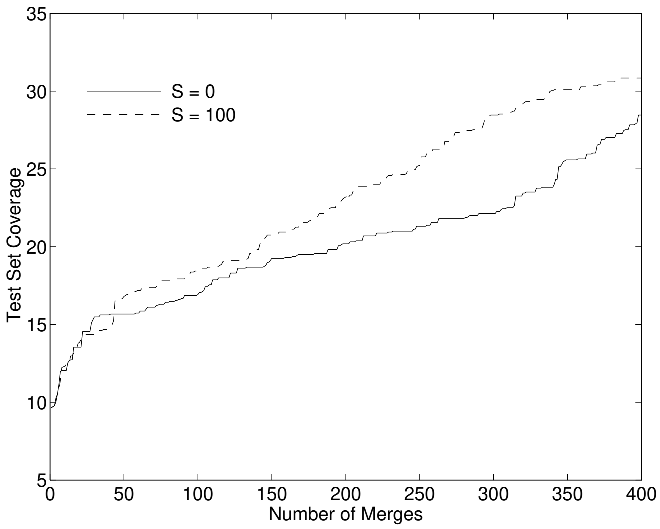

Figure 3-12: Comparison of test set coverage
This figure shows the evolution of test set coverage during the first 400 merges for two choices of the phrase set size: $S = 0$ (word-class-only) and $S = 100$. With $S = 100$ the inference process is better able to model the language structure, hence coverage is higher.

As S increases, the inference system is better able to model the language because it is able to incorporate phrases that compose the natural structure of the language. Figure 3-12 compares the test set coverage for $S = 0$ and $S = 100$ for the first 400 merges. During this time, the inference with $S = 100$ achieves higher coverage because language really does have phrase structure. But the word class only inference will eventually catch up because it over-generalizes substantially.

Figure 3-13 shows how the choice of S impacts the eventual coverage of the system. It is interesting that as S increases the asymptotic test set coverage decreases. All of these coverage plots tend to taper out to an asymptote, which is very interesting. Since the language size increases with each merge, this must mean that the merges are mainly causing the grammars to over-generate beyond that point.

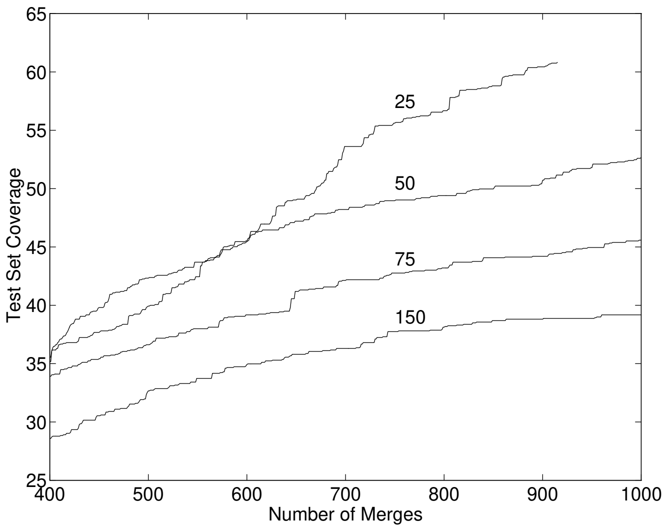

Figure 3-13: Test set coverage *vs.* phrase set size
The effect of phrase set size S on eventual test set coverage (for merges 400 to 1000). As the phrase set size increases, the asymptotic test set coverage decreases (minimum count threshold $M_c = 5$).

3.14.2 Grammar Size

The underlying motivation of the inference process is Occam's Razor, or "the simplest explanation is best". This is manifested in our inference process by the decrease in net size of the grammar through the iterations. While acquired rules are added to the grammar, the reduced sentence rules are being reduced to the same form, thus allowing us to discard the duplicate rules. Empirically, the second effect far outweighs the first, and the grammars decrease in size substantially. For these purposes, we measure the grammar size simply as the sum of the number of elements on the right sides of all rules in the grammar.

It is interesting to note that reduction of grammar size is *not* the explicit goal of the inference system. The goal is to choose similar units and merge them together. This has the side effect of substantially reducing the grammar size, which, by Occam's Razor, is an indication that our inference system is finding a good grammar to model the target language.

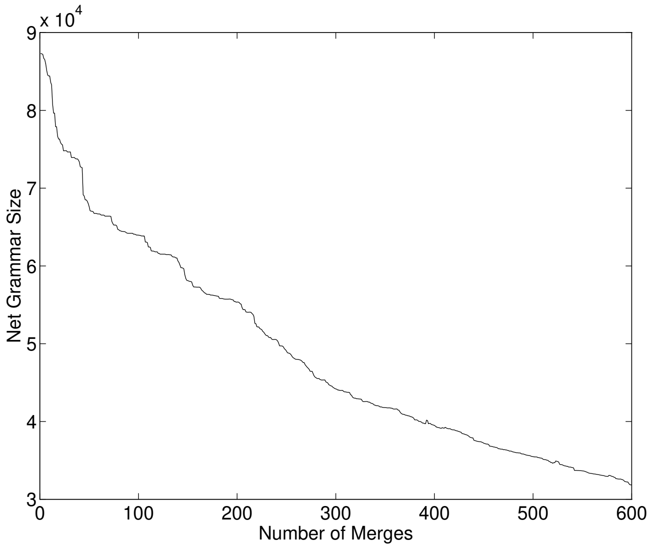

Figure 3-14: Grammar size
The evolution of the grammar size for the standard inference run. The grammar size decreases substantially in keeping with Occam's Razor, "the simplest explanation is best".

Figure 3-14 shows the evolution of the grammar size with the number of merges, for $M_c = 5$ and $S = 100$. Indeed, the grammar becomes very much smaller through the merges. But, at some point, the grammar is being oversimplified, which is causing it to over-generate. Again, we do not know where that point is.

All other plots of grammar size have roughly the same shape. Just as was the case with test set coverage, as M_c increases, any training sentence containing a word whose count is below M_c will most likely never be reduced to the same form as another sentence. Thus, the higher M_c, the higher the asymptotic limit on how small the grammar may become.

3.14.3 Number of Grammar Units

The evolution of the number of grammar units N reflects how much classing versus structure building the grammar is doing. If there is little structure building and lots of class building then N will decrease substantially. If the grammar has a lot of

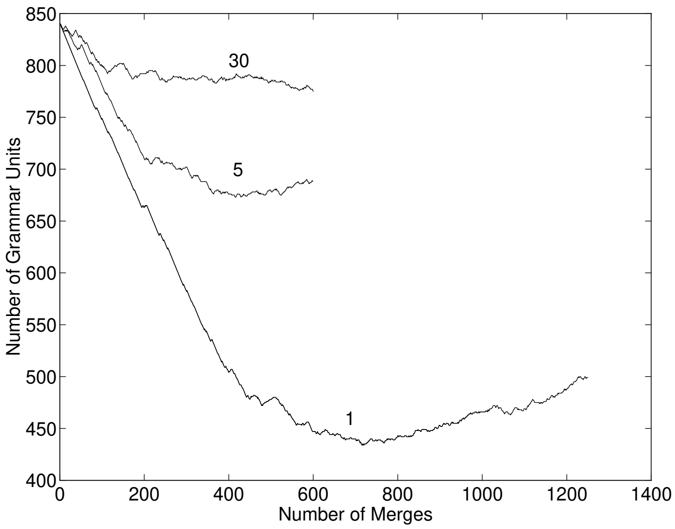

Figure 3-15: Number of grammar units *vs* minimum count threshold M_c
Effect of the minimum count threshold M_c on the evolution of the number of grammar units during the standard inference run. As M_c decreases more word classes are created, so the number of grammar units decreases substantially.

structure and relatively few classes, then N will increase. It seems that for all choices of S and M_c, the evolution of N tends to first decrease substantially as the grammar acquires word classes, then begins to level off or increase as the system acquires more phrases. This would seem to indicate that the language is actually composed in this structure.

Figure 3-15 shows how N varies as a function of M_c. By reducing M_c we are allowing the merging component to choose units which are relatively infrequent. Because of the bias of the phrase set to phrases which occurred frequently, these additional infrequent words are usually merged into word classes. Thus, decreasing M_c allows N to decrease more during inference.

Different choices for the parameter S also affect the evolution of N. This is shown in Figure 3-16. As S increases, N tends to decrease less, which is an expected effect. By offering the merging system more phrases to choose from, it chooses to merge phrases more frequently, thus limiting the reduction in N.

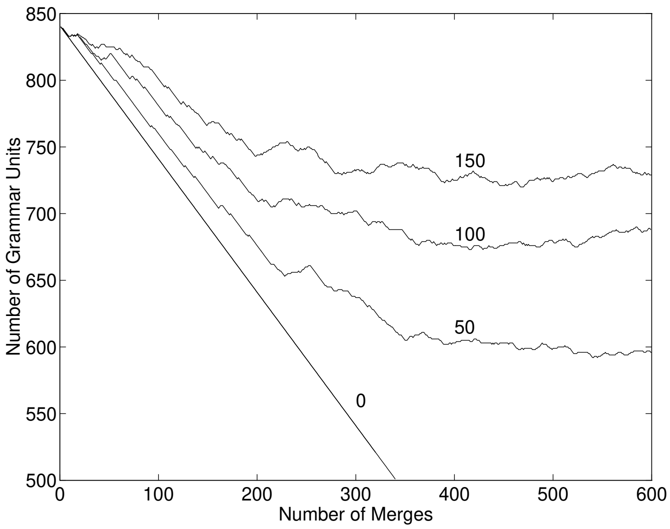

Figure 3-16: Number of grammar units *vs.* phrase set size S
Effect of the phrase set size S on the evolution of the number of grammar units during the standard inference run. As S increases more of the merges involve phrases thus limiting how much reduction occurs in the number of grammar units.

3.14.4 Distortion

The distortion between merges tends to increase over time because at each point we greedily select the merge with the least distortion. This increase is not monotonic, though, because a merge can spontaneously create a new class which is closer to other classes in term of divergence. Figure 3-17 shows the divergence values for each merge for the inference run. This value has been smoothed to make the overall pattern more visible. It is interesting that the distortion increases quite smoothly; there is no sharp point where the divergence becomes substantially larger.

3.15 Summary

In this chapter we have described the first step towards acquiring a language model. This step is an iterative grammar inference process which creates a sequence of grammars from a set of example training sentences. At each iteration, the current grammar

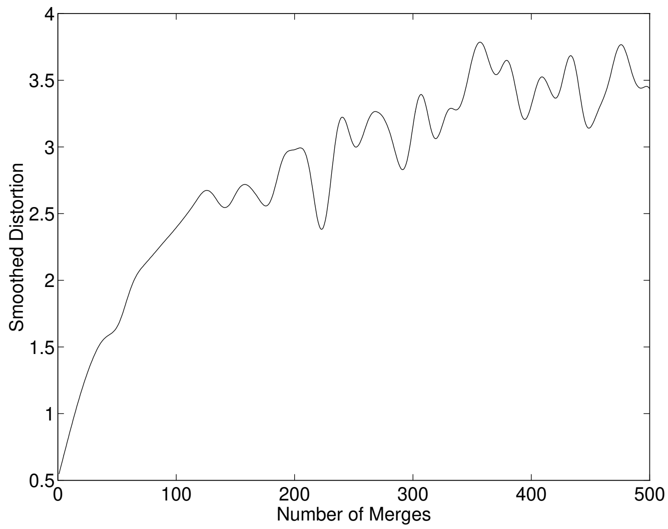

Figure 3-17: Merge distortion
Evolution of divergence distortion during the standard inference run. It gradually increases with the number of iterations. The plot is smoothed with a window size 30.

is transformed by *merging* two similar units together, thus generalizing from the previous grammar. Each resulting grammar therefore accepts *not only* all of the training sentences but also sentences beyond the training set. This allows the coverage of the independent test set to increase with each merge. It begins with a grammar which directly encodes the training set and then proceeds to iteratively generalize the grammar by merging similar units. This generalization also simplifies the grammars, thus reducing their size with each iteration. This is in keeping with Occam's Razor, or "the simplest explanation is best", which is the underlying philosophy of all inference algorithms.

At some point the grammars in the output sequence will become overly general and will very much over-generate and accept sentences beyond the actual target language. While we have no means of measuring the extent of this error, the next chapter will construct a probabilistic model from these grammars which will allow us to indirectly measure this extent of over-generation.

There are two important parameters which affect the outcome of this inference process. The first parameter is the minimum unit count M_c, which is a lower bound on the frequency of units which can be considered for merging. The second parameter is the phrase set size S, which is how many phrases at each iteration the merging component is allowed to choose from. The computational complexity of this algorithm is sensitive to both of these parameters.

Our interpretation of the results of the inference process is necessarily hampered by the fact that we have no idea which grammar is "best" is in the output grammar sequence. This sequence begins with grammars which are very specific because they predict little more than the training set, and ends with very general grammars which predict nearly all sentences. Somewhere in the middle is a "best" grammar in some sense. The next chapter will allow us to make this determination.

The results of this algorithm are quite impressive. The grammars and phrase classes which it acquires look very reasonable and intuitive. The size of the grammar decreases substantially through the inference process even though that is not directly the criterion for selecting merges. According to Occam's Razor, this implies that the inference process is indeed constructing a good grammar for the training set. At the same time, the test set coverage increases — this implies that the grammars are generalizing in *good* directions. While this test set coverage seems somewhat low, the next chapter will derive useful probabilistic models from these grammars which achieve full coverage.

Another curious result which was examined was the evolution of the number of grammar units N with each iteration. The number of these units reflects the relative proportion of the two functions of the grammar: classing and structure building. For all of the inference runs, we observed N decrease initially as the system favored acquiring word classes, and then level off as more phrases were chosen for merging. This indicates that the language really follows this sort of structure: words initially distribute into word classes, and then these word classes compose phrases.

We saw how the minimum unit count M_c affects the test set coverage and the evolution of N. With a small M_c the grammar favors even more strongly placing words into classes, thus the evolution of N drops quite a bit further before levelling off. S also affects both the test set coverage and the evolution of N in reasonable ways.

Chapter 4

Phrase Class n-gram

4.1 Introduction

The output of the inference process is a stream of acquired grammars, each generalizing a little over the previous one, representing likely structures and classes of the target language. Each grammar G in this stream denotes a language in the formal sense – i.e., a sentence is either accepted or rejected. From this grammar, and from the original training set, we will construct a probabilistic model which attempts to approximate the true underlying distribution of the language source. We can then objectively measure the overall performance of this system (inference plus probabilities) by measuring the perplexity and complexity of the resulting language models. From the stream of grammars we can produce a stream of probabilistic language models.

There have been many approaches developed to utilize a given CFG to produce a probabilistic model. For example, TINA, PLR, and SCFG, which were described in Section 2.11, achieve *exactly* that goal. Producing a grammar is only the beginning in building a language model – how probabilities are then assigned is also a difficult issue which is far from solved at present. Because of the unique nature of the grammars acquired by the inference algorithm, and because of the limitations of other structural approaches, we developed a new formalism called the Phrase Class n-gram model, or PCNG. Using standard language model evaluation techniques we can then evaluate the quality of the PCNG models derived from the grammar stream produced by the inference process. The performance of the PCNG models will then allow us to evaluate more objectively the performance of the inference process, and in particular, allow us to choose the "best" grammar in the sequence as that grammar which minimizes perplexity.

The PCNG model is entirely independent from the inference process, and is a valid model in its own right. It is therefore entirely possible and reasonable to use a hand-written grammar instead of the inferred grammars as input to the PCNG model.

4.2 Overview

Grammars acquired by the inference system have a unique structure because of the nature of the process by which they were created. The grammar was acquired in a bottom up manner, where words and phrases were first grouped into classes, then used to create higher classes, etc. As this bottom up acquisition is happening, the original training sentences are reduced and collapsed, but remain as the top level rules in the grammar. It is only the acquired rules which reflect the learning being done and which assign useful structure and classes to the language. The sentence rules remain as flat top-level rules only because they are legacies of the original training sentences. Furthermore, the sentence rules are extremely large and represent the bulk of the grammar size.

Because of these negative attributes of the sentence rules, when creating a phrase class n-gram model we will temporarily set aside the sentence rules from the grammar and retain *only* the acquired rules in the grammar. Just like the SCFG model, every rule in this grammar will be assigned a probability representing the likelihood that that rule (instead of any other applicable rules) is used when it applies. These probabilities will then help to compute the probability of a test sentence.

Without the sentence rules the grammar *cannot* ever reduce a test sentence to the single start symbol S. Therefore, the PCNG model uses the sentence rules which were set aside from the input grammar to train an n-gram model *directly* on their reduced forms. The n-gram model serves to *generalize* beyond the actual training sentences such that *any* sequence of units will receive some probability. While the input grammar may have had very limited coverage on an independent test set, the generalization accomplished by the n-gram model allows *full coverage* of the PCNG model on the test set. This n-gram probability will also contribute to the final probability of the test sentence.

To display the results of an inference process we can plot a **perplexity trace** for the run which shows the perplexity of PCNG model derived from each grammar in the inference output. These plots show the number of merges completed on the x axis, and the perplexity of the PCNG model derived from each grammar in the merge sequence on the y axis. All perplexities are computed with respect to the *independent* test set described in Section 1.2.

4.3 Details

The PCNG model makes two passes to determine the probability of a test sentence W. During the first pass the test sentence is reduced as much as possible using only the *acquired* rules of the input grammar G. This reduction process uses the deterministic bottom-up parsing strategy described in Section 3.13.3. During the reduction we compute the product of the probabilities of the rules used to reduce the sentence, just as in the SCFG model. This component of the probability is referred to as the **spatial** probability $\hat{P}_s(W)$.

The second pass uses an n-gram model to assign a **temporal** probability $\hat{P}_t(U)$ to the reduced sequence of units U, where n is an input parameter to the PCNG model. This n-gram model is actually derived from the sentence rules in the grammar, and is therefore defined over all grammar unit sequences. The probability of the test sentence is then the product $\hat{P}_s(W)\hat{P}_t(U)$. The acquired grammar rules are the mechanism for translating the sentence from the domain of word sequences to the domain of grammar unit sequences.

For example, consider the sentence "show me the least expensive flight from boston to denver" and the PCNG model containing the grammar, with associated rule prob-

$NT_0 \Rightarrow$ boston [.25]
$NT_0 \Rightarrow$ denver [.1]
$NT_1 \Rightarrow$ least expensive [1.0]
$NT_2 \Rightarrow NT_1$ flight, [.5]
$NT_3 \Rightarrow$ show me, [.5]

Figure 4-1: Example rules and probabilities.

abilities, shown in Figure 4-1. This sentence will be reduced to the sequence of units "NT_3 the NT_2 from NT_0 to NT_0". The probabilities of the rules used for this reduction are multiplied together to compute the spatial probability: $\hat{P}_s$("show me the least expensive flight from boston to denver") $= .25 \times .1 \times 1.0 \times .5 \times .5 = .00625$. Using this reduced sequence we then compute the n-gram temporal probability, and return as the probability of the sentence $\hat{P}_s(W)\hat{P}_t(U)$.

Because of the n-gram phase of the PCNG model, the PCNG model is a full coverage model, meaning it will assign a non-zero probability to every possible sentence. Thus, although this model is structure based, it will not have the robustness problems confronted by other structural language models. At the same time, the PCNG model achieves a substantial reduction in complexity because it is able to group words and phrases into equivalence classes. The PCNG model can be seen as a step from simple word n-gram models towards fully structural models. How large this step is depends on how completely the grammar tends to reduce sentences.

This model is referred to as a *phrase class* model because in the input grammar, the non-terminals implicitly represent an equivalence class of words and *word* phrases. This class contains all words and word phrases which can be derived from the non-terminal, along with the probability associated with each such phrase as computed during the derivation. Thus sum of the probability of each word and word phrase in each class will be 1. In assigning probability to a test sentence, the PCNG model makes the assumption that the occurrence of a particular word or word phrase is *independent* of the context in which it occurs *given* the class representing it. This is precisely like the word class n-gram model, extended to allow word phrases to be included in classes as well.

4.3.1 Training

The PCNG model needs to be trained to estimate the two different set of parameters — rule probabilities and n-gram probabilities. This training process proceeds as follows. Each sentence in the training set is reduced as far as possible using the input grammar G. During this reduction, each rule in the grammar records how many times it was used for the entire training set. These counts are represented as $C(A \Rightarrow \alpha)$ where $A \Rightarrow \alpha$ is a rule in the grammar.

Once this is finished, the counts are used to estimate a probability for each rule. If there are any rules whose count is below a minimum count threshold C_{min}, the count for that rule is raised to C_{min}, where C_{min} is typically 1.0 for our experiments. This guarantees that each rule has some non-zero probability. The probability of each rule is estimated as follows:

$$\hat{P}(A \Rightarrow \alpha) = \frac{C(A \Rightarrow \alpha)}{\sum_{(A \Rightarrow \gamma) \in G} C(A \Rightarrow \gamma)}$$

This guarantees that probabilities of all rules sharing the same left hand side sum to 1,

$$\sum_{A \Rightarrow \alpha} P(A \Rightarrow \alpha) = 1$$

which is necessary for the PCNG model to be a valid model. It is entirely possible for a rule to have probability 1, which means that no other rules in the grammar have A as their left hand side.

The now reduced training set is then used to train a simple n-gram model, as described in Section 2.9. This reduced training set is actually *very* similar to the sentence rules in the input grammar G. In particular, the sentence rules from the grammar are exactly equal to the reduced training set with the duplicate reduced forms removed. Because the n-gram is sensitive to word frequencies, it is necessary to retain the information as to which sentence rules were duplicated how many times. Thus, the n-gram model is effectively trained on the sentence rules from the input grammar. This can be seen as a means of *generalizing* from the sentence rules to achieve full coverage of all sequences of units. This step was necessary because the *actual* test-set coverage of these grammars was quite poor, as described in the previous chapter.

Because this n-gram model is computed over sequences of grammar units, its complexity is directly correlated with the number N of grammar units. N, in turn, varies with the kinds of units merged in each iteration of the inference process, as described in Section 3.11. Thus, we achieve a reduction in the complexity of the PCNG n-gram model over the word n-gram model if the acquired grammar performs more classing than structure building.

The additional parameters of the PCNG model are the input grammar and the rule probabilities. The number of parameters in the n-gram model tends to far outnumber the number of rules in the grammar. Thus, reduction in the n-gram complexity typically outweighs the addition of rule probabilities. Figure 4-2 shows how the model complexity of the bigram and trigram PCNG models evolves during inference. In this plot, the y axis plots the *relative* model complexity, which is what percentage of a normal word n-gram's number of parameters are required for the PCNG model. This plot starts at 100% for both the bigram and trigram PCNG because after zero merges the models are just the word bigram and trigram models. It then decreases substantially for both, and more so for the trigram. The bigram complexity then begins to rise again because the number of rules in the grammar is increasing with each iteration.

4.3.2 Validity

Before trusting the perplexity of the PCNG model, we must first prove that the model is a valid one – i.e., that the net probability mass that it allocates to all possible sentences is at most 1:

$$\sum_{\text{Word sequences } W} \hat{P}(W) \leq 1 \tag{4.1}$$

We can show that this is the case by starting with the sum involving all *unit* sequences. The n-gram model guarantees that the total probability mass allotted to

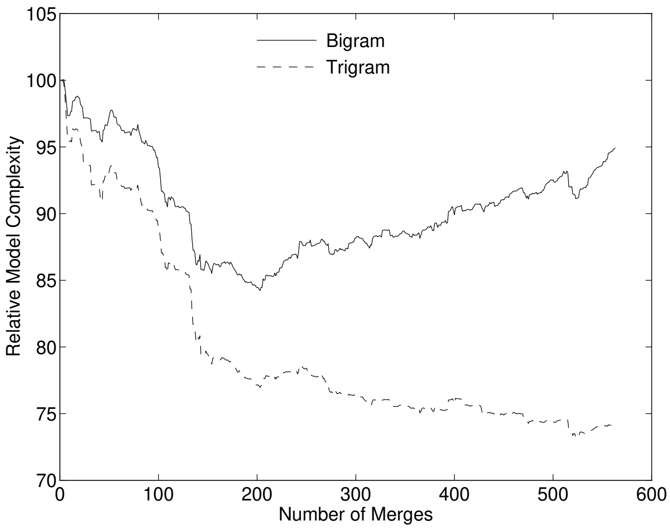

Figure 4-2: Model complexity
Relative number of parameters in the bigram and trigram PCNG models during the standard inference run, shown as a percentage normalized to the word n-gram model. Substantial reduction in complexity is achieved.

all sequences of units is exactly 1:

$$\sum_{\text{Unit sequences } U} \hat{P}_t(U) = 1 \tag{4.2}$$

We can define the set $D(U)$ to be all word sequences which are *derivable* from a unit sequence U. This set contains all combinations of possible expansions of the non-terminals in U. Let $\hat{P}_0(U, W)$ be the spatial probability associated with deriving W from U, where $W \in D(U)$. This quantity will be the product of the probabilities of the rules used in deriving W from U. The sum of $\hat{P}_0(U, W)$ across all $W \in D(U)$ is always 1:

$$\sum_{W \in D(U)} \hat{P}_0(U, W) = 1 \tag{4.3}$$

This can be proven by induction by showing that the sum of the probabilities of all word phrases derivable from a particular non-terminal is always one, and relies on the fact that rule probabilities for rules sharing the same left-hand-side non-terminal must sum to 1.

We can then look at the product of these two sets of probabilities, which must also be 1:

$$\sum_U \left(\hat{P}_t(U) \sum_{W in D(U)} \hat{P}_0(U,w) \right) = 1 \tag{4.4}$$

Let $R(W)$ denote the unique reduced unit sequence as produced by the deterministic parser. Every word sequence W maps to exactly one unit sequence $U = R(W)$. This implies that $W \in D(R(W))$ because the reduction process which mapped W to U is reversible. Also, $U = R(W)$ is a unit sequence and is will therefore be included in the summation in Equation 4.4. Consider the summation we wish to bound:

$$\sum_{\text{Word sequences } W} \hat{P}_s(W)\hat{P}_t(R(W)) \tag{4.5}$$

Because every element W in that summation is in *some* unit sequence $U = R(W)$, every term in the above summation *must* also be included in the summation in Equation 4.4. The above summation includes some subset of the terms in the summation in equation 4.4, which is bounded by 1. It follows:

$$\sum_{\text{Word sequences } W} \hat{P}(W) \leq 1 \tag{4.6}$$

It may very well be the case that the sum in equation 4.4 includes *additional* terms not included in equation 4.5. In this case the total probability mass will sum to *less than* 1, at the expense of perplexity. There are two cases which allow this to happen. The first case involves unit sequences which can *never* by realized by reduction of any word sequence W, but are nonetheless included in the summation in equation 4.4. This source of normalization error derives from the fact that the n-gram model allocates probability to *every* possible sequence of units U, even though some unit sequences cannot ever occur. For example, if the grammar has the rule "$NT \Rightarrow$ show me", then the unit sequence "could you show me NT_{14}" *cannot* occur, yet the n-gram model allocates some probability to it and that probability mass is lost. Fortunately, this value is quite small because the n-gram model will never see "show me" in the reduced training sentences.

The second case is when there exists a word sequence W which is contained *both* $D(U_1)$ and $D(U_2)$ for $U_1 \neq U_2$. This means the grammar is ambiguous for the word sequence W because there are two unit sequences which W can correspond to. While the inference process discourages the acquisition of ambiguous grammars, there are certain sequences of merges which can result in an ambiguous grammar. When this happens, any probability mass associated with a parse *different* from the parse chosen by the deterministic parser is *lost*. This normalization problem is more serious since the probability that is lost can be relatively substantial. This problem could easily be corrected by extending the parser to be more general so that it finds multiple parses of a given sentence. While this solution solves this normalization problem, we feel it is not necessary because this system does not acquire ambiguous grammars very easily, and the computational simplicity of the parser is valuable.

These implications mean that the perplexity traces that we plot are actually *upper bounds* of the true perplexity, which are the worst-case values for the perplexity.

4.3.3 Examples

It is helpful to understand what sorts of models the PCNG formalism produces when we consider different extremes of grammar choices. If the input grammar G contains no rules whatsoever, the resulting PCNG model is just the word n-gram model. Thus, when examining a perplexity trace for a particular inference run, the perplexity at $x = 0$ (x is the number of merges) will always be the word n-gram perplexity. If, at the other extreme, the grammar contains well written rules which somehow manage to parse every sentence they come across, then the model reduces to precisely a SCFG model.

Another extreme grammar is one which performs only classing operations. This means words are grouped into equivalence classes, and no rule has a phrase on its right hand side. In this case, the PCNG model based on this grammar will be identical to the word class n-gram model [11], as no sequence of two or more words will ever be

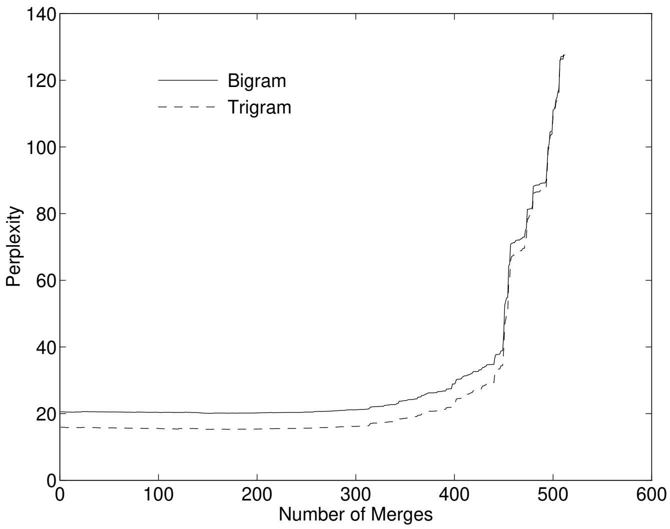

Figure 4-3: Word-class-only bigram and trigram perplexity trace
Evolution of the bigram and trigram perplexity for the word-class-only standard inference run. The sharp bend around 300 merges indicates that the language really does have certain word classes, but forcing these fundamental word classes to merge hurts the predictive power of the model.

reduced to a single non-terminal. This kind of grammar is produced by the inference process when the phrase set size S is set to 0 during inference. Figure 4-3 shows the full perplexity trace for this inference run. The figure shows the perplexity of both the bigram and trigram models, with $M_c = 5$. It is interesting to see that this curve has such a sharp bend in it. We take this as evidence that the target language really does have substantial word classes. As we collapse words into classes the predictive power of the model does not suffer too much, but as we continue to collapse words beyond a certain critical point, the perplexity suffers substantially.

The other extreme, in which the grammar performs no classing operations whatsoever, is also an interesting case. The grammar then contains only rules which build phrases, and every rule has a unique non-terminal as its left hand side. The right hand side of each rule will contain a phrase of words and non-terminals. This means the PCNG model will assign probability 1 to every rule, so the probability of any test sentence in this model is entirely determined from the temporal n-gram probability.

But the grammar can be reduced, without changing at all the resulting probability distribution of the PCNG model, such that every rule in the grammar has only words on the right hand side. Each rule in the grammar then maps a unique non-terminal to a unique word-only phrase. The PCNG model which results from this grammar is equivalent to the *word* n-gram model if the lexicon were extended to include "super words" representing the phrase for each non-terminal. For example, the grammar may consist of the rules:

$$NT_0 \Rightarrow \text{show me}$$
$$NT_1 \Rightarrow \text{what is}$$
$$NT_2 \Rightarrow \text{how many}$$

We could equivalently add three new words to the lexicon, "show_me", "what_is", and "how_many", and train a *normal* word n-gram, to achieve *precisely* the same probabilistic model. While it is not actually possible for the inference process to acquire a grammar of this form (since the merge operation always puts two units into a class), this has interesting implications for grammars which are written by hand.

4.3.4 Properties

When there are phrase rules in the grammar, the effect with the PCNG model is essentially to allow a given n-gram model to use more than just the previous $n-1$ words of context. For example, if the grammar has the rule "$NT \Rightarrow$ show me", then in the context of "show me", the bigram language model would ordinarily only see the word "me" as the context. Because of the above rule, however, the bigram component of the PCNG model will know that "show me" was the context. Likewise, in assigning the probability to a given unit occurring in a given context, the PCNG model predicts the probability of "show me" as a single unit, whereas the ordinary word n-gram model would predict only the probability of "show".

Thus, when we have a grammar with quite a few phrasal rules, a bigram model actually has access to quite a bit more context than just the previous word, which allows it to predict probabilities much like the trigram model. Because the word trigram perplexity is quite a bit lower than the word bigram perplexity, we expect this to help the bigram perplexity substantially.

The unigram PCNG model has some curious properties. The word unigram model makes the assumption that a given word has no probabilistic dependence on its context. This model is far from correct, as demonstrated by the substantially higher perplexity the unigram model has over the bigram or trigram models. The unigram model of a PCNG model has the curious property that the classes have absolutely no effect — i.e., the resulting temporal probability distribution is the same whether we group units into classes or not. Thus, the unigram PCNG model makes the assumption that a sentence consists of a sequence of independently drawn phrases. Instead of assuming the words are independent from one another, the unigram PCNG model breaks the sentence into a sequence of phrases which it then assumes are independent of each other.

Figure 4-4 shows the trace of the unigram perplexity for the standard inference run. It is very interesting to see that the unigram perplexity drops so substantially from 150 to 40. This substantial reduction in perplexity means that the inference process is actually breaking the language into phrasal fragments which really are more independent from one another than the original words. This is more a curiosity than a truly useful result because the bigram and trigram models have far lower perplexities.

4.3.5 Hand-written Grammars

While the PCNG model is motivated by the properties of the acquired grammars from the inference process, it can accept other grammars (e.g., hand-written grammars) so long as these grammars have some basic properties. The grammar must be a phrase class grammar, meaning it is written in a bottom up manner and it does not allow ambiguous parses. These requirements are not very strict and allow many present day properties of language models to be incorporated.

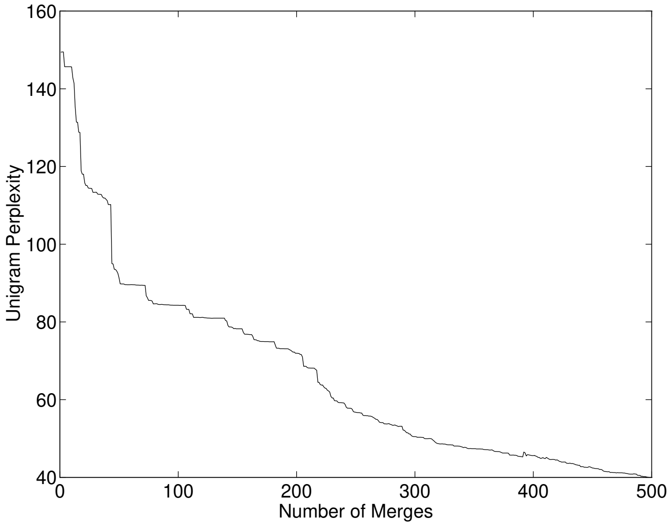

Figure 4-4: Unigram perplexity trace
Evolution of the unigram perplexity for the standard inference run. Substantial reduction is achieved because the PCNG model decomposes a sentence into phrasal fragments which are empirically more independent probabilistically than the words.

4.4 Evaluation

The inference process can be thought of as a "first pass" in acquiring a language model. It produces a grammar which tries to derive all sentences in the language. The "second pass" derives a PCNG model from that acquired grammar. If the first pass has poorly approximated the underlying language, then the second pass may be able to "smooth out" to some extent the limitations of the first pass. By measuring the quality of the overall language models, as a perplexity trace, we can then evaluate the performance of the inference process.

Consider the two types of error made by the inference process: under-generation and over-generation. Under-generation is something we can directly measure without probability by evaluating the coverage of the independent test set. However, over-generation we could not measure. By using the grammar to create a PCNG model we can now use perplexity to get at these two problems. Over-generation in the grammar will cause over-generation probabilistically, meaning probability mass will

```
what is restriction a_p five nine five
what flights leave after three a_m
i want to go from boston to dallas early a_m
flights from boston to denver arrive before three a_m
what is the cheapest fare from denver to boston
which of these the daily flight from denver to baltimore
show me the round_trip fare for flights arriving after four
show me fares
all_right do you have a list of flights into baltimore
i would like delta flight seventeen ninety is there a flight leaves san_francisco
    to san_francisco flights
seven p_m as trips
does delta airlines flight one oh three
please repeat that have a taxi from pittsburgh to denver on sunday
what flight do any limousines o+clock in oakland
what is the flights from boston to san_francisco one_way and san_francisco
book a flight from boston to san_francisco that stops pittsburgh before noon
beginning scenario c_m_u oh eight
show me the times eighteen
show me the afternoon
which flights with fares for united airlines flight two fifty three months
```

Table 4.1: Random PCNG trigram sentences
Random sentences from the PCNG model constructed after 200 merges of the standard inference run.

be allocated to sentences which are not valid, at the expense of valid sentences. This will therefore cause the perplexity to increase.

The under-generation problem is addressed by using the n-gram model in the PCNG model. While the grammar may under-generate and therefore fail to parse certain valid sentences, the PCNG model derived from that grammar blurs those sharp lines to achieve full coverage, thus accepting *all* sentences. However, not assigning enough probability to these sentences will result in an increased perplexity.

In this sense, mapping from a grammar to the PCNG model can offset to some extent any over-generation and under-generation that the grammar suffers from, by smoothing the rigid lines drawn by the grammar. But the extent to which the PCNG model is able to do this is limited by the structure of the grammar. Over-generation of the grammar will therefore lead to over-generation probabilistically, and the resulting perplexity will increase. In this sense, we can use the PCNG models to evaluate the over-generation of the acquired inference grammars.

4.5 Results

We can now use the PCNG model to objectively evaluate the quality of each grammar in the stream of grammars created by the inference process. We do this by creating the perplexity trace for the inference run, and determining that grammar which minimizes perplexity. This "best" grammar is one which generalizes to a reasonable extent, but not so much that the predictive power of the model suffers. Once we have this information, we can return to the inference process and reevaluate various quantities *at the point* of the "best" grammar.

The first results we show are the random sentences generated by a minimum perplexity PCNG model, as shown in Table 4.1. These sentences were generated from the trigram PCNG model after 200 merges of the standard inference run, which is the minimum perplexity grammar for this inference run. It is nice to see that these sentences look more reasonable than the n-gram random sentences shown in Chapter 2.

4.5.1 Perplexity Trace

We first computed the perplexity trace for the word-class-only case, i.e., with $S = 0$, and $M_c = 5$. Figure 4-3 shows both the bigram and trigram perplexity trace for the entire run. Figure 4-5 shows the trace for the first 300 merges of the bigram PCNG, and Figure 4-6 shows the same trace for the trigram model. It is very nice to see that the perplexity initially decreases for both models. For the bigram, the decrease was from 20.6 for the word bigram model to about 20.1 after 150 merges, a 2% decrease in perplexity. The trigram model starts at 15.9 as the word perplexity, and reaches a minimum of 15.3 after 150 merges, which represents a 3.8% decrease in perplexity. Both of these results are sizable.

The perplexity starts to increase because the inference process is merging classes together that are more and more dissimilar, thus forcing the model to over-generate. In all of these perplexity traces, the perplexity tends to drop at first, then reach a minimum point, then rise again. The reason for this is that at first the inference

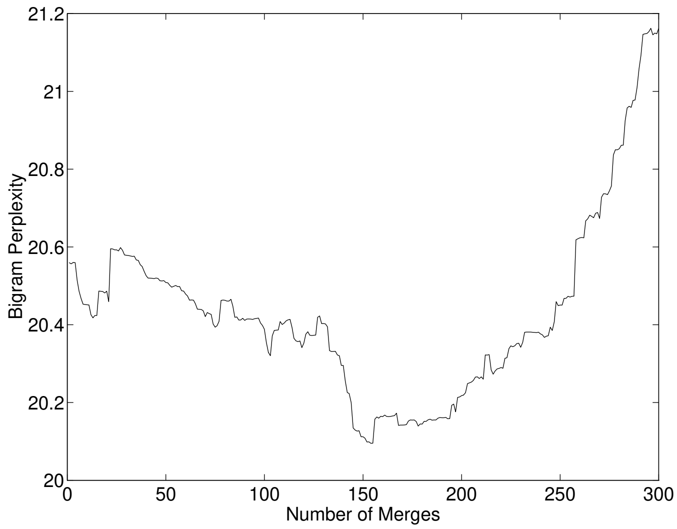

Figure 4-5: Word-class-only bigram perplexity trace
Bigram PCNG perplexity for the word-class-only standard inference run. A slight reduction is achieved over the word bigram model.

process is selecting units which are very appropriate for merging. When the units are merged their counts are shared according to the PCNG formalism. Their underlying distributions must actually be quite similar and sharing their counts allows more robust estimates of their distributions. But as the inference process proceeds, it is merging more and more dissimilar classes, which hurts the predictive power of the model; thus the perplexity eventually increases.

When we allow the inference system to consider phrases, the performance is better. Figure 4-7 shows the bigram perplexity trace for an inference run with $M_c = 5$ and $S = 100$. In the bigram case, the perplexity reduction is far more substantial than in the word class case, from 20.6 to 16.8 after about 350 merges – an 18% reduction in perplexity. The trigram perplexity drops from 15.9 to 14.9 after about 180 merges, a 6% reduction in perplexity.

There are a few curious properties to observe about these two graphs. First of all, the net drop in perplexity for the bigram model (3.6) is much more than that of the trigram (1.0). This is due to the fact that the bigram really appreciates having

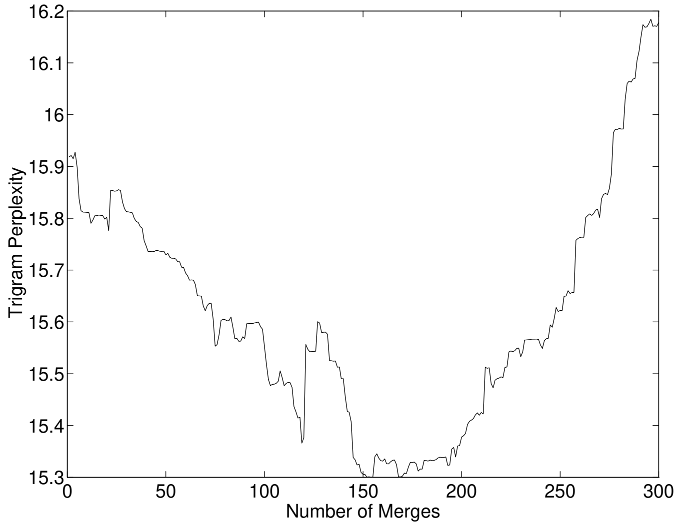

Figure 4-6: Word-class-only trigram perplexity trace
Trigram PCNG perplexity for the word-class-only standard inference run. A slight reduction is achieved over the word trigram model.

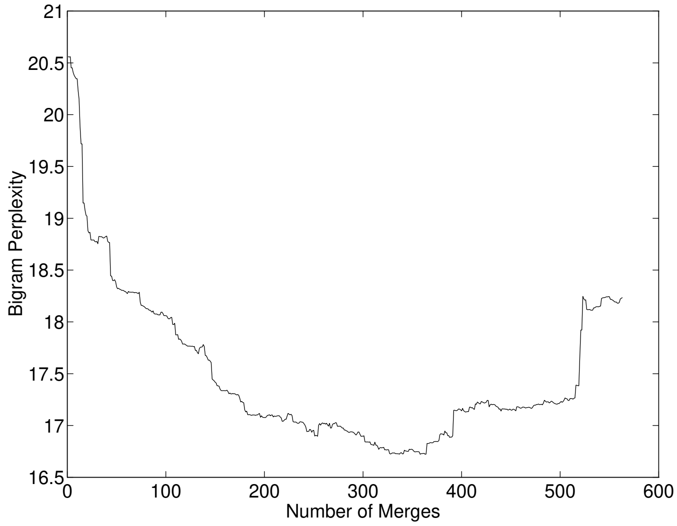

Figure 4-7: Standard run bigram perplexity trace
Substantial reduction is achieved because the phrases which are merged allow the bigram PCNG model to be conditioned on longer distance constraint.

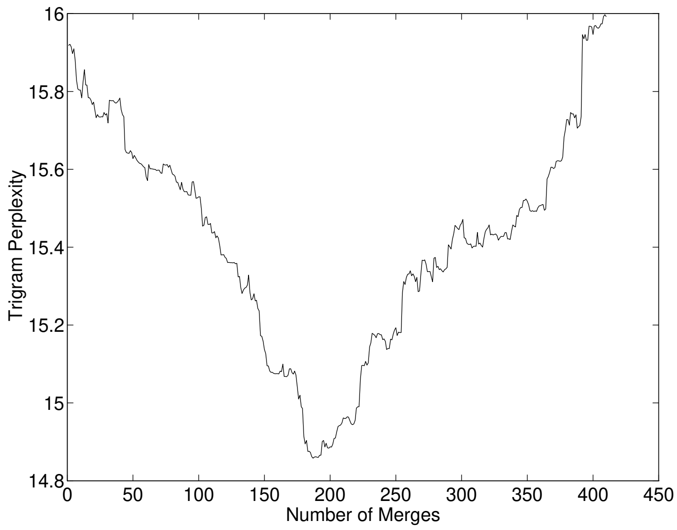

Figure 4-8: Standard run trigram perplexity trace
Appreciable reduction is achieved primarily because of the grammar collapsing many units into classes which allows more robust estimates of the trigram parameters.

phrasal units far more than the trigram model does. Also, because the perplexity is lower bounded by the true entropy of the language, the bigram model clearly has a high potential for improvement (the trigram model is evidence of this). When a phrasal unit occurs in a context, it provides the model with access to longer distance information than it would normally have access to, information which adds substantial constraint. This can be seen by comparing the word-class-only bigram trace with the trace in Figure 4-7. In the word-class-only case, the bigram perplexity only dropped 0.4, whereas when phrases are involved it drops by 3.6.

The trigram model does not benefit quite as much in involving phrases in addition to words. In the word-class-only case, the trigram perplexity dropped by 0.6, whereas in the phrase class case it dropped by 1.0, which is quite a bit better but not as much better as in the bigram case.

Second, the minimum perplexity of the trigram model occurs quite a bit sooner (after 180 merges) than for the bigram model (after 350 merges), and its minimum is quite a bit sharper than that of the bigram. The trigram perplexity also rises much faster, after reaching the minimum, than the bigram perplexity. Thus, depending on which model we wanted to use, we would stop the inference process at a different point.

As we change the inference parameters, M_c and S, the resulting perplexity traces look much the same — only the number of merges it takes to reach the minimum changes, and the value of the lowest perplexity changes. To get an idea of how large the phrase set size should be, we computed the perplexity traces for various different phrase set sizes with M_c fixed at 5, and found the minimum perplexity achieved in each of these traces for the bigram model and for the trigram model. Figure 4-9 shows the minimum bigram perplexity achieved as we vary the phrase set size, and Figure 4-10 shows the same plot for the trigram model.

It is clear from these plots that the bigram model very much benefits from having more phrases to choose from, but bottoms out at a phrase set size of about 100. The trigram model is less affected by the number of phrases each merge has to choose from. Somewhere between 60 and 100 phrases is the best size for the trigram model.

We can likewise see how the minimum achievable perplexity varies with M_c, the minimum count threshold. We tested various values of M_c from 1 to 50, keeping the phrase set size S fixed at 100. Figure 4-11 shows the resulting minimum bigram perplexity when M_c is varied, and Figure 4-12 shows the minimum trigram perplexity. Both exhibit exactly the same pattern that decreasing M_c can only reduce the perplexity further. We did no expect to see perplexity continue to reduce M_c became very small because we thought that allowing the inference system to consider words whose distributional probability vectors were not very robust would hurt the predictive power of the eventual PCNG model. It was surprising to see that instead the perplexity was very near the lowest for $M_c = 1$.

As we reduce the M_c parameter the inference algorithm must do more work to reach the minimum perplexity — it must iterate for a larger number of merges. Figure 4-14 shows the correlation between how long the inference algorithm must run to reach the minimum value and the setting of M_c, for both the bigram and trigram PCNG models. As M_c decreases, the number of merges necessary to reach the

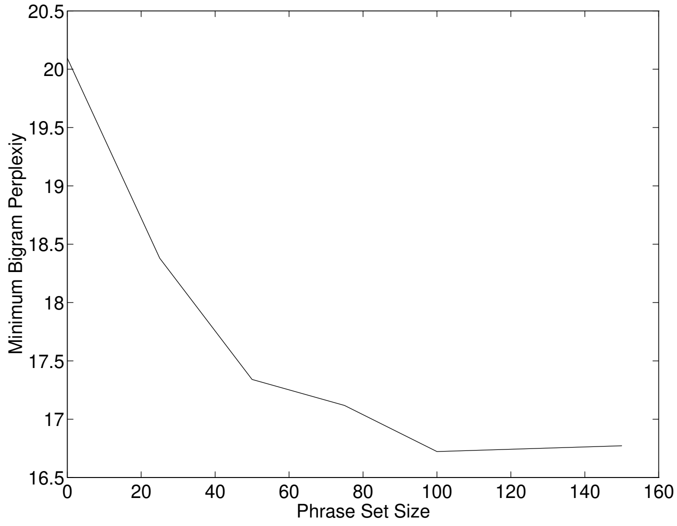

Figure 4-9: Minimum bigram perplexity *vs.* phrase set size S
Minimum bigram PCNG perplexity as the phrase set size S is varied, with minimum count threshold $M_c = 5$. Increasing S allows the bigram PCNG model to be conditioned on longer distance constraint thus consistently reducing perplexity.

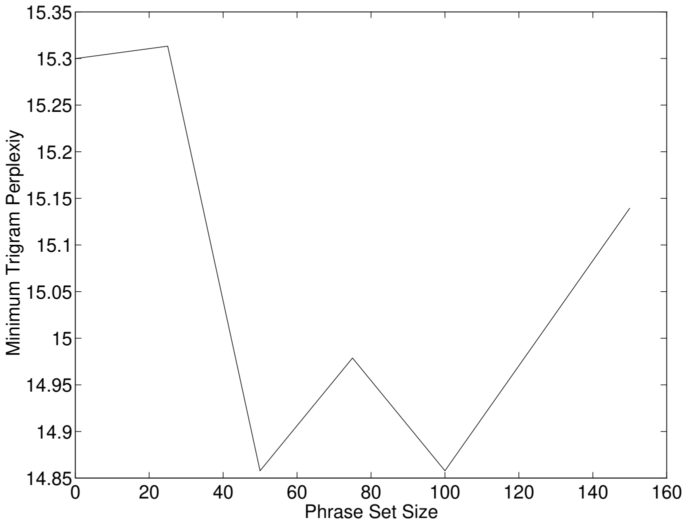

Figure 4-10: Minimum trigram perplexity *vs.* phrase set size S
Minimum trigram PCNG perplexity as the phrase set size S is varied, with minimum count threshold $M_c = 5$. The trigram PCNG model does not make much use of the phrases, thus increasing S beyond 50 does not help the minimum perplexity.

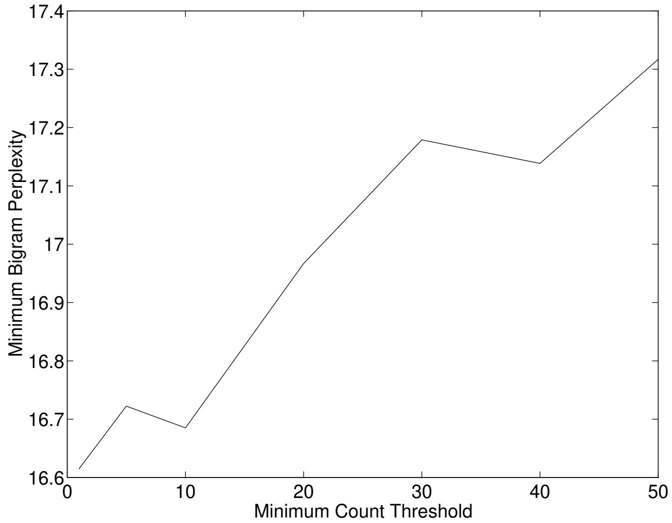

Figure 4-11: Minimum bigram perplexity *vs.* minimum count threshold M_c
Minimum bigram PCNG perplexity as the minimum count threshold M_c is varied. Decreasing the minimum count threshold during inference only improves the performance of the bigram PCNG model. Phrase set size S is fixed at 100.

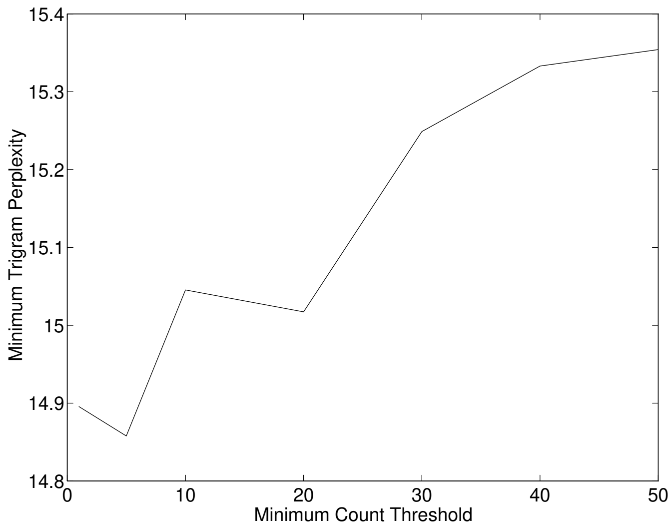

Figure 4-12: Minimum trigram perplexity *vs.* minimum count threshold M_c
Minimum trigram PCNG perplexity as the minimum count threshold M_c is varied. Decreasing the minimum count threshold during inference tends to improve the performance of the trigram PCNG model. Phrase set size S is fixed at 100.

minimum perplexity increases, especially as M_c becomes very low.

Figure 4-13 shows the correlation between the size of the phrase set and how long the inference algorithm (run with $M_c = 5$) must run to achieve the minimum perplexity for that perplexity trace. Allowing the inference process to choose from more phrases at each iteration increases the number of iterations necessary to reach the minimum perplexity.

4.5.2 Test Set Coverage

Because the PCNG model yields full coverage regardless of the coverage of the input grammar, the coverage of the grammar in the formal sense is not directly relevant to the PCNG model. In the previous chapter we were disappointed to see that the inference algorithm achieved such a low coverage on the test set. We now know, looking at the perplexity traces, that that coverage was achieved at a substantial cost of over-generation. Now, armed with minimum perplexity as a metric to choose the "best" grammar from the stream of grammars, we can return to the coverage issue and examine the test set coverage for the best grammar. Figure 4-15 shows the test set coverage of the minimum perplexity trigram grammars as M_c is varied and S is fixed at 100. The coverage is extremely low, around 23%, at the minimum perplexity points, which is surprising. There is a general trend for this coverage to decrease as M_c is increased, which is expected because increasing M_c excludes many words from the generalization process.

Figure 4-16 shows the same plot as we vary the phrase set size S, keeping M_c at 5. Again, the coverage is disappointing. There is a trend for this coverage to increase as the phrase set size is increased. This is an expected pattern because if the inference algorithm can choose from more phrases, it can better model the language.

4.6 Summary

In this chapter we described the PCNG model, which is a novel probabilistic language model well suited to the grammars acquired by the inference process. This model

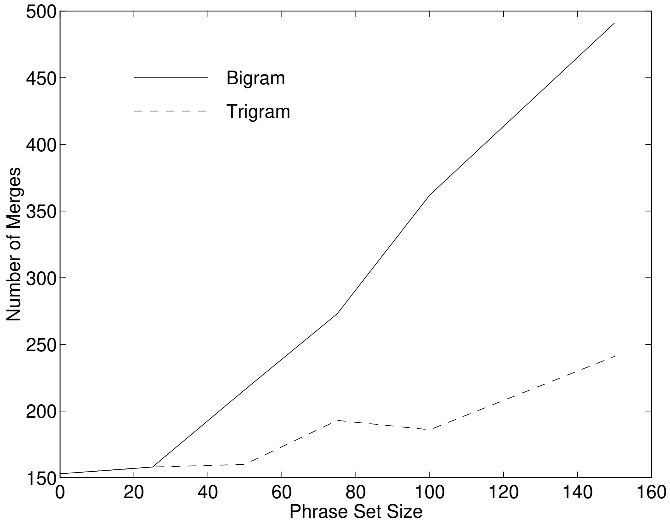

Figure 4-13: Minimum merge point *vs.* phrase set size S
Number of merges needed to reach the minimum perplexity of the bigram and trigram PCNG models as the phrase set size S is varied. The bigram model consistently requires more computation, and both models require more computation as S increases.

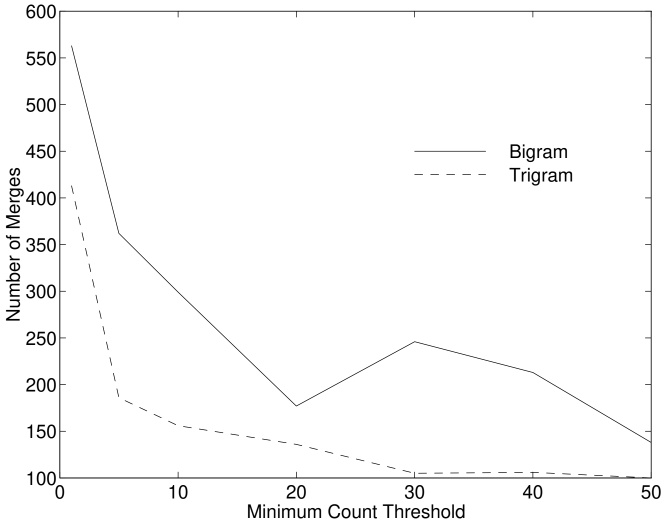

Figure 4-14: Minimum merge point *vs.* minimum count threshold M_c
Number of merges needed to reach the minimum perplexity of the bigram and trigram PCNG models as the minimum count threshold M_c is varied. The bigram model consistently requires more computation, and both models require more computation as M_c decreases.

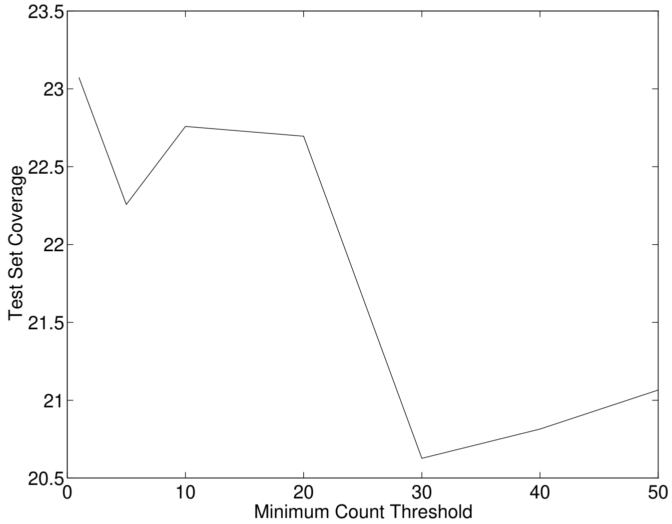

Figure 4-15: Test set coverage revisited
Test set coverage at the minimum perplexity point for the trigram PCNG model as the minimum count threshold M_c varies. Coverage is quite low, and increases slightly as M_c decreases.

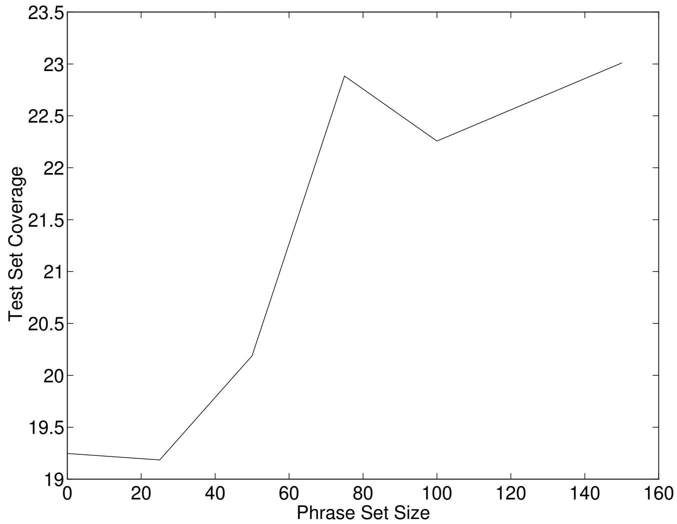

Figure 4-16: Test set coverage revisited
Test set coverage at the minimum perplexity point for the trigram PCNG model as the phrase set size S varies. Coverage is quite low, and increases slightly as S increases.

is a natural generalization of the word class n-gram model to phrase classes, and a hybrid n-gram/SCFG model. It is able to take advantage of the rules acquired by the inference process so as to reduce the complexity of the resulting model, but at the same time does not suffer from the robustness difficulties of other structural models because it achieves full coverage.

This model allowed us to construct perplexity traces of the grammar sequences acquired by the inference process and thereby select the best grammar from the sequence which achieves sufficient generalization while minimizing over-generalization.

This model, combined with the acquired grammars, achieved substantial reductions in perplexity of the trigram model and is thus very practical. The random sentences generated from this model seem more reasonable than the simple word trigram sentences. We observed that the bigram perplexity is reduced far more than the trigram, and that the bigram model benefits substantially by merging phrases in, and the trigram model less so. This allowed the trigram PCNG model to achieve its minimum perplexity after fewer merges than the bigram model.

Chapter 5

Speech Recognition

5.1 Introduction

The most direct objective measure of how well a language model models a language is to what extent that language model reduces the word error rate of a speech recognition system, when compared to other language models. However, this measure is not as simple as perplexity because it is a function of the particular speech recognition system. Furthermore, to evaluate a language model in such a framework requires integrating the language model into the particular search strategy employed by the recognizer, which can be a difficult process if the language model and recognizer differ.

We decided to evaluate the acquired PCNG language models by integrating them with the SUMMIT speech recognition system developed in the Spoken Language Systems group in the Laboratory for Computer Science at MIT [47]. SUMMIT is a segment based speech recognition system which typically employs class based n-gram models as the language model. [18]. We tested the performance of the system using acquired PCNG models and found them to improve recognition performance when compared to the word trigram model.

Not only is word accuracy of interest, but the computational cost is also important. A more powerful language model should do a better job constraining the search space so that less computation is necessary to find the best word sequence. However, a more powerful language model is also typically more expensive computationally. These two issues, computational cost versus word accuracy, need to be examined closely in deciding which language model to integrate into a recognition system.

5.2 Summit

The speech recognition system used to evaluate this system was the ATIS version of SUMMIT [47], which has been developed over the past six years in the Spoken Language Systems group at MIT. Summit is segment based, which means the incoming speech signal is explicitly segmented into likely phonetic segments before being classified into phonetic units.

Summit can be divided into two phases. The first phase breaks the incoming speech signal into likely phonetic segments and classifies each segment according to which phone it could be. This phase makes use of acoustic models to label each segment with a probability vector representing the likelihood that that segment is each of the possible phones. The output of this phase is a network of such segments. Summit typically uses on the order of 50 to 100 phones as the basic acoustic units to model the words.

The second phase, called the search phase, is responsible for taking this acoustic network and searching for a sequence of words that could have produced the speech signal. This phase is actually further broken into two steps. The **Viterbi** search is the first pass through the acoustic phonetic network to look for likely words. This search combines the constraints of the network, a lexicon, and a word bigram model, to search for likely words. It operates on the phonetic segments one at a time. The second step of the search process is a more thorough search which uses a more powerful language model. This search is done with an A^* search [3], and it operates with words as the fundamental units. It uses the results of the Viterbi search to help it through the search.

5.3 Search

5.3.1 Lexicon Graph

A critical component in the search phase of the recognizer is the **lexicon graph**. The lexicon graph stores all words which may be used by the speaker to create sentences, as well as all transitions from one word to another according to the word bigram model. Each word is represented as a graph such that multiple paths through the graph, from an initial node to a final node, correspond to different legal pronunciations of the word. Each arc of this graph is labelled with a single phone label and a score representing the cost of using that arc, or the likelihood of a pronunciation which uses that arc. The graph for each word is hand-written at first to create a base-form pronunciation. It then undergoes some automatic transformations based on phonetic pronunciation rules which attempt to model word contextual pronunciation effects [46]. The score on each arc is trained automatically using a corrective training process. The lexicon graph incorporates the word bigram model, which allows the individual word graphs to be interconnected with arcs labelled with the bigram score for the two connected words. These connections also handle any intra-word contextual effects.

A distinct word representing silence, **-pau-**, is required to start every sentence. Likewise, the silence word **-pau2-** is required to finish every sentence. These requirements, plus the connections between all word pairs by the bigram language model, define the large lexicon graph which describes all possible sentences at the *segmental* level. A particular sentence can have many different paths traversing through this graph because each word may have different pronunciations, but all such paths must start with one of the initial nodes of -pau- and end with one of the final nodes of -pau2-. The number of nodes in the lexicon graph will be defined as L, and will just be the sum of the number of nodes of all words in the lexicon.

5.4 Search Space

From the point of view of the recognizer, the search space contains not only all possible sentences which could be spoken, but for each possible sentence, all possible ways that that sentence could be pronounced and aligned with the recorded utterance. This is because the goal of the search phase of a recognizer is to find the best sentence and the highest scoring alignment *at the same time*.

This very large space may be represented as a large directed graph called the **alignment graph** G_s, with initial and final nodes. This graph is a function of the acoustic segment network produced by the first phase of recognition and the static lexicon graph stored within the recognizer. Each node of G_s is a pair $\langle n, b \rangle$ of a node n from the lexicon graph and segmental boundary b in the utterance. If the utterance has B boundaries, then this graph has $L \times B$ nodes. This graph has an arc from node $\langle n_1, b_1 \rangle$ to node $\langle n_2, b_2 \rangle$ if n_1 has an arc to n_2 in the lexicon graph *and* there is an acoustic segment connecting b_1 and b_2 in the acoustic network. This arc will be assigned a score which is derived from the score on the arc between node n_1 and n_2 in the lexicon graph and the probability of the segment between boundaries b_1 and b_2 in the acoustic network. The initial nodes consist of those nodes $\langle n, b \rangle$ whose boundary b is 1, and whose n is an initial node of -pau- in the lexicon graph. The final nodes $\langle n, b \rangle$ are conversely those nodes whose boundary b is the end boundary of the utterance, B, and whose node n is a final node of -pau2- in the lexicon graph.

There is a one-to-one correspondence between paths through this large graph and alignments of sentences with the acoustic evidence. Thus, we have reduced the problem of finding the best sentence and alignment to one of searching a large graph for the best path.

5.5 Viterbi Search

The Viterbi search [45] is an efficient search algorithm, based on dynamic programming, which finds the best path through G_s. It is a time synchronous search, which means it makes a single pass through G_s one boundary at a time. For each boundary it computes the shortest paths up until that boundary for *all* nodes in G_s corresponding to that boundary. As a side effect, the Viterbi search also produces a vector, for each acoustic boundary, representing the best path up until that boundary ending at every possible node in the lexicon. These vectors will be useful for later stages of the search.

For each boundary b the Viterbi search maintains the best path from the initial nodes of G_s to all nodes in G_s corresponding to b. It loops through all boundaries in time-order, from $1, 2, ..., B$, updating all nodes at that boundary before moving to the next one. For each node it searches for all incoming arcs to that node to see which would incrementally create the best path to this node, and records that maximum arc and resulting total score as a back-pointer. When the search is completed the best scoring node among the final nodes in G_s is selected, and the best path is reconstructed by following the back-pointers *backwards* of each node in the path.

The time and space complexities of the Viterbi search are fixed functions of L and B. This is an excellent property as it means we can always know how long and how much space the search will take to complete.

The Viterbi search is exhaustive to some extent because it computes the best path for *every* node/boundary pair. One of the critical reasons why the Viterbi search can accomplish this is that at each point in stores only the *very best* path. This unfortunately means that we cannot efficiently compute the top N best sentence alignments using the Viterbi search. Another reason for the efficiency of the Viterbi search is because it employs a very simple language model: the word bigram model. Unfortunately, other more complex language models can perform much better than the bigram model, in terms of perplexity, so we would like to be able to use those models. This leads us to the A* search.

5.6 A^* Search

Because the Viterbi search cannot easily integrate more powerful language models, SUMMIT uses another search strategy, called the A^* search [3], to tie in more powerful language models. Unlike the Viterbi search, the A^* search is a time *asynchronous* search which derives from the best first search formalism. It allows us to integrate more powerful language models *and* to compute the top N best scoring sentences from the recognizer. The A^* search uses the results of the Viterbi search to help constrain its choices. One disadvantage of the A^* search is that the worst case computation time can be exponential in the number of words in the sentence. Empirically, however, the time complexity is much better.

In many applications generating the top N best sentences is useful or necessary. For example, it is common to generate the top N best sentences from the recognizer and use a natural language parser to filter the N best sentences for one that parses. Another example application is one which allows more efficient experiments computing the effect of a new language model on recognition accuracy. This can be done by storing the N best outputs for each utterance and then *resorting* these outputs according to a new language model.

The A^* search maintains a queue containing many partial paths in G_s which have been explored so far. Each path describes some sequence of words, not necessarily an entire sentence, *and* how those words align with the acoustic network. At any point in the search, it is entirely reasonable that many paths have the same sequence of words but different alignments of those words with the acoustic network. At the beginning of the search, only the paths consisting of a single initial node in G_s are placed into the queue. At each iteration, the best scoring path is extracted from the queue and extended by all words which might follow that path. Each of these extensions creates a new path with a new score, and these paths are inserted into the queue. If the path extracted from the queue is **complete**, meaning it ends on a final node of G_s, then that path is reported as a complete path, and the search can continue searching for additional complete paths.

The A^* search uses a parameter D, called the **stack depth** to determine when it should stop searching. The A^* search will continue producing completed paths until the scores of the paths fall below the best scoring path *minus* D. As D increases, the number of paths N which can be computed from the A^* search increases sharply. Typically the A^* search is run until either N paths are completed, or the paths scores fall below the stack depth threshold, whichever comes sooner. D also has implications on the computational complexity of the search as it has an effect on what words each partial path should be extended by.

The A^* search in SUMMIT differs from the Viterbi search in that it works with *words* as the atomic units of a path. Every path is always extended by a word at a time, whereas the Viterbi search was done one *arc*, or acoustic segment, at a time.

The big difference between the A^* search and an ordinary best first search is the fact that the A^* search makes use of additional information in performing its search. This information is a function $h(p)$ which maps any path p to an upper bound on the scores of all possible completions of path p. The function h represents an additional source of knowledge which can "predict the future" of a particular path by identifying the score of the best possible completion of the path. When a newly-extended path p is inserted into the queue, the score of that path will be computed as a combination of its *true* score so far and its estimate $h(p)$ of its future.

If the function $h(p)$ is guaranteed to be a true upper bound, then the A^* search is said to be **admissible**. This means that the completed paths which are extracted from the queue are guaranteed be in decreasing order of their scores. In particular, the first complete path will be the best scoring path. If $h(p)$ cannot guarantee that it will always be an upper bound, then the A^* search using $h(p)$ is inadmissible, which means paths may complete out of order. If this is the case, one option is to run the search to generate the top N sentences, and then look for the best scoring path among those N sentences. If N is large enough, and if $h(p)$ is not "too far" from an upper bound, then the best path is often near the top of the N completed paths.

In SUMMIT, $h(p)$ is provided by an initial Viterbi search. In particular, a Viterbi search is first computed in the forward direction in time. Then, the A^* search searches the reversed direction in time (i.e., starting from the end of the utterance, moving backward to the start). Because the Viterbi search in the forward direction stores the best partial path up to each boundary, for every node, the A^* search can use the scores of the Viterbi search as its estimate of the future. One direct implication of this is that if the language model used during the Viterbi search (the bigram model) is mismatched with the model used during the A^* search (typically a word class quadgram model in SUMMIT), then the search will be inadmissible.

When a path is extended by each possible word, a language model is consulted, along with the acoustic models, to assign a score to that extension. The acoustic scores are comparable to log probabilities, so we combine the acoustic and language model score as follows:

$$S' = S + A + \alpha(\gamma + log\hat{P}(w_i|w_1...w_{i-1}))$$

where S is the score of the current path, S' is the score when we extend this path by w_i, A is the acoustic score of the extension, and $\hat{P}(w_i)$ is the language models probability of extending by the word w_i given the words so far in the path. The parameter γ is referred to as the **language model offset**, and the parameter α is the **language model scale factor**. Typically α is 70, and the γ is 2.7.

The language model in the A^* search can afford to be more costly computationally, and therefore more powerful predictively, than the bigram model used in the Viterbi search. This is because the number of path extensions during the A^* search for a typical utterance is far fewer than the corresponding number of extensions for the Viterbi search.

5.6.1 Computational Complexity

The A^* search can run in *exponential* time in the number of words in the sentence, which is unfortunate. In practice, however, the search tends to be quite efficient. The computational time of an utterance is typically measured by the number of path extensions accomplished to complete N paths for the utterance U. We will denote this function as $C_U(N)$ — it states how many paths had to be extended to complete N paths. Very little can be formally said about how $C_U(N)$ varies with U and N, but empirically we will see that it seems to grow somewhat slower than linearly.

Empirically, it seems the running time of the search is actually quite sensitive to the upper-bound function $h(p)$. If $h(p)$ is not a tight upper bound for every path p, the A^* search takes a *very* long time to complete N paths. For this reason, it is often quite a bit less expensive in terms of computation time to use an inadmissible $h(p)$ and resort the N outputs. The search is also *very* sensitive to the stack depth threshold D because D affects how many path extensions will be performed for each partial path. As D increases the computation increases sharply.

5.6.2 Upper Bound Bigram

In order to have an admissible A^* search we need to compute a word bigram model, for the Viterbi search, which guarantees that the scores assigned by this model are always an upper bound of the scores assigned by the PCNG model. This bigram-like model will **not** be a proper language model, as its probability mass can in general sum to more than 1. This is not a problem for the Viterbi search – i.e., we can still use the model because we are not using the answer paths that come from the Viterbi search as our answer. Thus, we will not refer to it as a bigram model, and simply call it a function $f_u(w_1, w_2)$ which returns a number between 0 and 1 such that for every possible word sequence $W = w_1...w_N$:

$$\prod_{i=1}^{N-1} f_u(w_i, w_{i+1}) \geq \hat{P}(w_1, ..., w_N)$$

where $\hat{P}(w_1, ..., w_N)$ is the probability that the PCNG model assigns to the word sequence. The word pairwise product of f_u must be greater than or equal to the probability assigned by the PCNG model.

Furthermore, we would like the upper bound to be as *tight* as possible in the interest of efficiency for the A^* search. This upper bound function f_u must be computed as a function of the grammar and rule probabilities and of the n-gram component of the PCNG model.

We can examine some simple cases first. If the grammar has no rules, and we are looking at a trigram model, then the PCNG model reduces to a simple word trigram model, such that the probability of a word depends on the previous *two* words. Computing f_u in this case is quite simple:

$$f_u(w_1, w_2) = \max_{w_0} \hat{P}(w_2 \mid w_0, w_1)$$

Since the upper bound function f_u has no knowledge of which word *preceded* word w_1 when looking at the word sequence w_1 w_2, it must assume the *worst*, or assume that it was the word w_0 that allows maximal probability of w_2 given w_0 and w_1. The quadgram case is similar, except the maximization is computed over all word pairs.

If we begin to add rules to the grammar, the situation becomes more complicated as there is more freedom in how we set f_u. For example, consider a grammar which contains only the two rules:

$$NT_0 \Rightarrow \text{one way } [0.6]$$
$$NT_0 \Rightarrow \text{round trip } [0.4]$$

If the PCNG model is using a bigram model, for simplicity, $f_u(w_1, w_2)$ could just be set to $\hat{P}(w_2 \mid w_1)$ for all bigrams *not* involving the words "one", "way", "round", and "trip". For any bigram involving one of these words we must consider all possible ways that the two above rules might apply to some sentence with that pair of words, and choose that parse which allocates the most probability to this pair of words as the value for f_u. For example, consider $f_u(the, one)$. "One" could be the start of NT_0 with the first rule applying, or it could be just the word "one" *not* then followed by "way". These two cases need to be separately considered to decide how each of them would allocate probability to the pair of words.

This notion can be applied recursively to handle grammars with more rules and deeper recursion in these rules. The general algorithm functions as follows. It will make two passes. In the first pass, it examines all non-terminals in the grammars and recursively computes which bigrams could be derived by these non-terminals along with the portion of the probability mass that should be allocated to these bigrams. For example, with the rules:

$$NT_0 \Rightarrow \text{one way } [0.6]$$
$$NT_0 \Rightarrow \text{round trip } [0.4]$$
$$NT_1 \Rightarrow NT_0 \text{ flight } [0.5]$$
$$NT_1 \Rightarrow \text{earliest } NT_0 \text{ flight } [0.25]$$
$$NT_1 \Rightarrow \text{cheapest } NT_0 \text{ fare } [0.25]$$

NT_0 would be record {(one way), (round trip)} as the bigrams it could derive, and NT_1 would record {(one way), (round trip), (way flight), (trip flight), (way fare), (way flight), (earliest one), (earliest round), (cheapest one), (cheapest round)}. These sets of bigrams are recursively computed. To divide the probability evenly we take the nth root of the probability for a given rule whose right hand side is length n and divide it evenly among all elements on the right-hand-side of that rule. This is equivalent to dividing the log probability by n.

Once this first pass is complete, each non-terminal has a list of the bigrams for which it can be responsible. For the second pass, we examine all n-gram sequences which occurred during training. For each of these, if a non-terminal is involved we compute how much probability should be allotted to all the bigrams derivable from that non-terminal, and maximize this over all possible n-gram sequences. We repeat this for the $(n-1) - gram$, etc, all the way to the unigram model.

We have some empirical results of this upper bound model, as shown in Figure 5-1. This figure shows a histogram representing how tight the upper bound model is per word in the test set. The mean of this plot is 0.61, which means on average the log probability (base e) assigned to a word by the upper bound function f_u is 0.61 in excess of what the real PCNG model assigns. The plot was made for a trigram PCNG model acquired after 190 merges of the standard inference run. Each sentence in our independent test set accounts for one element in the histogram.

Unfortunately, when we tested the performance of this system empirically, the A^* search could not even complete *one* utterance (within our patience limit, that is). It spent all of its time extending partial paths, never completing a path. The A^*

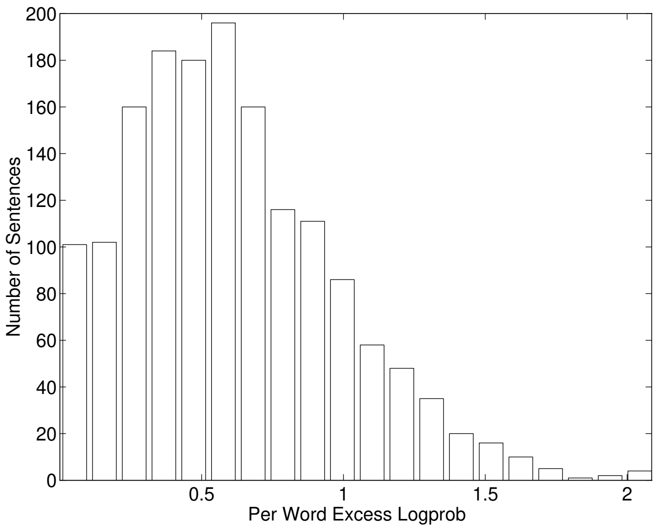

Figure 5-1: Excess log probability for upper bound bigram model
Per word excess log probability of function f_u for the phrase class trigram model.

search is far too sensitive to loose upper bounds, rendering this upper bound model impractical.

5.7 Language Model Integration

A fundamental computational issue in recognition is how a language model is integrated with the speech recognition system.

5.7.1 Full Integration

One approach, called **full integration**, uses the model directly during the A^* search. This is perhaps the simplest and most elegant solution, but it imposes some difficult constraints. First, the number of language model extensions which are typically computed during a direct A^* search is substantial. A language model which is expensive computationally will therefore incur a substantial penalty for this. Furthermore, the A^* search extends paths one word at a time. If the language model is not formulated in a word-by-word manner, this form of integration is difficult. The simple word n-

gram models or word class n-gram fit both of these criteria and are thus amenable to full integration.

5.7.2 N-best Resorting

Another means of integration is **N-best resorting**. In this framework the normal A^* search continues until a stack depth D or n paths, whichever occurs sooner, and records the completed paths that result. During this search it can use any simple n-gram language model. Then the new language model can look at the N-best list for each utterance and *resort* the paths according to its probability by replacing the language model component of the scores with its own scores.

This tends to be a computationally inefficient means of integration because computing the N-best lists takes substantial computation. Furthermore, the more mismatched the two language models are, the further from the top of the list the truly best path, according to the new language model, may be. Thus, as the language models become more disparate in how they assign probabilities to sentences, n must increase at the expense of computation. Where n is set is usually determined empirically by calculating the effect it has on word accuracy and computation.

One advantage of this approach is that the number of computations required of the new language model is very small in comparison to the other means of integration. Another advantage is that this form of integration is *very* easy to implement. The N-best lists can be computed one time and kept in storage. Then, as new language models are developed, they can be tested quickly to see how it affects recognition word accuracy. For these reasons, N-best resorting is typically only used for experimental language model development purposes. When a language model is determined to prove worthwhile by reducing word error rate, one of the other two integration schemes is developed.

5.7.3 Word Network Integration

The final means which we will discuss for language model integration is the **word network**. This form of integration actually falls between the other two in that it requires more computation from the language model than N-best resorting, but less than full integration. Furthermore, the word network is a more compact way to represent the same information of the N-best lists — it actually represents *all* completed paths to a stack depth D, which may correspond to a very large n. As D increases, the number of completed paths n which must be computed to achieve a stack depth of D can grow very quickly. The word network is thus a generalization of the N-best lists.

Word networks also have same advantage of N-best lists that they can be computed once and stored for later experimentation, but the same disadvantage for full integration that in searching a word network the language model must predict word probabilities one at a time. Their computational efficiency actually makes them attractive not only for experimentation purposes but also for real-time speech recognition.

A word network represents possible word alignments and scores within the utterance as a connected graph. It is actually computed using a modified A^* search, called the word network A^* search, where paths which end at the same word/boundary node in the alignment graph are *merged* together. It is this merging that creates a network of words instead of a simple N-best list as output from the search. During this search a simple language model, usually a word bigram, is used. Again, the word network A^* search is run to a depth D which guarantees that any paths whose score is within D of the best scoring path will be included in the word network.

We can then use the word network for an utterance as input to an A^* search using the more powerful language model. Because the word network has already limited the number of word arcs that need to be considered, this more powerful language model has to compute far fewer extensions than it would with full integration. The parameter D needs to be chosen as a function of how disparate the probabilities of the two models (the bigram model and the more powerful model) are on typical test sentences. As long as the difference in probability of the new model and the bigram model on a particular test sentence corresponds to less than D (after being scaled scaled by α), the new model can find the best path in the word network.

Therefore, for any language model we would like to use during word network integration, we can empirically compute how the model compares to the word bigram model over the test set of sentences, and use this comparison to determine a reasonable value for the stack depth D.

5.8 PCNG Integration

In this section we describe an approach to integrate a PCNG model with the A^* search. Since both the full integration and word network integration paradigms use an A^* search, what is developed here can be used for *both* forms of integration.

The word-by-word issue is the difficulty which the PCNG model must overcome. What is difficult about this is that the model does not know what the final parse of the sentence may be. Several possible rules may be able to apply depending on what words complete the sentence prefix. For example, consider a grammar containing the rules:

$$NT_0 \Rightarrow \text{one way flight } [0.5]$$
$$NT_0 \Rightarrow \text{one way fare } [0.5]$$
$$NT_1 \Rightarrow \text{one } [1.0]$$

The sentence prefix "show me the one" could end up parsing in different ways depending on how it is completed. The system has no way of knowing which could apply. Furthermore, this ambiguity could continue recursively if there are other rules which depend on NT_0 or NT_1. Somehow the PCNG model needs to be restructured so that it is able to provide a reasonable probability for the above sentence prefix.

One possible solution would be to place multiple paths in the queue representing the possible choices of future rule applications, with their different probabilities. This would mean that whenever there was ambiguity as to which rule may end up applying, we simply create as many duplicate paths as necessary to enumerate all possible outcomes, and place them in the queue with their different total scores. The problem with this approach is that it is wasteful of space in the queue, since there can actually be very many such possible parses.

A better solution is to develop a modified PCNG model which internally keeps track of all parses which could possibly apply to sentences containing the prefix. Then, the net probability of a particular word extension can be computed by adding the total probabilities allowed by all of the parses. An easier solution is to simply choose the partial parse which has the highest probability and report that as the probability for the word extension. These two approaches will *always* yield the same answers since a complete sentence, by definition of the PCNG model, has exactly one parse. Thus, whether we compute the sum or the maximum, that value will be the same when we extract a completed path.

We chose to implement this procedure using a modified version of Earley's parser [15], which is an efficient CFG parser. Earley's parser processes a sentence in one pass, keeping track of all possible rules applications at every word. Our word-by-word version of the PCNG model does precisely this for every path extension. It then searches through the possible rules that apply to determine that potential parse that has the highest score, and returns that as the language model probability for the word extension.

5.9 Results

For our recognition experiments we tried two forms of integration of the PCNG model with the recognizer: N-best resorting and word network integration. For N-best resorting we tested two different language models for generating the N-best list to begin with: a word bigram and a word trigram model. We test results with only trigram language models since they achieve the best results. We use three language models in this evaluation. One model is the baseline word trigram model, which had a test set perplexity of 15.92. The next model is the minimum perplexity grammar for the word-class-only inference run, with $M_c = 5$. This model has perplexity 15.33 on the test set. In this section we refer to this model as the **word class trigram** model. The third language model is the minimum perplexity grammar acquired during the standard inference run, which has perplexity 14.89 on the test set. This model is referred to as the **phrase class trigram** model.

For all of these experiments we use an independent test set consisting of 508 utterances. This set of utterances is independent from the training set *and* from the development test set used for the perplexity experiments. The word networks were computed with a stack depth $D = 800$. During the experiments we will vary the number of paths N searched to find the best path to study the effect this has on word accuracy and computation.

5.9.1 Language Model Disparity

We must first explore the probabilistic differences between the language models we would like evaluate and the bigram language model used to generate the word networks. The bigram language model was used to generate all of the word networks, and thus the word networks guarantee that any path within $D = 800$ of the top scoring path is represented within the word network. But if we then use a language model which differs substantially from the bigram model, the best path may not be in the word network a large portion of the time.

Figure 5-2 shows the disparity between the word trigram PCNG model and the word bigram model. This plot shows the trigram log probability minus the bigram log probability for each sentence. The scaling factor used for integration with the A^* search is 70, which means a stack depth of 800 corresponds to a log probability disparity of 11.43. It is very nice to see that almost all sentences fall below this point. Figure 5-3 shows a similar plot for the word-class PCNG model, and Figure 5-4 shows the plot for the phrase-class PCNG model. All of these models are similar enough to the bigram model to license using the word network search with a stack depth of 800.

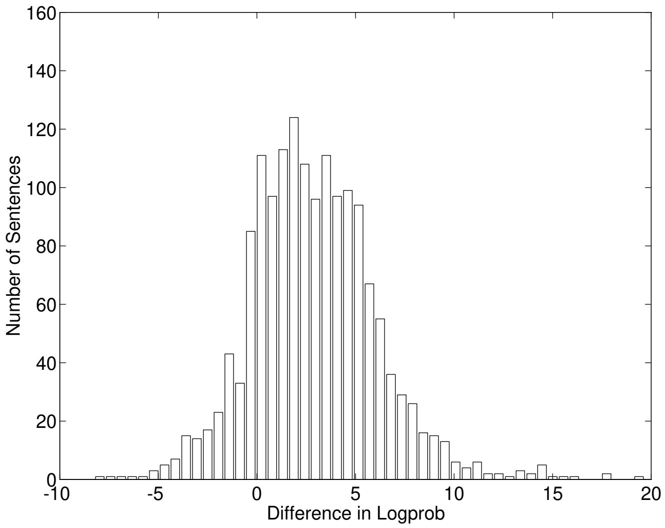

Figure 5-2: Trigram word n-gram disparity
This histogram shows how the word trigram and bigram PCNG differ in predicting probabilities for sentences. The plot shows the difference in log probability of the two models over all sentences in the development test set.

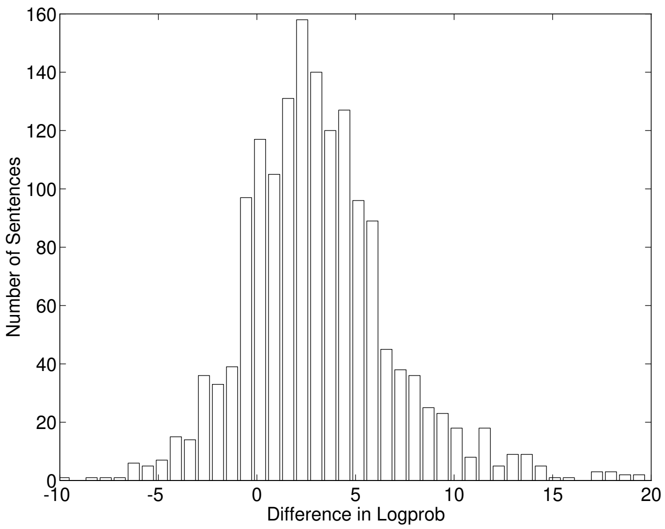

Figure 5-3: Trigram word-class disparity
Histogram of sentence log probability differences between acquired word class trigram model after 175 merges and the basic word trigram model. The differences fall almost entirely below the 11.5 cutoff.

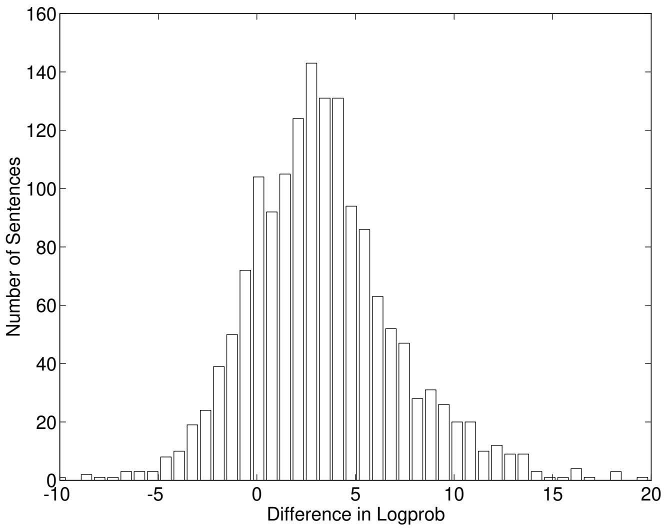

Figure 5-4: Trigram phrase-class disparity
Histogram of sentence log probability differences between acquired trigram PCNG model after 200 merges of the standard inference run, and the basic word trigram model. The differences fall almost entirely below the 11.5 cutoff.

5.9.2 Word Accuracy

We compute the **word accuracy** of the recognition system as a means to measure performance. Word accuracy is determined by finding the optimal alignment of the reference (answer) sentence to the recognized sentence, and determining how many substitution, deletion, and insertion errors were incurred. Word accuracy is then the percentage of correctly recognized words of all words in the reference sentences *minus* the percentage of words inserted into the recognized sentences. The **word error rate** is the sum of substitution, deletion, and insertion errors.

For each of the three different integration methods we need to use the top N paths to search for the best scoring path. None of the integration techniques are admissible, so we must consider the top N paths, and possibly resort them according to new language model scores. As we increase N we have a better chance of actually finding the true best scoring path, and we thus expect the word accuracy to improve as N increases. While this tends to be the case, it is *not* necessarily a monotonic

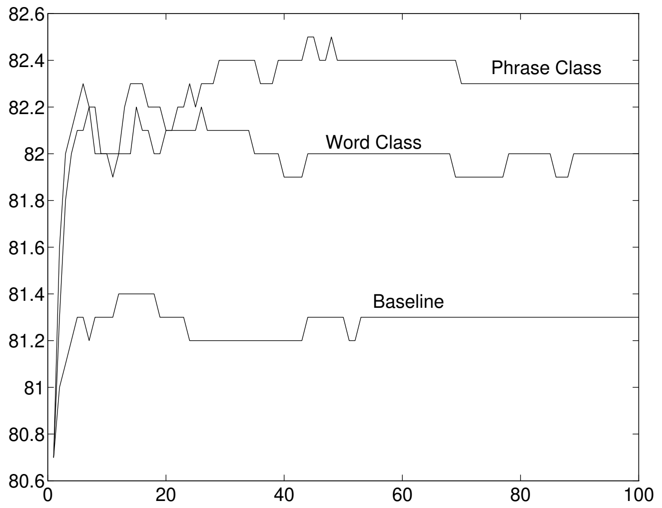

Figure 5-5: Word accuracy *vs.* number of completed paths N
This shows hows varying N changes the resulting word accuracy for N-best resorting. The two curves were computed by the trigram N-best resorting integration using the word class and phrase class PCNG models.

function because that best scoring path could very well be a worst output of the recognizer. Figure 5-5 shows how word accuracy improves with larger values of N, comparing the performance of the three language models.

The word accuracy clearly varies as a function of N in this Figure. For low values of N the word accuracy is quite a bit lower than its asymptotic value. However, once N reaches a certain point, the word accuracy remains approximately the same. This plot can be used to empirically determine how far to run the N-best outputs to achieve reasonable word accuracies. This plot empirically shows the *degree* of the inadmissibility. By observing that the word accuracy really does not change too much as N is increased beyond a certain point, we know that the extent of inadmissibility is limited. It is very nice to observe that the asymptotic word accuracy does indeed correspond to the perplexities of these language models. These results represent a substantial asymptotic reduction in the word error rate, from 18.7 to 17.7, which is a 5% reduction.

	Word Trigram	Word Class Trigram	Phrase Class Trigram
Bigram N-best resorting	80.8	81.5	81.6
Trigram N-best resorting	81.3	82.0	82.3
Word Network	81.3	81.8	81.6

Table 5.1: Asymptotic word accuracies
This table shows the improvement in word accuracy of recognition using the acquired word class and phrase class grammars

We can compute the asymptotic word accuracies for all nine combinations of language model and integration algorithm. Table 5.1 shows exactly this plot. One interesting feature which stands out with this plot is the disparity between the word accuracies of the bigram and trigram N-best integration approaches. Figure 5-6 compares these two integration approaches using the phrase class language model. The disparity is due to the fact that the bigram N-best integration is using a weaker language model than the trigram N-best integration, and because our values of N and D are too small. In theory, if we were to allow N and D to grow arbitrarily large, these two approaches must achieve exactly the same asymptotic word accuracy.

Another very surprising result from Table 5-6 is the substantial difference in word accuracy for the word network and trigram N-best integration. The direct word network search should allow for performance on par with the corresponding N-best resorting integration simply because the word network is a more compact means of representing N-best lists. One likely difference for this is because we are sorting the N-best list created by the word trigram model, which is more powerful than the bigram model used to create the word network. The 81.6% word accuracy in the corresponding bigram N-best integration is directly comparably with the word network result.

5.9.3 A^* Computation

As we vary N to choose a good point in terms of word accuracy, we pay for this with increased computation by the A^* search. This computation is typically measured by the number of path *extensions* which have to be completed to generate the N sentences. Figure 5-8 shows the number of extensions *per word in the test sentence*

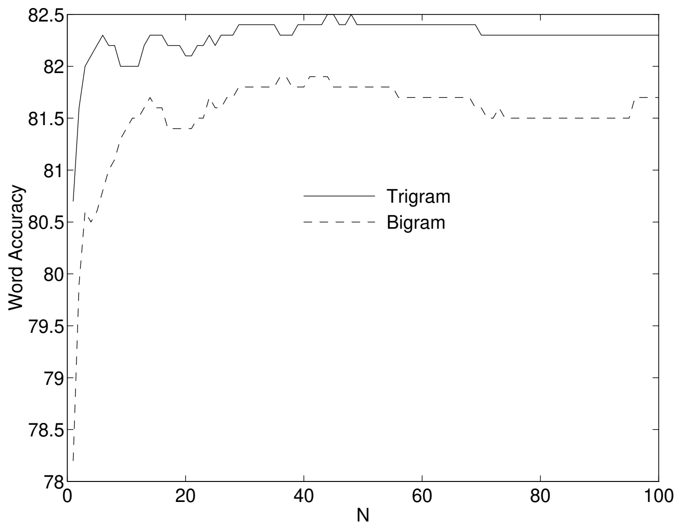

Figure 5-6: Bigram and trigram N-best resorting
In computing the N-best lists we can use either a bigram or a trigram language model. These N-best lists are then resorted with our optimal word class and phrase class models. The disparity between the two is surprising.

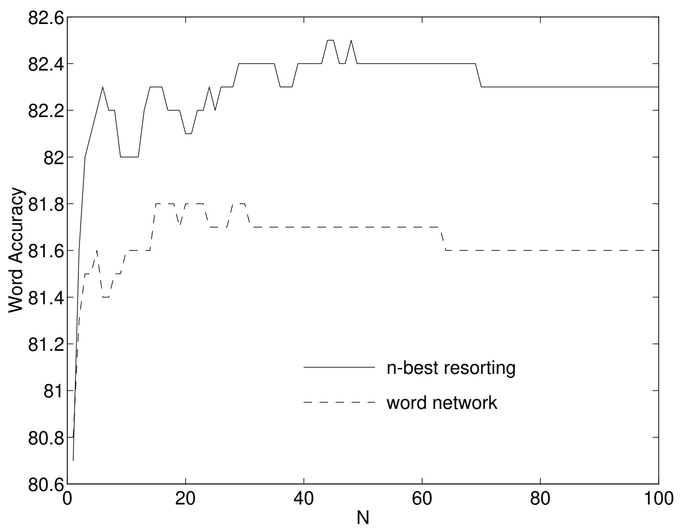

Figure 5-7: N-best resorting and word network integration
This compares the performance of the best phrase class PCNG model on the two means for integration: N-best resorting and word networks. It is very surprising that performance of the word network is so poor compared to the N-best resorting

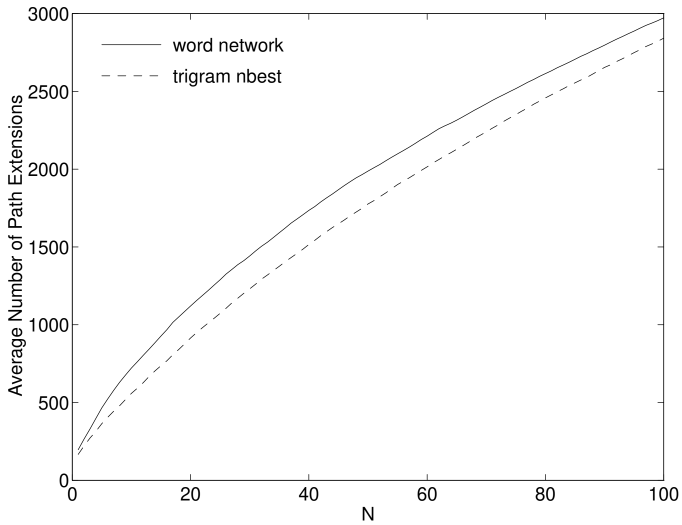

Figure 5-8: A^* path extensions
This plot shows the average number of path extensions done by the A^* search using the N-best resorting approached computed with a trigram *vs.* the word network approach. Computation is slightly higher with the word network.

required to compute the N paths for both the trigram N-best integration and word network integration, using the phrase-class PCNG model. It is surprising that the word network integration requires slightly more computation to compute the first N paths to complete. Both of these plots are falling off below a linear rate as N increases.

To get a direct measure on how inadmissible the A^* search is we can examine the average rank of the best scoring path for each of the three integration methods. This is shown in Figure 5-9 for the phrase-class PCNG model. It is very interesting to see that the rank *continues* to increase as N increases. We would have expected it to eventually taper off, indicating that the top scoring path typically falls somewhere near the top of the N-best list.

5.10 Summary

In this chapter we described a means for integrating the PCNG model with the A^* search for speech recognition. We investigated how to compute an upper bound model

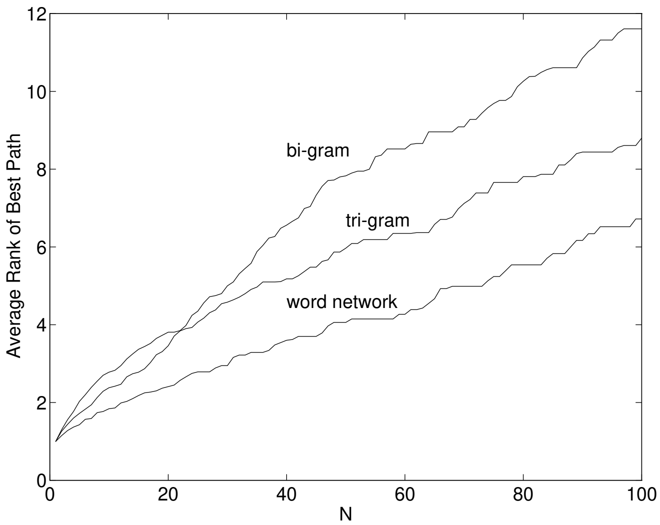

Figure 5-9: Average rank of best scoring path
This plot shows how the average rank varies for the phrase class trigram model when test the three integration schemes

for the bigram search such that the resulting A^* search was admissible, and soon found this model to be very impractical as it drastically increased the computation required by the search. We then realized that the loss of admissibility was not too damaging in practice since we can run the search to produce N answer paths and then search for the best scoring path in this list. This does, however, require substantial computation.

We described three means for integration: full integration, word networks, and N-best resorting. In this chapter we tested the latter two forms of integration, and found some unexpected dependencies among the resulting word accuracies. It was surprising that the word accuracy of N-best resorting was so sensitive to whether a word bigram or trigram model was used to generate the N-best list originally. It was also *very* surprising to see that the trigram N-best integration performed quite a bit better than the word network integration.

We believe that some of these unexpected results are due to some implicit biases present in the word network approach because they were created with the bigram model. In particular, as the new language model deviates more from the bigram model, it seems likely that the computational cost will increases substantially. We were hoping to observe that the more powerful language models allowed the A^* to do fewer extensions to generate the N-best lists, but we believe the implicit bias of the word network towards models which are similar to the word bigram model far outweighs any such effects. We intend to explore these issues more thoroughly in the future.

Chapter 6

Conclusions

6.1 Summary

The learning system described in this thesis produced definite improvements in the performance of language models, in terms of perplexity, and in speech recognition error rate, for the ATIS domain which we were working with. This process is entirely automated, and consists of first acquiring a stream of grammars using the inference process, and then building a probabilistic PCNG model from the acquired grammars to select the minimum perplexity grammar.

We evaluated the inference system by measuring the perplexity of the acquired grammars, and found the perplexity to be substantially reduced for both the bigram *and* trigram PCNG models, which represents a substantial improvement in language model performance. Unfortunately, at these optimal points for each inference run the actual test set coverage of the grammar was quite low. This would seem to imply that the inference system generalizes poorly at the sentence level. But the PCNG model renders the issue of coverage academic, since it is a full coverage model.

The PCNG model represents a new formalism for using grammars to derive a probabilistic model. It is a useful model because it can take a partial grammar and construct a full coverage language model, which reduces *both* the number of parameters and the perplexity when compared to the word n-gram models. This formalism should prove valuable for as an aid for hand-written grammars as well.

The integration of the PCNG model with the A^* search is also a novel contribution. We demonstrated that the automatically acquired PCNG models substantially reduce word error rate, but at the increased cost of computation for the word network coupling explored. This increased computation cost is actually a smooth function of how many rules are present in the grammar and is thus an element under control of the designer of the speech recognition system.

6.2 Future Work

There are a number of directions where this research should be taken. First, the inference algorithm should be tested on more complex domains. The ATIS domain is a relatively simple domain, both in its language structure and in the actual size of the training set and lexicon used. A more complex domain would challenge the inference algorithm computationally, and would require re-engineering some of the components to make them more efficient. It would be interesting to see if this notion of simple phrase classes would be useful in modeling richer languages.

Further efforts should be made to expand the power of the inference algorithm. Presently, the languages it acquires are all finite languages. It would be interesting to extend this system to full CFG's, and to allow it to maintain different possible parses of the training sentences at the same time, as it searches for a good grammar. Such extensions would formally give the inference system more power, which would presumably allow it to better model the actual language.

Another future direction would be to try to augment the inference process with some notion of understanding a sentence. Presently, most structured language models in speech recognition are used as the *understanding* component of the system. But because this inference process pays no attention to the *semantics* of the training sentences, the output is not immediately useful for understanding. The classes that the system learns, however, very often look semantically similar, so there are possibilities in this direction.

Structured language models offer advantages over simple n-gram approaches because they extract additional information from the sentence. For example, one interesting direction to explore would be to see if the structure acquired by this system can be used to more effectively adapt to a particular speaker's usage of the language. This could result in substantial reductions in perplexity and in word error rates.

Further experiments must be done to more thoroughly understand the implications of integrating more powerful language models directly into the search phase of speech recognition. Almost all systems at present choose to make a rough first pass with an approximate language model, and then refine that pass using a more powerful model. Since the PCNG effectively represents a *small step* from the simple word n-gram model to a more powerful model, it should have some interesting implications for computation/accuracy tradeoffs with a full integration scheme.

Appendix A

Perplexity

Perplexity is an information theoretic metric used to evaluate how close the distribution of a given language model $\hat{P}$ is to the "true" distribution of the language. Because we cannot know the true distribution we must approximate it by selecting an independent test set according to the true distribution and measuring an empirical information-theoretic distance between $\hat{P}$ and the distribution seen in the test set. This appendix will relate perplexity to the fundamental entropy of the language, in particular showing the perplexity is lower bounded by 2^H.

The perplexity is computed from the average log probability of words in the test set. The test set will be denoted as $t^N = w_1 w_2 ... w_N$, where $w_1 w_2 ... w_N$ are the words in the sentences in the test set listed end-to-end and separated by a unique sentence boundary marker. This boundary marker is a distinct word, and counts as a word in the perplexity computation. Perplexity is based on the quantity

$\hat{H}$:

$$\hat{H} = -\frac{1}{N} log_2 \hat{P}(t^N) \tag{A.1}$$

$$= -\frac{1}{N} \sum_{i=1}^{N} log_2 \hat{P}(w_i | w_1, ..., w_{i-1}) \tag{A.2}$$

This formulation is fully general in that it makes no assumptions about the nature of $\hat{P}$. The perplexity is then $2^{\hat{H}}$.

To relate $\hat{H}$ to H, a few assumptions must be made about the language source.

First, we assume that the underlying distribution used to create the training and test sets is the same source distribution, and that the training and test sets are independent samplings according to this distribution. The source is assumed to be *ergodic*, meaning that a test set which is reasonably large is in some sense "prototypical" of the language. These assumptions are not unreasonable, and were made anyway to be able to create the language model. They could be relaxed to some extent without changing the results proven in this appendix, but they make the proof more straightforward.

The true entropy H of a source which generates output symbols $w_1, w_2, \ldots$ indefinitely is defined as [14]:

$$H = \lim_{n \to \infty} -\frac{1}{n} \sum_{w^n \in W^n} P(w^n) log_2 P(w^n) \tag{A.3}$$

where the sum is taken over *all* possible sequences of words of length n, and $P(w^n)$ is the true underlying probability of the sequence of words. The quantity inside the limit monotonically decreases as n in the limit increases. H can therefore be approximated by removing the limit and summing across all sequences length N where N is sufficiently large:

$$H = -\frac{1}{N} \sum_{w^N \in W^N} P(w^N) log_2 P(w^N) \tag{A.4}$$

We assume that the size of the test set N is sufficiently large – this assumption is most likely a very safe one as the languages we deal with tend not to have *tremendously* long distance effects (i.e., on the order of N).

Because we've assumed the source is ergodic, if we randomly draw a *particular* sample of size N, according to the underlying distribution the particular sample will be "typical". Therefore, our test sample t^N will be a typical sample, according to the Asymptotic Equipartition Principle for ergodic sources [14]. This means that the average of all the model log probabilities across *all* word sequences of length N will be very close to the log probability of the test set t^N:

$$\hat{H} = -\frac{1}{N} log_2 \hat{P}(t^N) \tag{A.5}$$

$$\approx -\frac{1}{N} \sum_{w^N \in W^N} P(w^N) log_2 \hat{P}(w^N) \tag{A.6}$$

We can now relate the quantities $\hat{H}$ and H. In particular, consider the difference $\hat{H} - H$:

$$\begin{aligned}
\hat{H} - H &\approx -\frac{1}{N} \sum_{w^N \in W^N} P(w^N) log_2 \hat{P}(w^N) + \frac{1}{N} \sum_{w^N \in W^N} P(w^N) log_2 P(w^N) && \text{(A.7)} \\
&= -\frac{1}{N} \sum_{w^N \in W^N} P(w^N)(log_2 \hat{P}(w^N) - log_2 P(w^N)) && \text{(A.8)} \\
&= \frac{1}{N} \sum_{w^N \in W^N} P(w^N)(log_2 P(w^N) - log_2 \hat{P}(w^N)) && \text{(A.9)} \\
&= \frac{1}{N} \sum_{w^N \in W^N} P(w^N) log_2 \frac{P(w^N)}{\hat{P}(w^N)} && \text{(A.10)} \\
&= \frac{1}{N} D(P \parallel \hat{P}) && \text{(A.11)} \\
&\geq 0 && \text{(A.12)}
\end{aligned}$$

where $D(P \parallel \hat{P})$ is the relative entropy [14] between the probability distributions P and $\hat{P}$. Relative entropy is a natural information theoretic distance metric between probability distributions which is always greater than or equal to zero. Equality is achieved if and only iff $P = \hat{P}$. It is also *precisely* the basis for the divergence distance metric used to select merges during inference. Thus, $\hat{H} \geq H$, with equality only when the language model has precisely the underlying distribution. Perplexity is therefore:

$$\approx 2^{\frac{H + D(P \parallel \hat{P})}{N}} \tag{A.13}$$

This means we may interpret perplexity as a distance metric between the language model distribution and the true underlying distribution. As we produce better and better language models, $D(P \parallel \hat{P})$ decreases, and the perplexity becomes closer and closer to the lower bound limit 2^H.

Appendix B

Example Merge Sequence

This section describes an example sequence of merges to illustrate the process a little more clearly and intuitively. Figure B-1 shows the initial grammar and language of this example inference run. In the example, the training set contains only six sentences as shown in the figure, and the initial grammar has one rule for each sentence. Likewise, the initial language is just the training set. Each successive figure shows the resulting grammar and language as a result of that particular merge. Sentences newly added to the language, or rules changed in the grammar, are highlighted in a bold typeface.

The first merge shown in Figure B-2 that is chosen is (pittsburgh, boston) $\rightarrow$ NT_0. The grammar is modified to allow pittsburgh and boston to be interchangeable. Likewise, the language is expanded to accept new sentences which differ from present sentences by only the words boston or pittsburgh. The second merge is (NT_0, philadelphia) $\rightarrow$ NT_1, shown in Figure B-3. This merge selected the non-terminal which resulted from the previous merge to merge with another city name. This merge creates the rule $NT_1 \Rightarrow NT_0$, which makes NT_0 an unnecessary non-terminal. Thus, we can safely replace all occurrence of NT_0 with NT_1, removing it from the grammar. For the third merge, the phrase "please show" is merged with the word "show" (please show, show) $\rightarrow$ NT_2. This merge is done carefully so as to avoid creating recursive rules, as described in Section 3.7.1. Again, two new rules are added to the grammar, and the occurrences of the two units are replaced everywhere with NT_2.

The fourth merge is (i would like, i need) $\rightarrow NT_3$. The resulting grammar and language are shown in Figure B-5. Fifth, we choose to merge (would like, need) $\rightarrow NT_4$). This merge actually created duplicate rules out of the acquired rules added for the previous merge, so one of the duplicates was discarded. This illustrates how the system can learn very large phrases early on and then later divide those phrases up internally. Furthermore, this merge had no effect on the accepted language – all it did was to change the structural description of the phrases "i would like" and "i need". The final merge is (NT_4, want) $\rightarrow NT_5$. This process actually reduced two originally different sentence rules to the same form, so one of them is discarded.

With each merge the language increases as new generalized sentences are added. In this manner the size of the language increases quite quickly (it started with 6 sentences and ended with 23). Also, the sentence rules in each grammar represent the training sentences reduced according to the acquired grammar so far. Any sentences which reduced to the same form during this inference process are discarded, thus simplifying the grammar by decreasing its size.

S $\Rightarrow$ i would like the least expensive flight from boston to denver
S $\Rightarrow$ please list the early flights from pittsburgh to denver
S $\Rightarrow$ please show me the flights that leave around noon
S $\Rightarrow$ i want to go to oakland
S $\Rightarrow$ show me the cheapest fare to philadelphia
S $\Rightarrow$ i need to go to oakland

i would like the least expensive flight from boston to denver

please list the early flights from pittsburgh to denver

please show me the flights that leave around noon

i want to go to oakland

show me the cheapest fare to philadelphia

i need to go to oakland

Figure B-1: Example starting grammar and language

$\mathbf{NT_0}$ $\Rightarrow$ pittsburgh | boston
S $\Rightarrow$ i would like the least expensive flight from $\mathbf{NT_0}$ to denver
S $\Rightarrow$ please list the early flights from $\mathbf{NT_0}$ to denver
S $\Rightarrow$ please show me the flights that leave around noon
S $\Rightarrow$ i want to go to oakland
S $\Rightarrow$ show me the cheapest fare to philadelphia
S $\Rightarrow$ i need to go to oakland

i would like the least expensive flight from boston to denver

i would like the least expensive flight from pittsburgh to denver

please list the early flights from pittsburgh to denver

please list the early flights from boston to denver

please show me the flights that leave around noon

i want to go to oakland

show me the cheapest fare to philadelphia

i need to go to oakland

Figure B-2: After merge (pittsburgh, boston $\rightarrow NT_0$)

$\mathbf{NT_1}$ ⇒ **pittsburgh | boston | philadelphia**
S ⇒ i would like the least expensive flight from $\mathbf{NT_1}$ to denver
S ⇒ please list the early flights from $\mathbf{NT_1}$ to denver
S ⇒ please show me the flights that leave around noon
S ⇒ i want to go to oakland
S ⇒ show me the cheapest fare to $\mathbf{NT_1}$
S ⇒ i need to go to oakland

i would like the least expensive flight from boston to denver
i would like the least expensive flight from pittsburgh to denver
i would like the least expensive flight from philadelphia to denver
please list the early flights from pittsburgh to denver
please list the early flights from boston to denver
please list the early flights from philadelphia to denver
please show me the flights that leave around noon
i want to go to oakland
show me the cheapest fare to philadelphia
show me the cheapest fare to boston
show me the cheapest fare to pittsburgh
i need to go to oakland

Figure B-3: After merge (NT_0, philadelphia $\rightarrow NT_1$)

$NT_1 \Rightarrow$ pittsburgh | boston | philadelphia
$\mathbf{NT_2} \Rightarrow$ please show | show
S $\Rightarrow$ i would like the least expensive flight from NT_1 to denver
S $\Rightarrow$ please list the early flights from NT_1 to denver
S $\Rightarrow$ **$\mathbf{NT_2}$** me the flights that leave around noon
S $\Rightarrow$ i want to go to oakland
S $\Rightarrow$ **$\mathbf{NT_2}$** me the cheapest fare to NT_1
S $\Rightarrow$ i need to go to oakland

i would like the least expensive flight from boston to denver
i would like the least expensive flight from pittsburgh to denver
i would like the least expensive flight from philadelphia to denver
please list the early flights from pittsburgh to denver
please list the early flights from boston to denver
please list the early flights from philadelphia to denver
please show me the flights that leave around noon
show me the flights that leave around noon
i want to go to oakland
show me the cheapest fare to philadelphia
please show me the cheapest fare to philadelphia
show me the cheapest fare to boston
please show me the cheapest fare to boston
show me the cheapest fare to pittsburgh
please show me the cheapest fare to pittsburgh
i need to go to oakland

Figure B-4: After merge (please show, show $\rightarrow NT_2$)

$NT_1 \Rightarrow$ pittsburgh | boston | philadelphia
$NT_2 \Rightarrow$ please show | show
$\mathbf{NT_3} \Rightarrow$ **i would like** | **i need**
S $\Rightarrow$ $\mathbf{NT_3}$ the least expensive flight from NT_1 to denver
S $\Rightarrow$ please list the early flights from NT_1 to denver
S $\Rightarrow$ NT_2 me the flights that leave around noon
S $\Rightarrow$ i want to go to oakland
S $\Rightarrow$ NT_2 me the cheapest fare to NT_1
S $\Rightarrow$ $\mathbf{NT_3}$ to go to oakland

i would like the least expensive flight from boston to denver
i need the least expensive flight from boston to denver
i would like the least expensive flight from pittsburgh to denver
i need the least expensive flight from pittsburgh to denver
i would like the least expensive flight from philadelphia to denver
i need the least expensive flight from philadelphia to denver
please list the early flights from pittsburgh to denver
please list the early flights from boston to denver
please list the early flights from philadelphia to denver
please show me the flights that leave around noon
show me the flights that leave around noon
i want to go to oakland
show me the cheapest fare to philadelphia
please show me the cheapest fare to philadelphia
show me the cheapest fare to boston
please show me the cheapest fare to boston
show me the cheapest fare to pittsburgh
please show me the cheapest fare to pittsburgh
i need to go to oakland
i would like to go to oakland

Figure B-5: After merge (i would like, i need $\rightarrow NT_3$)

$NT_1 \Rightarrow$ pittsburgh | boston | philadelphia
$NT_2 \Rightarrow$ please show | show
$NT_3 \Rightarrow$ i $\mathbf{NT_4}$
$\mathbf{NT_4} \Rightarrow$ **would like | need**
S $\Rightarrow$ NT_3 the least expensive flight from NT_1 to denver
S $\Rightarrow$ please list the early flights from NT_1 to denver
S $\Rightarrow$ NT_2 me the flights that leave around noon
S $\Rightarrow$ i want to go to oakland
S $\Rightarrow$ NT_2 me the cheapest fare to NT_1
S $\Rightarrow$ NT_3 to go to oakland

i would like the least expensive flight from boston to denver
i need the least expensive flight from boston to denver
i would like the least expensive flight from pittsburgh to denver
i need the least expensive flight from pittsburgh to denver
i would like the least expensive flight from philadelphia to denver
i need the least expensive flight from philadelphia to denver
please list the early flights from pittsburgh to denver
please list the early flights from boston to denver
please list the early flights from philadelphia to denver
please show me the flights that leave around noon
show me the flights that leave around noon
i want to go to oakland
show me the cheapest fare to philadelphia
please show me the cheapest fare to philadelphia
show me the cheapest fare to boston
please show me the cheapest fare to boston
show me the cheapest fare to pittsburgh
please show me the cheapest fare to pittsburgh
i need to go to oakland
i would like to go to oakland

Figure B-6: After merge (would like, need $\rightarrow NT_4$)

$NT_1 \Rightarrow$ pittsburgh | boston | philadelphia
$NT_2 \Rightarrow$ please show | show
$NT_3 \Rightarrow$ i $\mathbf{NT_5}$
$\mathbf{NT_5} \Rightarrow$ would like | need | **want**
S $\Rightarrow$ NT_3 the least expensive flight from NT_1 to denver
S $\Rightarrow$ please list the early flights from NT_1 to denver
S $\Rightarrow$ NT_2 me the flights that leave around noon
S $\Rightarrow$ NT_3 to go to oakland
S $\Rightarrow$ NT_2 me the cheapest fare to NT_1

i would like the least expensive flight from boston to denver
i need the least expensive flight from boston to denver
i want the least expensive flight from boston to denver
i would like the least expensive flight from pittsburgh to denver
i need the least expensive flight from pittsburgh to denver
i want the least expensive flight from pittsburgh to denver
i would like the least expensive flight from philadelphia to denver
i need the least expensive flight from philadelphia to denver
i want the least expensive flight from philadelphia to denver
please list the early flights from pittsburgh to denver
please list the early flights from boston to denver
please list the early flights from philadelphia to denver
please show me the flights that leave around noon
show me the flights that leave around noon
i want to go to oakland
show me the cheapest fare to philadelphia
please show me the cheapest fare to philadelphia
show me the cheapest fare to boston
please show me the cheapest fare to boston
show me the cheapest fare to pittsburgh
please show me the cheapest fare to pittsburgh
i need to go to oakland
i would like to go to oakland

Figure B-7: After merge (NT_4, want $\rightarrow NT_5$)

Bibliography

[1] A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling.* Englewood Cliffs, NJ: Prentice-Hall, 1972.

[2] J. K. Baker. "Trainable grammars for speech recognition", *Proc. Conference of the Acoustical Society of America*, 547–550, June 1979.

[3] A. Barr, E. Feigenbaum and P. Cohen. *The Handbook of Artificial Intelligence.* Los Altos, CA: William Kaufman, 1981.

[4] R. Bod. "Monte carlo parsing", *Proc. International Workshop on Parsing Technologies*, 1–12, August 1993.

[5] L. Breiman, J. Friedman, R. Olshen and C. Stone. *Classification and Regression Trees.* Monterey, CA: Wadsworth and Brooks/Cole Advanced Books and Software, 1984.

[6] E. Brill. "A simple rule-based part of speech tagger", *Proc. Conference on Applied Natural Language Processing*, 152–155, 1992.

[7] E. Brill. "A Corpus-Based Approach to Language Learning", Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1993.

[8] E. Brill, D. Magerman, M. Marcus and B. Santorini. "Deducing linguistic structure from the statistics of large corpora", *Proc. DARPA Speech and Natural Language Workshop*, 275–281, June 1990.

[9] E. Brill and M. Marcus. "Automatically acquiring phrase structure using distributional analysis", *Proc. DARPA Speech and Natural Language Workshop*, 155–159, February 1992.

[10] P. Brown, S. Chen, S. DellaPietra, V. DellaPietra, R. Mercer and P. Resnik. "Using decision-trees in language modeling", Unpublished Report, 1991.

[11] P. Brown, V. Della Pietra, P. de Souza, J. Lai and R. Mercer. "Class-based n-gram models of natural language", *Computational Linguistics*, Vol. 18, No. 4, 467–479, December 1992.

[12] S. Chen. "The Automatic Acquisition of a Statistical Language Model for English", Ph.D. Thesis in progress, Harvard University.

[13] K. Church and W. Gale. "A comparison of the enhanced good-turing and deleted estimation methods for estimating probabilities of english bigrams", *Computers, Speech and Language*, Vol 5, No 1, 19–56, 1991.

[14] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* New York: John Wiley and Sons Inc., 1991.

[15] J. Earley. "An efficient context-free parsing algorithm", *Communications of the ACM*, Vol. 13, 94–102, 1970.

[16] K. Fu and T. L. Booth. "Grammatical inference: introduction and survey — part 1", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 5, No. 1, 95–111, January 1975.

[17] K. Fu and T. L. Booth. "Grammatical inference: introduction and survey — part 2", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 5, No. 5, 409–423, July 1975.

[18] J. Glass, D. Goddeau, D. Goodine, L. Hetherington, L. Hirschman, M. Phillips, J. Polifroni, C. Pao, S. Seneff and V. Zue. "The MIT ATIS system: January 1993 progress report", *Proc. DARPA Spoken Language Systems Technology Workshop*, January 1993.

[19] D. Goddeau. "Using probabilistic shift-reduce parsing in speech recognition systems", *Proc. International Conference on Spoken Language Processing*, 321–324, October 1992.

[20] E. M. Gold. "Language identification in the limit", *Information and Control*, Vol. 10, 447–474, 1967.

[21] I. J. Good. "The population frequencies of species and the estimation of population parameters", *Biometrika*, Vol. 40, 237–264, 1953.

[22] I. Hetherington, M. Phillips, J. Glass and V. Zue. "A^* word network search for continuous speech recognition", *Proc. European Conference on Speech Communication and Technology*, 1533–1536, September 1993.

[23] L. Hirschman, *et al.*, "Multi-site data collection for a spoken language system", *Proc. DARPA Speech and Natural Language Workshop*, 7–14, February 1992.

[24] M. Jardino and G. Adda. "Language modelling for CSR of large corpus using automatic classification of words", *Proc. European Conference on Speech Communication and Technology*, 1191–1194, September 1993.

[25] F. Jelinek. "Self-Organized language modeling for speech recognition", *Readings in Speech Recognition*, Alex Waibel and Kai-Fu Lee, eds., 450–506, 1990.

[26] F. Jelinek, J. Lafferty and R. Mercer. "Basic methods of probabilistic context free grammars", *Speech Recognition and Understanding: Recent Advances, Trends, and Applications*, P. Laface and R. De Mori, eds., Vol. 75, 1992.

[27] S. M. Katz. "Estimation of probabilities from sparse data for the language model component of a speech recognizer", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 35, 400–401, 1987.

[28] S. Kullback. *Information Theory and Statistics.* New York: John Wiley and Sons Inc., 1959.

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

[30] J. C. Martin. *Introduction to Languages and the Theory of Computation.* New York: McGraw-Hill Inc., 1991.

[31] E. Newport, H. Gleitman and E. Gleitman. "Mother, I'd rather do it myself: some effects and non-effects of maternal speech style", *Talking to Children: Language Input and Acquisition*, C. E. Snow and C. A. Ferguson, eds., New York: Cambridge University Press, 1977.

[32] H. Ney. "Stochastic grammars and pattern recognition", *Speech Recognition and Understanding: Recent Advances, Trends, and Applications*, P. Laface and R. De Mori, eds., Vol 75, 319–344, 1992.

[33] H. Ney and U. Essen. "On smoothing techniques for bigram-based natural language modeling", *Proc. International Conference on Acoustics, Speech, and Signal Processing*, 825–828, May 1991.

[34] S. Penner. "Parental responses to grammatical and ungrammatical child utterances", *Child Development*, Vol. 58, 376–384, 1987.

[35] F. Pereira and Y. Schabes. "Inside-outside reestimation from partially bracketed corpora", *Proc. DARPA Speech and Natural Language Workshop*, 122–127, 1992.

[36] F. Pereira, N. Tishby and L. Lee. "Distributional clustering of English words", *Proc. 30th Annual Meeting of the Asoociation for Computational Linguistics*, 128-135, 1992.

[37] S. Roucos. "Measuring perplexity of language models used in speech recognizers", *BBN Technical Report*, 1987.

[38] S. Seneff. "TINA: A natural language system for spoken language applications", *Computational Linguistics*, Vol. 18, No. 1, 61–86, 1992.

[39] S. Seneff, H. Meng and V. Zue. "Language modelling for recognition and understanding using layered bigrams", *Proc. International Conference on Spoken Language Processing*, 317–320, October 1992.

[40] C. E. Shannon. “Prediction and entropy of printed English”, *Bell Systems Technical Journal*, Vol. 30, 50–64, January 1951.

[41] S. M. Shieber. “Evidence against the context-freeness of natural language”, *Linguistics and Philosophy*, Vol. 8, 333–343, 1985.

[42] R. Solomonoff. “A Formal Theory of Inductive Inference”, *Information and Control*, Vol. 7, 1–22, 234–254, 1964.

[43] A Stolcke and J. Segal. “Precise n-gram probabilities from stochastic context-free grammars”, *ICSI Technical Report*, January 1994.

[44] L. G. Valiant. “A theory of the learnable”, *Communications of the ACM*, Vol. 27, 1134–1142, 1984.

[45] A. Viterbi. “Error bounds for convolutional codes and an asymptotic optimal decoding algorithm”, *IEEE Transactions on Information Theory*, Vol. 13, 260–269, April 1967.

[46] V. Zue, J. Glass, D. Goodine, M. Phillips and S. Seneff. “The SUMMIT speech recognition system: phonological modelling and lexical access”, *Proc. International Conference on Acoustics, Speech, and Signal Processing*, 49–52, 1990.

[47] V. Zue, J. Glass, M. Phillips and S. Seneff. “Acoustic segmentation and phonetic classification in the SUMMIT speech recognition system”, *Proc. International Conference Acoustics, Speech, and Signal Processing*, 389–392, 1989.