Statistical Source Channel Models for Natural Language Understanding

by

Mark E. Epstein

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 1996

Approved

For Mara

Acknowledgments

There are many people who have helped me finally reach this point. I'd first like to thank Ken Davies. He is the manager that rescued me from a life in development and hired me into the Thomas J. Watson Research Center. Ken also helped me get accepted into IBM's Graduate Work Study program even though he knew that he would lose a good percentage of my time to coursework. I registered for two courses each semester for several years while working for Ken, and he never complained (at least not to me).

After leaving Ken, Eddie Epstein, my second manager while in the Graduate Work Study program, continued to encourage my educational endeavors. Despite a tight schedule, he never asked me to compromise my studies.

Dr. Salim Roukos, my current manager, deserves special thanks. He not only continued to support me through the GWS program, but also gave me ATIS as a research project. For the first time in seven years, I was able to work on a Ph.D. topic at IBM. And while statistical natural language understanding was (and still is) very important to IBM, Salim had enough confidence in me to let me work on it. It is questionable whether or not I would have completed the degree if Salim had not given me this opportunity. But Salim's role was not just managerial, it was professional as well. Salim was always available to answer my questions about the modeling, which often showed my ignorance in the field. He was always patient, and took the time to explain his answers at a level I could understand.

Dr. Todd Ward, a colleague of mine at IBM, has also "been there" for me. I cannot count the number of times that Todd helped me figure out a solution to a problem, either mathematical or programming. Whenever I was not sure about a solution to a problem, Todd was my sounding board. I'm sure that his individual research efforts were slowed by our meetings, but that never stopped him from helping me. Todd also acted as a counselor, providing insight on how to complete a doctorate!

Former IBMer, Dr. Stephen Della Pietra, is without a doubt the brightest mathematician with whom I have ever worked. Like Salim and Todd, he knows statistical modeling at a much greater depth than I do, and he never minded "bringing down" the level of his explanations to one where I could understand and absorb the material. Stephen was my mentor, and without his expert tutelage, I would not have learned the mathematical foundations on which this dissertation is based. His departure from IBM in 1995 did not terminate his assistance, as I feared it might. He was always available by phone, e-mail, and on several occasions, at his house. I am sure he thinks he knows how indebted I am, but he can easily double or triple his best guess. Without his continued training, I would not have been able to complete this dissertation.

Dr. Kishore Papineni deserves recognition for all his hard work on the pattern matcher. This monolithic code had been developed by four different programmers, and was a huge mess. As it became clear that modifications needed to be made to support many features needed for this thesis,

Kishore made the necessary adjustments. Without his insight on running the database tests using differential SQL, I'm sure that my test experiments would still be running!

Lastly, I would like to thank my advisor, Dr. Ralph Grishman. Over the past several years, our advisor/student relationship has been rather unconventional. I became a student of Dr. Grishman's in 1989, when we started working on statistical methods to determine selectional constraints. For many years, this research progressed at a snail's pace. If I met with Dr. Grishman more than 5 times a year, that was considered frequent. In 1993, I abandoned this research topic in favor of statistical natural language understanding when I joined Salim's group. Dr. Grishman had absolutely no problem with me switching to a new research area, and in fact encouraged it. While I have been in "research mode" for the last two years, I continued my track record of visiting him at NYU intermittently. It would have been entirely reasonable for him to request a more "conventional" relationship, in which I would visit him weekly, discuss my research, and co-author papers with him. But he realized that for the type of research that interested me, my colleagues at IBM were better suited to assist me. Thus, I did not have to play the typical "graduate student game", that most Ph.D. candidates have to play. It is his patience, understanding, and tolerance of the uniqueness of my situation for which I am most grateful.

Preface

The problem of *Natural Language Understanding* (NLU) has intrigued researchers since the 1960's. Most researchers working in computational linguistics focus on *linguistic* solutions to their problems. They develop grammars and parsers to process the input natural language into a *meaning representation*. In this thesis, a new approach is utilized. Borrowing from the field of *communication theory*[75], an *information theoretic* approach to natural language understanding is applied. This is based on the *source-channel* model of communication.

The source-channel model of NLU assumes that the user has a *meaning* in the domain of the application that he wishes to convey. This meaning is sent through a *noisy channel*. The observer receives the English sentence as output from the noisy channel. The observer then submits the English sentence to a *decoder*, which searches the meaning space to try to recover the initial meaning sent through the channel. The decoder uses mathematical models of the channel and the meanings to process the English sentence. Thus, the following problems must be addressed in a source-channel model for NLU:

- A mathematical model of the noisy-channel must be developed.
- The parameters of the model must be set, either manually or by an automatic training procedure.

- A decoder must be built to search through the meaning space for the most likely meaning to have generated the observed English.

This dissertation focuses on the first two of these problems. Several mathematical models of the noisy channel are developed. They are trained from a corpus of context independent sentence pairs consisting of both English and the corresponding meaning. The parameters of the models are trained to maximize the likelihood of the model's prediction of the observed training data using the *Expectation-Maximization algorithm*[20]. Results are presented for the Air Travel Information Service (ATIS) domain [69].

Contents

Dedication Page iii

Acknowledgments iv

Preface vii

List of Figures xiv

List of Appendices xv

1 Introduction 1

1.1 Statistical Natural Language Understanding 1

1.2 Statistical Machine Translation 4

1.3 Statement of Thesis 7

1.4 Organization of Thesis 8

2 Statistical Natural Language Understanding 11

2.1 The Source-Channel Paradigm 11

2.2 Related Work 14

2.2.1 Source-Channel Modeling of AT&T 14

2.2.2 BBN - Hidden Understanding Models 17

2.2.3 Decision Trees 20
2.2.4 Grammatical Inference 20
2.2.5 Statistical Understanding With Parsers 21
2.2.6 Neural Nets for Understanding 21
2.3 Maximum-Likelihood Modeling by the Expectation Maximization Algorithm 22
2.4 The Decoder and Language Model 37

3 Air Travel Information Service **39**
3.1 The ATIS Corpus 40
3.2 The ATIS Database 42
3.3 NL-Parse 45

4 Statistical NLU for ATIS **47**
4.1 Analysis of English 47
4.2 Synthesis of Meaning 49
4.3 Hand Alignments 53
4.4 The ATIS Decoder/Pattern Matcher 54
4.5 Evaluation 57
4.5.1 Evaluation Metrics 57
4.5.2 Evaluation Test Sets 58

5 Basic Word Alignment Model **61**
5.1 Introduction 61

5.2 Formulae 64
5.3 Count Derivation 67
5.4 Training 68
5.5 Smoothing 70
5.6 Results 73
5.6.1 Exact Match Maximum Likelihood Results for DEV94 74
5.6.2 Cross Entropy Results for DEV94 78
5.6.3 Viterbi Percentage Results for DEV94 80

6 Basic Clumping Models **83**

6.1 Introduction 83
6.2 Formulae 88
6.3 Count Derivation 92
6.3.1 Computing the Clump Probabilities 93
6.3.2 Using the Baum-Welch Algorithm For Clump Models . 95
6.3.3 Count Update Formulae for Models A, AHW, and ALM Using the Chain Rule 96
6.4 Training 98
6.5 Smoothing 100
6.6 Results 101
6.6.1 Exact Match Maximum Likelihood Results for DEV94 103
6.6.2 Cross Entropy Results for DEV94 106
6.6.3 Viterbi Percentage Results for DEV94 106

7 Clumping Models With Fertilities **108**

7.1 Introduction 108

7.2 Formulae 111

7.3 Count Derivation 115

7.4 Training 118

7.5 Smoothing 120

7.6 Results 120

7.6.1 Exact Match Maximum Likelihood Results for DEV94 121

7.6.2 Cross Entropy Results for DEV94 124

7.6.3 Viterbi Percentage Results for DEV94 125

8 Discussion **126**

8.1 Errors Due to the Translation Model 131

8.1.1 Substitution Errors 132

8.1.2 Insertion Errors 135

8.1.3 Complex Errors 140

8.2 Reducing the Error Rate 143

8.2.1 Solutions For the Spurious Word Problem 145

8.2.2 Reducing Permutation Errors 154

9 A Distortion Model **158**

9.1 Introduction 158

9.2 Formulae 166

9.3 Count Derivation 169

9.4 Training 170
9.5 Smoothing 171
9.6 Results 171
9.6.1 Exact Match Maximum Likelihood Results for DEV94 172
9.6.2 Cross Entropy Results for DEV94 174
9.6.3 Viterbi Percentage Results for DEV94 175

10 Summary **176**

10.1 Summary of Results 178
10.2 How Do the Results Compare to Other ARPA HLT Participants 185
10.3 How Portable are These Results 187
10.3.1 Formal Language 188
10.3.2 Synthesis 189
10.3.3 Training 190
10.3.4 Modeling 191
10.4 Final Summary 192

Bibliography **193**

Appendices **207**

List of Figures

1.1 Analysis-Transfer-Synthesis Paradigm 5

2.1 Source-Channel Model Paradigm 12

2.2 A Concave Function 31

5.1 Example Alignment 63

List of Appendices

A Extended Backus-Naur Form Grammar for ATIS 208

B Yacc Grammar for ATIS 210

C More Decoding Results for Model 1 227

D More Decoding Results for Models A, AHW, and ALM 231

E Errors Made in DEV94 Not Due to the Smoothed Model B Deficiencies 235

Chapter 1

Introduction

1.1 Statistical Natural Language Understanding

The artificial intelligence community has long been interested in natural language understanding. Initial efforts in NLU utilized word spotting techniques (as in ELIZA[80]) or grammars (as in STUDENT[8])[1]. Researchers knew that statistical approaches could be used to solve these problems, but these were considered less glamorous. Consider a quote from Minsky:

> Bobrow's program (i.e. STUDENT) does not have any cautious statistical devices that have to be told something over and over again, so its *learning* is too brilliant to be called so. In fact, there

[1] A brief, but good survey of early NLU systems is given by Bobrow[8].

> is no explicit use of probabilistic notions anywhere in this book: It seems that as we incorporate more and more sophisticated heuristic methods, the need for senseless sources of variation in behavior become less and less necessary.[56].

Unfortunately, early successes in Computational Linguistics have not scaled to more general applications. More complex natural language inputs were not handled well by the initial algorithms. In 1990 ARPA decided to sponsor a spoken language understanding competition to objectively measure the performance of different NLU systems. Different research sites would work on the identical problem, to see the merits of different approaches. Data were collected by each of the sites for the selected domain, the Air Travel Information Service (ATIS) domain[69]. Initial results showed a 50-75% error rate[60], which improved to 30-40% after participants worked on the problem further[61]. Most of these sites had existing NLU systems that were built for other domains. But natural language input created by hundreds of real users revealed the inability of the systems to handle the linguistic phenomena that occur in spontaneous speech.

Since the mid 1980's, more and more researchers in Computational Linguistics have begun to reconsider statistical or hybrid statistical-linguistic systems. There are many reasons for this:

- The inherent difficulty in porting broad coverage grammars to specific sublanguage domains.

- The increase in compute and disk capabilities of micro-computers. Years ago, only the simplest of statistical algorithms could be applied.
- The increase in the availability of large corpora for training statistical models.
- The increase in portability provided by statistical systems.

Defining what it means to "understand" is a very difficult and subjective problem. Many researchers have different opinions. Most AI textbooks address this issue to some degree, and then they jump into a description of how syntax, semantics, and pragmatics are used to perform the understanding. Indeed, Minsky's quote above seems to summarize the opinions of many AI researchers with respect to statistical modeling. Elaine Rich mentions that understanding is the process of converting an input sentence from one representation into another[72]. In particular, a user interacting with a computer expects to receive quick and accurate information in response to a query. The creation of this response, either by another person or a program, demonstrates the responder's "understanding" of the user's query. If a program or another human consistently gives correct answers, this constitutes understanding. If the answers are frequently wrong, the user will claim "this person" or "this system" doesn't understand in frustration.

This definition of understanding is important, for statistical systems frequently do not capture "classical" linguistic knowledge. They do not need to do syntactic or semantic analysis. They do not need the real world knowledge

and deduction capabilities of humans. A program can understand natural language if it is given enough training samples from a *bilingual corpus* consisting of pairs of English sentences and their meanings. The program uses these data to train the parameters of a mathematical model according to the *maximum-likelihood criterion*. These parameters can then be used to *decode* any future English sentence into the most likely meaning that could generate it. As in other statistical natural language applications, for example detecting word collocations, it is likely that the parameters will implicitly capture some syntactic, semantic, and lexical knowledge[17].

While it is true that the mathematical models will have tens, or even hundreds of thousands of parameters, current computer technology has advanced to the point where these models are trainable in only a few hours. In fact, IBM has built a machine translation system, capable of translating French into English using statistical models[13, 7]. This system has millions of parameters, and is vastly more complex than performing NLU in a sublanguage domain. Since NLU for a sublanguage domain has far fewer parameters than machine translation, the understanding models can be trained with a reasonably sized corpus.

1.2 Statistical Machine Translation

One paradigm used for machine translation is that of *analysis-transfer-synthesis*[32]. This is a fancy name for a relatively simple concept, illustrated in figure 1.1.

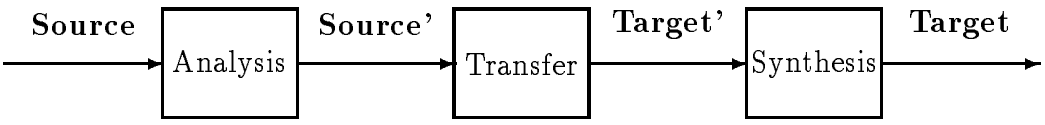

Figure 1.1: Analysis-Transfer-Synthesis Paradigm

The translation from a source language into a target language is performed in three stages:

Analysis This is a pre-processor that converts the input source language into an intermediate representation that is suitable for transferring into a target language representation. An example would be a parser that produces parse trees in some canonicalized deep structure suitable for the transfer.

Transfer This is the process of converting the analyzed representation into a representation of the target language.

Synthesis This is a post-processor that converts the representation of the target language into a natural language sentence in the target language.

Note that this paradigm is very general, and is suited to many applications. Basically the analysis and synthesis steps are transformations from the original and target languages into intermediate forms more suitable for transferring. This paradigm is utilized in this thesis for natural language understanding. Input English is analyzed using a statistical tagger[18, 49]. This tagged English is better suited for statistical transfer than the original English because there are vastly fewer words in the tagged English vocabulary.

Hence fewer parameters are needed in the transfer models. The statistical models transfer the tagged English into a "meaning representation", called the *formal language*, which can be subsequently synthesized into a response for the user. For a database query application, the meaning representation can be a database query language like SQL, or can be a pseudo-language that can be processed into a database query.

In statistical machine translation, a bilingual corpus is used to train the parameters of a statistical model. Millions of sentences that are translations of each other are used to train the parameters of the model. The model makes no attempt to do a syntactic or semantic analysis of the data. Instead, the model learns how to translate by utilizing parameters suitable to map strings in one analyzed language to synthesized strings in the other language. Example parameters include the number of words to generate for a given word, what these words are, and the order these words should be placed in the synthesized sentence[13].

One problem with the analysis-transfer-synthesis paradigm using a deterministic transfer component is that the analysis could introduce an error. For example, the word "may" can be a name, a modal, or a date. If it is mistagged, then an error will certainly result. One could allow ambiguous analyses, but then the transfer and synthesis stages will have to be designed to handle multiple analyses, and to select the most likely result.

1.3 Statement of Thesis

Having introduced natural language understanding and statistical machine translation, I can now state my thesis. The research presented in this thesis, contains a hierarchy of statistical models for performing natural language understanding for ATIS. The goal is to show that statistical models trained using maximum-likelihood estimation can achieve a reasonable level of understanding, with minimal linguistic knowledge. While the results will not "beat" the best systems in the latest ARPA Human Language Technologies workshop[62], it will attain performance near theirs, yet maintain more portability in that no domain specific grammar is required. Also, these systems were built over a five year span, using teams of experienced computational linguists. The ATIS system described in this thesis was built over a two year period. The system uses very general mathematical models, and has minimal domain knowledge. Thus, this system could easily be ported to new applications (provided training data are available). "Beat" is a highly subjective word.

Simple models will be developed, trained, and tested, and the resulting parameters will be used to bootstrap more complex models. The parameters of these models will be trained to maximum-likelihood estimates (MLE), using the *Expectation-Maximization* (EM) algorithm[20].[2] The following fun-

[2] While research has shown that other estimation procedures might be more powerful, for example *Maximum Mutual Information Estimation* (MMIE)[14], they will not be investigated in this thesis.

damental issues involved in maximum-likelihood estimation will be investigated:

- Effect of varying the number of iterations of the EM algorithm
- Effect of providing hand-supervision on a model's performance
- Effect of the amount of training data used for a model

This thesis will focus on context-independent ATIS queries, the *class A* queries[28]. The *class D* and *class X* queries are not treated in this thesis. The models developed here could certainly be applied to these queries, though slight modifications would be required.

In the end, this thesis hopes to convince the reader that statistical natural language understanding is not only viable, but a reasonable alternative to the more common "linguistic" approach. Already statistically trained part of speech taggers have matched or surpassed hand tailored ones in terms of performance[49, 50]. The same is true for statistical parsing[71, 48, 46] and machine translation[7]. This thesis intends to convince the reader that eventually this will be true for natural language understanding as well.

1.4 Organization of Thesis

This thesis is organized as follows:

- Chapter 2 introduces statistical natural language understanding at a very theoretical level. This chapter also describes the work that others

have done in statistical NLU.

- Chapter 3 describes the ATIS domain, the one used in this thesis to demonstrate the algorithms.

- Chapter 4 revisits statistical natural language understanding, but from the pragmatic viewpoint of developing an understanding system for ATIS.

- Chapter 5 presents the first of many models for statistical NLU, which is aptly named model 1. This chapter also discusses many of the issues involved in maximum likelihood estimation, for example the number of iterations of the EM algorithm, and cross entropy as a function of the amount of training data and iteration number. Model 1 generates English words independently from the formal language representation. Since this model is relatively simple, this chapter includes a lengthy discussion of maximum likelihood estimation. The MLE issues are not addressed to nearly the same depth in the other modeling chapters.

- Chapter 6 presents the basic clumping models, which are generalizations of model 1, and are named model A, AHW (A with headwords), and ALM(A with a bigram language model). A clumping model partitions the input English into nonoverlapping substrings or clumps, and requires each clump to be generated from one formal language word.

- Chapter 7 generalizes the basic clumping models by parameterizing the number of clumps that a formal language word generates. The number of clumps for each meaning concept is called its *fertility*. Six models are discussed in this chapter. Models B , BHW, and BLM assume that the fertility of each formal language word can be modeled by a Poisson process. Models C, CHW, and CLM allow general probability distributions for fertilities.

- Chapter 8 discusses the results in great depth, and suggests modeling enhancements that could lead to further improvements in performance.

- Chapter 9 presents the final model of the thesis, which models the proximity of clumps aligned to the same formal language word. This proximity information, called *distortion*, is introduced to solve a class of problems due to two different meanings having the same formal language representation.

- Chapter 10 summarizes the work presented in this thesis. It begins by running the official ARPA evaluations using the models described in the thesis, compares the results to other ATIS solutions, and lists future work that could improve the results.

In addition, there are several appendices, which are introduced when first referenced.

Chapter 2

Statistical Natural Language Understanding

2.1 The Source-Channel Paradigm

In this thesis, the "transfer" stage of the understanding process of figure 1.1 is modeled using the *source-channel model* of Communication Theory, as shown in figure 2.1.

In this model, the speaker conceptualizes a query in the synthesized meaning space. However, he does not express the query in this theoretical space, but instead "sends" it through a *noisy channel*. The noisy channel is composed of many complex processes. The meaning is converted into English words in the speaker's mind, and these words are then output, usually via speech or typed text. The speech or text is then submitted to a decoder,

which searches for the most likely meaning, denoted *meaning'*, to have generated the observed English.

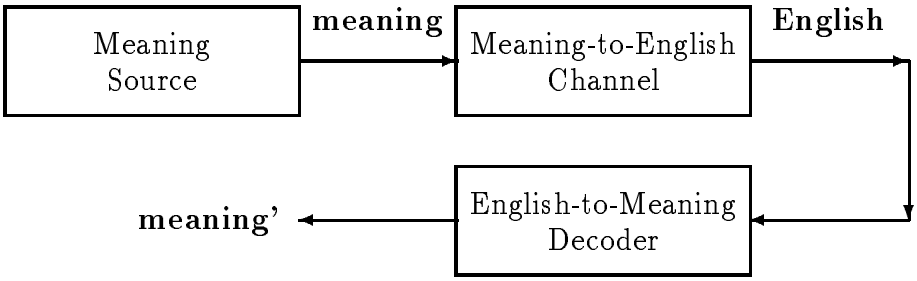

Figure 2.1: Source-Channel Model Paradigm

Modeling how humans communicate is a very difficult if not impossible problem. Once a person decides what they want to communicate, they have to select words, gestures, expressions, tone, stress to convey. It is in the selection and enunciation of words that speakers often stutter, use the wrong words, or say things they did not intend. Internally, they know what they want to convey, but noise is introduced in generating the speech. In communication and information theory, one focuses on fundamental properties of channels (e.g. channel capacity) and results about coding given channels with different properties. In this thesis, the meaning to English channel is the complex psycholinguistic channel in which our thoughts are converted into spoken words.

In this thesis, no attempt is made to study the channel with the same mathematical vigor as done in communication theory. Instead, the channel is modeled using statistical models whose parameters are set to maximize the likelihood of the model to predict observed training data. In the source-

channel paradigm, the decoder must search for the meaning with the most probable meaning, namely:

$$p(Meaning' \mid English) \geq p(Meaning_j \mid English) \text{ for all j} \tag{2.1}$$

This is called the *ideal-observer* or *minimum-error* rule [81]. Using Bayes rule, the conditioning can be inverted:

$$p(Meaning \mid English) = \frac{p(English \mid Meaning)p(Meaning)}{p(English)} \tag{2.2}$$

Maximizing $p(Meaning \mid English)$ for an observed English sentence is identical to maximizing the numerator of equation 2.2, since the denominator is a constant for all conjectured meanings. This is well known in statistical speech recognition systems [2].

In order to use the source-channel model for natural language understanding, one must provide realizations of three components of equation 2.2:

Translation Model This models $p(English \mid Formal)$.[1]

Language Model This models $p(Formal)$

Decoder This is the program that searches the meaning space, scoring candidate meanings using equation 2.2.

In the remaining sections of this chapter, the introduction to statistical

[1]Remember, "Formal" will be used to denote meaning.

natural language understanding is concluded. Section 2.2 describes statistical NLU done by other researchers in the field. In section 2.3, the Expectation-Maximization algorithm[20] is described, which is the algorithm used in this thesis to perform maximum-likelihood estimation of the model parameters. Section 2.4 briefly discusses the components of the source-channel model that are not investigated in this thesis.

2.2 Related Work

In the last ten years, statistical methods have been used for more and more problems in computational linguistics. These have ranged from simple taggers[18] to complex parsers[71, 48, 46]. In addition, there has also been published work in natural language understanding. Two research efforts, at AT&T and BBN, have used the source-channel model, as done in this thesis. There are also a few NLU projects using different modeling paradigms. In the following subsections, some of these related areas are briefly surveyed.

2.2.1 Source-Channel Modeling of AT&T

AT&T has a statistical NLU program named CHRONUS (Conceptual Hidden Representation of Natural Language Unconstrained Speech)[65, 63, 66, 64, 45]. The CHRONUS source-channel model uses *semantic concepts* for its meaning space. The acoustic signal of speech is the output of the noisy channel. The decoder must therefore find the maximum word string W and

concept string C, given acoustic string A according to:

$$p(\tilde{W}, \tilde{C} \mid A) = \max_{W \times C} p(W, C \mid A) \tag{2.3}$$

Using Bayes rule, CHRONUS searches the concept space in order to maximize:

$$p(W, C \mid A) = \frac{p(A \mid W, C)p(W, C)}{p(A)} \tag{2.4}$$

Since A is the same for all W and C, the W and C which maximize equation 2.4 also maximizes:

$$p(A \mid W, C)p(W \mid C)p(C) \tag{2.5}$$

The $p(A \mid W, C)$ term is only needed for the speech recognition, and is represented by a *hidden Markov model* (HMM). The $p(W \mid C)$ term is called the language or syntactic model. The $p(C)$ term is called the conceptual or semantic model. Both the language and conceptual models can be formulated using conditional probabilities. Assuming that there are M words in the sentence, $W_1, \cdots W_M$, each with concept $C_1, \cdots C_M$, then:

$$p(W \mid C)p(C) = \begin{array}{l} \prod_{i=2}^{M} p(w_i \mid w_{i-1} \cdots w_1, C)p(w_1 \mid C) * \\ \prod_{i=2}^{M} p(c_i \mid c_{i-1} \cdots c_1)p(c_1) \end{array} \tag{2.6}$$

For practical reasons, order 1 or 2 Markov processes are used to approximate the products. For example, the conditional probabilities $p(w_i \mid$

$w_{i-1} \cdots w_1, C)$ can be approximated by $p(w_i \mid w_{i-1} w_{i-2}, c_i)$.

The AT&T researchers discovered that training the Markov process parameters using the Baum-Welch algorithm[6] did not work for these models. They found it necessary to manually tag each word with a concept, and then accumulate the frequency counts of each parameter in the tagged data. This tagging was called "segmenting", as one can identify substrings of words (e.g. segments), and then give the segment a class tag. They did come up with a clever method to facilitate bootstrapping. They initially segmented 547 sentences manually, and attained estimates for the model parameters. They then used their models to decode each sentence in the corpus, and compared the database result with the reference result included with the training corpus (see chapter 3 for more on the types of data included in the ATIS corpus). If the result matched, they assume that the model's segmentation was correct, and used it as training data during the next iteration.

The CHRONUS system also uses analyzed English. Three aspects of their analysis are published:

- They use only word stems. For example, "flight", "flights", "flying", and "fly" are treated the same.

- Ambiguous tokens are "passed through to the decoder", creating a lattice of possible tokenizations. They decode all of these in parallel using a Viterbi decoder, and pick the one that scores the best.

- To make order-1 Markov models more effective, they reduce two words

to one when possible. For example, determiners are subsumed by the word that follows them and "one way" becomes "one-way".

2.2.2 BBN - Hidden Understanding Models

The BBN work, named *Hidden Understanding Models* (HUM)[54, 53, 52], also uses the source-channel paradigm for NLU. The source sends a meaning M through a noisy channel, and the observer receives the English sentence W. The observer then processes the English with a decoder, which uses Bayes rule to maximize:

$$p(M \mid W) = \frac{p(W \mid M)p(M)}{p(W)} \tag{2.7}$$

Again, $p(W)$ is a constant, so one needs to maximize only the numerator at decoding time.

The modeling of $p(W \mid M)$ and $p(M)$ are performed by hidden Markov models, as in CHRONUS. Semantic concepts (states in the HMM) are connected to each other by arcs. The model is hidden in that the decoder must find the most likely path through the HMM. Unlike CHRONUS, only some of the concepts are designated as "terminal" concepts, which generate words in W. The transition probabilities from one concept to another form the basis for $p(M)$, which they call the *semantic language model*. The generation of words at the terminal concepts is the basis for $p(W \mid M)$, which they call the *lexical realization model*. Nonterminal concepts are included as place

markers, so that a parse tree can be built automatically from a path through the HMM. In fact, the HMM consists of a fully connected subgraph for each semantic concept. Each subgraph contains a unique entry state, that has arcs into each node in the subgraph. Similarly, each subgraph contains a unique exit state, that can be reached from any node in the subgraph.

The probability of generating a path is then:

$$p(Path) = \prod_{t \in Path} \begin{bmatrix} p(state_n \mid state_{n-1}, context) & \text{if in semantic language model} \\ p(word_n \mid word_{n-1}, context) & \text{if in lexical realization model} \end{bmatrix} \tag{2.8}$$

In equation 2.8, *context* serves as a reminder that the probabilities are conditioned according to the subgraph or state the system is in. There are many subgraphs that contain an "origin" and a "destination". The arc probability from the "origin" state to the "destination" state depends on the subgraph in which these states appear. If these states are in the "flight" subgraph or the "fare" subgraph, the arc probability can have a different value. These probabilities are part of the semantic language model. In the lexical realization model, the probability of generating words depends on the state and the previously generated words for this state.

In fact, $p(Path)$ can be factored as follows:

$$p(Path) = \prod_{state} p(state_n \mid state_{n-1}, context) * \prod_{words} p(word_n \mid word_{n-1}, context) \tag{2.9}$$

This factoring reveals that the semantic language model and lexical realiza-

tion models are bigram models (i.e. order-1 hidden Markov models). While higher order HMMs could be used, due to limited training data, bigrams are utilized. As in AT&T's CHRONUS system, BBN uses hand segmented data to train the model parameters. The parameters are smoothed using techniques found in [38].

The BBN and AT&T work are very similar. The major difference is that BBN produces a conceptual tree as output. Having just terminal nodes gives the leaves of the tree, but does not indicate how they are connected. Their HMM has nonterminal states to encode non-terminal nodes in the parse tree. They do not generate any words, but they are predicted by the semantic language model. Thus, BBN can output a conceptual tree instead of simply a segmentation. The fundamental modeling for each system is a hidden Markov model.

There are two main differences between the AT&T and BBN work from the work in this thesis. The AT&T and BBN statistical systems have models which require hand aligned training data. The models presented in this thesis can be trained with no hand aligned/segmented training data. One of the utility of statistical systems is their ability to be "automatically trained" if given enough training data. In order to port their systems to a new domain, one would have to first hand align the training data. This thesis assumes that this alignment is not available, and presents a hierarchy of models which can be trained without this information. The second difference is that the AT&T and initial BBN systems require grammars or rule based systems to

map from the words that align to a particular semantic concept to the actual SQL query. The meaning space used in this thesis was indirectly derived from SQL. It maintains enough of the SQL so that this mapping is unambiguous. Hence, generating the database query does not require a domain expert to provide rules. A recent BBN paper reveals that they now do this statistically using decision trees[55].

2.2.3 Decision Trees

Researchers at CRIM have built an understanding system for ATIS using decision trees[41, 42]. They built 106 decision trees, each responsible for the inclusion or exclusion of a particular clause in SQL (their meaning space). For example, to decide whether or not to display the fare_id, there is a decision tree (refer to chapter 3 for a discussion of ATIS). The questions asked at each node are of the form "does the English sentence match the following regular expression".

2.2.4 Grammatical Inference

There has been lots of research over the years on building grammars and transducers. Most of the algorithms developed are deterministic, and use a corpus only to incrementally modify a set of rules. While they are corpus driven, they are not statistical approaches. For a good survey of Grammatical Inference, consult the survey by Fu[22].

In a recent paper[15], a CFG is induced, where rules are added not to cover the observed training data, but to instead maximize the likelihood of the grammar to produce the data. That is, rules are examined, and the grammar "trained" using the Inside-Outside algorithm[3]. No search procedure is utilized; the rules are found in a deterministic fashion, and their inclusion is predicated on the gain in the objective function.

2.2.5 Statistical Understanding With Parsers

The "standard" linguistic approach to natural language understanding, involves parsing an input sentence syntactically and semantically, to produce a parse tree, using a grammar designed for the domain in question. Recently, parsers have been developed that use statistical models to produce parse trees without the aid of a phrase structure grammar[10, 47, 71, 34, 48, 46]. The models are then applied in either a top down or bottom up fashion, to build a parse tree directly. Semantic actions can then be implemented in a second pass over the parse tree, and are deterministic.

2.2.6 Neural Nets for Understanding

Neural networks are another mathematical way to model understanding. One such effort by a team at Bell Labs used neural networks for NLU in a call routing task[25, 23, 24]. The authors used mutual information to establish the weights between an English word and semantic concepts. What is most

novel is that the entire network is built from training data. As new words are seen, a new input node is added to the neural network input layer and then the model is retrained.

2.3 Maximum-Likelihood Modeling by the Expectation Maximization Algorithm

In this thesis, the parameters of a model are trained using *maximum likelihood estimation* (MLE). What this means, is that the parameters are adjusted so that they predict the observed training data with the greatest probability possible. Other parameter estimation techniques can be used, like maximum mutual information estimation[14], but these are not investigated in this thesis. In this section, the *Expectation-Maximization* (EM) algorithm[20] is described, which is a way of performing maximum likelihood estimation.

To introduce the concept behind MLE, consider a die tossing experiment. Suppose there is an unfair die with sides $A, B, \cdots F$, which respectively have the probability $p(A), p(B), \cdots p(F)$ of being tossed. The goal is to determine $p(A), p(B), \cdots p(F)$ experimentally. We all know the answer is to toss the die numerous times, accumulate the frequency counts $c(A), c(B), \cdots c(F)$ associated with each event, and then normalize:

$$\tilde{p}(A) = \frac{c(A)}{c(A) + c(B) + \cdots + c(F)} \tag{2.10}$$

The computed probability $\tilde{p}(A)$ is written with a "tilde" to indicate that it is not the actual probability, but rather an empirically measured quantity.

But why is this the maximum likelihood estimate? The likelihood of the observed sequence of C tosses is given by:

$$\mathcal{L} = p(A)^{c(A)}p(B)^{c(B)}p(C)^{c(C)}p(D)^{c(D)}p(E)^{c(E)}p(F)^{c(F)} \tag{2.11}$$

where $C = c(A)+\cdots+c(F)$. The goal is to assign values for $p(A), p(B), \cdots p(F)$ such that $\mathcal{L}$ is maximized for the observed sequence, subject to the linear constraint that $p(A) + p(B) + \cdots + p(F) = 1$. Maximizing $\mathcal{L}$ is identical to maximizing:

$$\frac{1}{C}\log \mathcal{L} \tag{2.12}$$

since C is a constant and log is monotone increasing. Thus, one has to maximize:

$$\begin{aligned} \frac{1}{C}\log \mathcal{L} &= \frac{1}{C}\log(p(A)^{c(A)}p(B)^{c(B)}\cdots p(F)^{c(F)}) & (2.13)\\ &= \tilde{p}(A)\log(p(A)) + \cdots + \tilde{p}(F)\log(p(F)) & (2.14) \end{aligned}$$

Since there is a linear constraint, one can use the method of Lagrangian multipliers to solve for $p(A)$:

$$\frac{\partial}{\partial p(A)}(\tilde{p}(A)\log(p(A)) + \cdots + \tilde{p}(F)\log(p(F)) - \lambda((p(A) + \cdots + p(F)) - 1)) = 0 \tag{2.15}$$

The partial derivative leads to:

$$\frac{\tilde{p}(A)}{p(A)} - \lambda = 0 \tag{2.16}$$

$$\frac{\tilde{p}(A)}{p(A)} = \lambda \tag{2.17}$$

$$p(A) = \frac{\tilde{p}(A)}{\lambda} \tag{2.18}$$

All that remains is to solve for λ and substitute into equation 2.18. Applying this derivation for each of the six unknown variables $p(A), p(B), \cdots p(F)$, one will get six equations in seven unknowns. Fortunately, the linear constraint gives a seventh equation:

$$\frac{\tilde{p}(A)}{\lambda} + \frac{\tilde{p}(B)}{\lambda} + \cdots + \frac{\tilde{p}(F)}{\lambda} = 1 \tag{2.19}$$

This means:

$$\lambda = \tilde{p}(A) + \tilde{p}(B) + \cdots + \tilde{p}(F) \tag{2.20}$$

$$= 1 \tag{2.21}$$

Thus, one discovers that the MLE value for $p(A)$ is indeed $\tilde{p}(A)$.

This die example can actually be generalized into a framework suitable for natural language understanding. Suppose instead of one die, you had 294 different types of dice, with each type being unfair in an identical way. Instead of being 6-sided suppose each die had 640 sides. Now imagine that

you are given the outcomes of a set of experiments. Each experiment can use any number of dice, usually between 5-10. Each different type can be used more than once in an experiment if desired. In each experiment, a die is tossed 0 or more times, and on average, there are 10 tosses per experiment. You are given the dice that were used, the number of tosses that occurred, and the outcome of the tosses, but neither the number of times each die was used nor the outcome of tossing any particular die. The likelihood formula for this experiment is much more complex than in the single die case. Denote the observed outcome of an experiment by a tuple E consisting of words $e_1, \cdots e_{\ell(E)}$, where each e_i is one of the 640 outcomes. Denote the tuple of dice used in the experiment by F, where each die f_i is one of the 294 types of dice.[2] Lastly, assume that A is a tuple of which die was used for each toss (e.g. $a_i \in \{1, 2, \cdots \ell(F)\}$). If one knew which dies was used for each toss, then it would be trivial to calculate the likelihood. Remember though, the data for an experiment consists only of E and F. While A is not known, the likelihood $\mathcal{L}$ for a single experiment can still be expressed in terms of A, F, and $\ell(E)$. Thus:

$$p(E \mid F) = \sum_A p(E, A \mid F) \tag{2.22}$$

$$p(E, A \mid F) = p(E \mid A, F, \ell(E))p(A \mid F, \ell(E))p(\ell(E) \mid F) \tag{2.23}$$

[2]The choice of F to denote "die" will become evident later.

If one assumes that all alignments $p(A \mid F, \ell(E))$ are equally likely, then:

$$p(E, A \mid F) = \frac{p(\ell(E) \mid F)}{\ell(F)^{\ell(E)}} \prod_{i=1}^{\ell(E)} p(e_i \mid f_{a_i}) \tag{2.24}$$

Since A is hidden, it is not possible to accumulate the frequency counts directly, since one does not know which A was used in the generation of E from F. Intuitively, one could imagine that each A were possible, and use *fractional* frequency counts for $p(e_i \mid f_{a_i})$ calculated by:

$$c(E, A' \mid F) = \frac{p(E, A' \mid F)}{\sum_A p(E, A \mid F)} \tag{2.25}$$

$$= \frac{p(E, A' \mid F)}{p(E \mid F)} \tag{2.26}$$

Each parameter used in $p(E, A \mid F)$, for example $p(e_i \mid f_{a_i})$, has a count value. The count value for each of these parameters would be incremented by $c(E, A' \mid F)$. Of course, this requires initial values for the parameters $p(e_i \mid f_j)$ in order to compute the fractional count in equation 2.26. Upon accumulating the fractional counts for each parameter across the entire set of experiments, the parameters are finally re-estimated by normalizing the accumulated counts into probability distributions:

$$p(e_i \mid f) = \frac{c(e_i \mid f)}{c(f)} \tag{2.27}$$

This process can then be iterated using the new parameter estimates. This it-

erative procedure is called the Expectation-Maximization algorithm[20], and is used to perform MLE when the likelihood is formulated using hidden states. The EM algorithm is used as follows:

1. Pick a set of initial statistics for the parameters. Denote these as Θ_0. Set the iteration number, it to 0.

2. Iterate until the stopping criterion is met:

   (a) For each parameter, set its count to 0.

   (b) Iterate over each pair of training sentences (E, F):

      i. Calculate the *expected* number of times each parameter is used in a pair of training sentences. This is done by computing a fractional count for each of the hidden A as given by equation 2.26.

      ii. Increment the count for each parameter used in (E, F) by the fractional count.

   (c) Increment the iteration number it.

   (d) *Maximize* the likelihood of the training set by re-estimating each parameter of Θ_{it} using equation 2.27.

In the previous example, the choice of the number of dice, the number of sides, and even the variables used to denote these were done to facilitate the link between the discussion of the EM algorithm and statistical NLU. In statistical NLU, one is given a training corpus containing pairs of English

sentences and their formal language. The English words are denoted E and the formal language F. Each $e \in E$ is generated by an $f \in F$, but this information is hidden. That is, there is a hidden alignment A between words in E and words in F. As it happens, the ATIS corpus (see chapter 3) requires vocabularies of 640 English words and 294 Formal words. One can use "die tossing" as the basis for a model, and this leads to the first statistical model presented in this thesis (see chapter 5).

To formally show that the EM algorithm performs MLE, it is necessary to show that for each iteration, the likelihood has not decreased. Since the likelihood is bounded above by 1.0, this implies convergence of the EM algorithm.[3] Let E denote some observed event and let F denote the source. Let Θ be a vector containing all the parameters of a model that are assumed to be probabilities. Thus, each Θ_i is between 0 and 1, and $\sum_i \Theta_i = 1$. Denote the probability of generating E from F according to a model with parameters Θ by $p_\Theta(E \mid F)$. The log likelihood for a training set of a model with hidden states is:

$$\log \mathcal{L}(\Theta) = \sum_{(E,F)} \tilde{c}(E,F) \log p_\Theta(E \mid F) \tag{2.28}$$

$$= \sum_{(E,F)} \tilde{c}(E,F) \log \sum_A p_\Theta(E,A \mid F) \tag{2.29}$$

where $\tilde{c}(E,F)$ is the number of times the pair (E,F) of aligned sentences

[3]The likelihood space can be quite complex, and the EM algorithm might converge to a local maximum or a saddle point of the likelihood space.[20]

appears in the training corpus of n pairs of sentences, and A represents each hidden way of generating E from F.

Now let's define the *objective function* as:

$$\mathcal{O}(\Theta) = \frac{1}{n} \log \mathcal{L}(\Theta) \tag{2.30}$$

$$= \sum_{(E,F)} \tilde{p}(E,F) \log p_\Theta(E \mid F) \tag{2.31}$$

If one shows that $\mathcal{O}(\Theta)$ is monotone increasing, then $\mathcal{L}(\Theta)$ is also monotone increasing. It is interesting to note the similarity between $\mathcal{O}(\Theta)$ and the *cross-entropy* prediction the model gives for the training data. These are related by a constant -1 factor. Thus, maximizing $\mathcal{O}(\Theta)$ is equivalent to minimizing the model's entropy on the training data.

The proof that $\mathcal{O}(\Theta)$ is monotone increasing requires comparing $\mathcal{O}(\Theta)$ for two different sets of parameters, the initial set used to accumulate the parameter counts, and the subsequent set computed by normalizing these counts. The initial parameter vector will be denoted by Θ, while the new parameter vector will be denoted by Θ'. In order to show that $\mathcal{O}(\Theta') \geq \mathcal{O}(\Theta)$, first define the *relative objective function* as:

$$R(\Theta', \Theta) = \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \log \frac{p_{\Theta'}(A, E \mid F)}{p_\Theta(A, E \mid F)} \tag{2.32}$$

This is very nearly $\mathcal{O}(\Theta') - \mathcal{O}(\Theta)$, except the summation over all A is now done outside the logarithm, and the log probabilities are weighted by $p_\Theta(A \mid$

$E, F)$. Intuitively, $R(\Theta', \Theta)$ uses the parameter values in vector Θ to weigh the log likelihood of a particular alignment proportionally by $p_\Theta(A \mid E, F)$.

To prove that $\mathcal{L}(\Theta)$ is monotone increasing, requires two theorems.

Theorem 1 $\Theta' = \Theta \Rightarrow R(\Theta', \Theta) = 0$

Proof:

$$\begin{aligned} R(\Theta, \Theta) &= \sum_{(E,F)} \tilde{p}(E, F) \sum_{A} p_\Theta(A \mid E, F) \log \frac{p_\Theta(A, E \mid F)}{p_\Theta(A, E \mid F)} && (2.33) \\ &= \sum_{(E,F)} \tilde{p}(E, F) \sum_{A} p_\Theta(A \mid E, F) \log 1 && (2.34) \\ &= 0 && (2.35) \end{aligned}$$

Thus, after performing an iteration of the EM algorithm, should the same parameter set result, then the relative objective function has value 0.

Theorem 2 $\mathcal{O}(\Theta') \geq \mathcal{O}(\Theta) + R(\Theta', \Theta)$

Proof: The key to proving this is the concavity of the logarithm function, which is known as *Jensen's Inequality*. For concave functions f:

$$f(\sum_i p_i x_i) \geq \sum_i p_i f(x_i) \tag{2.36}$$

if $p_i \geq 0$ and $\sum_i p_i = 1$. This is illustrated pictorially in figure 2.2 for two p_i. Basically, any point along a line between two points x_1 and x_2 lies at or below the value of the function at that point.

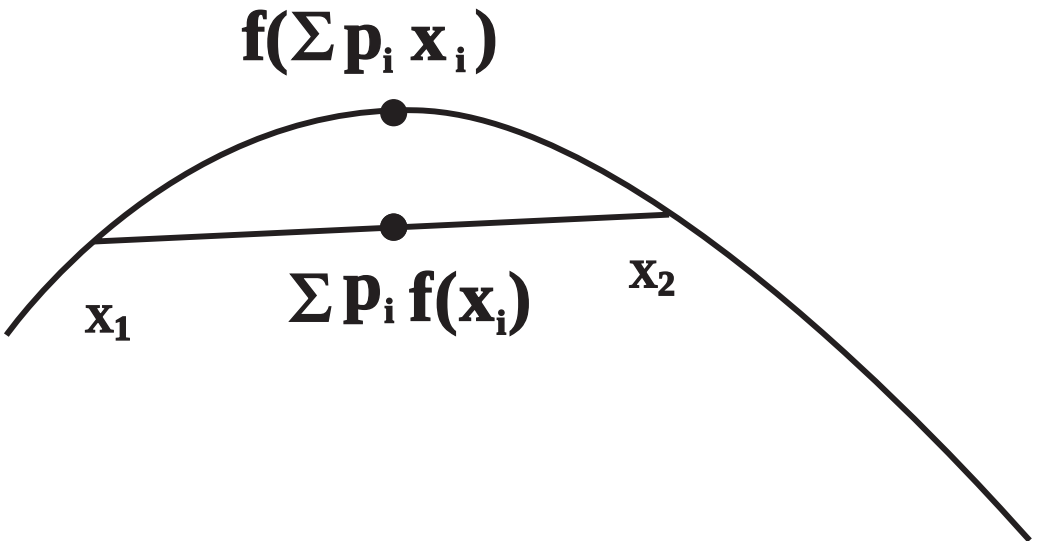

Figure 2.2: A Concave Function

Thus, since log is concave:

$$\log(\sum_i p_i x_i) \geq \sum_i p_i \log x_i \tag{2.37}$$

Therefore:

$$R(\Theta', \Theta) = \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \log \frac{p_{\Theta'}(A,E \mid F)}{p_\Theta(A,E \mid F)} \tag{2.38}$$

$$\leq \sum_{(E,F)} \tilde{p}(E,F) \log \sum_A p_\Theta(A \mid E,F) \frac{p_{\Theta'}(A,E \mid F)}{p_\Theta(A,E \mid F)} \tag{2.39}$$

Borrowing two relations from probability theory, namely:

$$p(A \mid B,C) = \frac{p(A,B \mid C)}{p(B \mid C)} \tag{2.40}$$

$$\sum_{A} p(A, B \mid C) \;=\; p(B \mid C) \tag{2.41}$$

equation 2.39 can be simplified further:

$$\begin{aligned} R(\Theta', \Theta) &\leq \sum_{(E,F)} \tilde{p}(E, F) \log \sum_{A} \frac{p_{\Theta'}(A, E \mid F)}{p_{\Theta}(E \mid F)} && (2.42) \\ &= \sum_{(E,F)} \tilde{p}(E, F) \log \frac{p_{\Theta'}(E \mid F)}{p_{\Theta}(E \mid F)} && (2.43) \\ &= \begin{array}{l} \sum_{(E,F)} \tilde{p}(E, F) \log p_{\Theta'}(E \mid F) - \\ \sum_{(E,F)} \tilde{p}(E, F) \log p_{\Theta}(E \mid F) \end{array} && (2.44) \\ &= \mathcal{O}(\Theta') - \mathcal{O}(\Theta) && (2.45) \end{aligned}$$

Thus, the objective function is monotone increasing if the relative objective function is maximized on each iteration. This is because $R(\Theta', \Theta)$ attains a value of 0 if $\Theta' = \Theta$. Thus, the maximum over all Θ' has to be at least 0. Fortunately, it is easy to maximize $R(\Theta', \Theta)$ under the weak constraint that $p_{\Theta}(A, E \mid F)$ involves just products of the parameters. That is:

$$p_{\Theta}(A, E \mid F) = \prod_{i \in A} \Theta_i^{C_{i,E,A,F}} \tag{2.46}$$

Equation 2.46 shows that the product is taken of each parameter in the hidden derivation and each parameter is used $C_{i,E,A,F}$ times (the number of times that parameter i is used in hidden derivation A between E and F).

Before showing how to maximize the relative objective function, it is useful to see how equation 2.46 can be solved for C_{Θ_i}, the count parameter Θ_i

receives when using a hidden model with parameters Θ over a training corpus. $C_{i,E,A,F}$ can be found by taking the logarithm and the partial derivative with respect to Θ_i of each side of equation 2.46. This then gives:

$$C_{i,E,A,F} = \Theta_i \frac{\partial}{\partial \Theta_i} \log p_\Theta(A, E \mid F) \tag{2.47}$$

Similar to conditional probabilities:

$$\begin{aligned} C_{\Theta_i,E,F} &= \sum_A p_\Theta(A \mid E, F) C_{i,E,A,F} && (2.48) \\ C_{\Theta_i} &= \sum_{(E,F)} \tilde{p}(E, F) C_{\Theta_i,E,F} && (2.49) \end{aligned}$$

Note that if one sums over all A in equation 2.48, $C_{\Theta_i,E,F}$ can be expressed in terms of $p_\Theta(E \mid F)$:

$$\begin{aligned} C_{\Theta_i,E,F} &= \sum_A p_\Theta(A \mid E, F) C_{i,E,A,F} && (2.50) \\ &= \sum_A p_\Theta(A \mid E, F) \Theta_i \frac{\partial}{\partial \Theta_i} \log p_\Theta(A, E \mid F) && (2.51) \\ &= \sum_A \frac{p_\Theta(A \mid E, F)}{p_\Theta(A, E \mid F)} \Theta_i \frac{\partial}{\partial \Theta_i} p_\Theta(A, E \mid F) && (2.52) \\ &= \sum_A \frac{p_\Theta(A \mid E, F)}{p_\Theta(A, E \mid F)} \frac{p_\Theta(E \mid F)}{p_\Theta(E \mid F)} \Theta_i \frac{\partial}{\partial \Theta_i} p_\Theta(A, E \mid F) && (2.53) \\ &= \sum_A \frac{1}{p_\Theta(E \mid F)} \Theta_i \frac{\partial}{\partial \Theta_i} p_\Theta(A, E \mid F) && (2.54) \\ &= \frac{1}{p_\Theta(E \mid F)} \Theta_i \frac{\partial}{\partial \Theta_i} \sum_A p_\Theta(A, E \mid F) && (2.55) \\ &= \frac{1}{p_\Theta(E \mid F)} \Theta_i \frac{\partial}{\partial \Theta_i} p_\Theta(E \mid F) && (2.56) \end{aligned}$$

$$= \Theta_i \frac{\partial}{\partial \Theta_i} \log p_\Theta(E \mid F) \tag{2.57}$$

Using the method of Lagrangian multipliers, $R(\Theta', \Theta)$ can be maximized by taking partial derivatives with respect to Θ'_i and setting to 0. Without loss of generality, assume $\Theta_i \geq 0$ and $\sum_i \Theta_i = 1$. Allowing partitioned subsets of Θ to sum to 1 just adds extra Lagrangian multipliers, but the derivation remains fundamentally the same. By applying the method of Lagrangian multipliers:

$$\frac{\partial}{\partial \Theta'_i}[R(\Theta', \Theta) - \lambda(\sum_i \Theta'_i - 1)] = 0 \tag{2.58}$$

Therefore:

$$\begin{aligned}
\lambda &= \frac{\partial}{\partial \Theta'_i} R(\Theta', \Theta) && (2.59)\\
&= \frac{\partial}{\partial \Theta'_i} \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \log \frac{p_{\Theta'}(A,E \mid F)}{p_\Theta(A,E \mid F)} && (2.60)\\
&= \frac{\partial}{\partial \Theta'_i} \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \log p_{\Theta'}(A,E \mid F) && (2.61)\\
&= \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \frac{\partial}{\partial \Theta'_i} \log p_{\Theta'}(A,E \mid F) && (2.62)\\
&= \frac{1}{\Theta'_i} \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \Theta'_i \frac{\partial}{\partial \Theta'_i} \log p_{\Theta'}(A,E \mid F) && (2.63)
\end{aligned}$$

Finally, by using equations 2.47, 2.48, and 2.49:

$$\lambda = \frac{1}{\Theta'_i} \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) \Theta'_i \frac{\partial}{\partial \Theta'_i} \log p_{\Theta'}(A,E \mid F) \tag{2.64}$$

$$= \frac{1}{\Theta_i'} \sum_{(E,F)} \tilde{p}(E,F) \sum_A p_\Theta(A \mid E,F) C_{i,E,A,F} \tag{2.65}$$

$$= \frac{1}{\Theta_i'} \sum_{(E,F)} \tilde{p}(E,F) C_{\Theta_i,E,F} \tag{2.66}$$

$$= \frac{1}{\Theta_i'} C_{\Theta_i} \tag{2.67}$$

Solving for Θ_i' and using the fact that $\sum_i \Theta_i = 1$, the desired re-estimation formula for a parameter is attained:

$$\Theta_i' = \frac{C_{\Theta_i}}{\sum_i C_{\Theta_i}} \tag{2.68}$$

Setting the new value of a parameter Θ_i' to the count that it receives in a training corpus relative to an existing parameter set, normalized by the appropriate denominator, maximizes the relative objective function. Hence, the likelihood $\mathcal{L}(\Theta)$ will either remain the same (in which case all the parameters retained their original values), or it will increase.

Some care must be taken when using the EM algorithm.

- Since $C_{i,E,A,F} = \Theta_i \frac{\partial}{\partial \Theta_i} \log p_\Theta(A, E \mid F)$, this will always be 0 (or undefined depending on the partial derivative) if Θ_i is 0. Thus, it is important that the initial values for each Θ_i be non-0.

- While the likelihood $\mathcal{L}(\Theta)$ is monotone increasing, there could be many local maxima. For objective functions that are concave there is a unique global maximum. In this case, the EM algorithm will converge to this global maximum. This is true for Model 1 (see chapter 5). But all

other models described in this thesis do not have a global maximum. For these models, the final parameter settings to which the EM algorithm converges depends upon the initial parameters. This isn't even guaranteed to be a local maximum. It is guaranteed to converge to a *critical point* of $\mathcal{L}$ (all partial derivatives of $\mathcal{L}$ with respect to the parameters Θ_i are 0). A *saddle point* is an example of critical point that is not a local maximum.

- Since the EM algorithm converges to one of many possible critical points, it is important to pick initial parameters using knowledge about the problem being solved and the models being used. One common trick is to assign reasonable parameter values by manually aligning some pairs of sentences (E,F). This is called a hand alignment (see section 4.3). A hand alignment *exposes* the hidden structure so one can directly accumulate counts for the observed events. In fact, one can use hand-created alignments for any portion of the training data, and this does not effect the proof that the likelihood will increase. The count formula for $C_{i,E,A,F}$ has to be replaced for these sentences of course.

- One problem with the EM algorithm is determining how many times to iterate. While the likelihood of the training data is guaranteed to increase, one has to be careful about under and over training. One could measure the cross entropy on a test set, but this is no guarantee of improvement. The only safe way to know that you have done

enough iterations is to try the parameter sets on some *held-out* data using a decoder and language model. Dempster, Laird and Rubin discuss convergence issues in their seminal paper[20]. In particular, they mention that some parameters may converge rapidly, whereas others require many iterations. Further convergence issues, including a specific example utilizing hidden Markov models are given by Nadas and Mercer[58].

- When deriving the count formula for $C_{i,E,A,F}$, care must be taken to ensure there is no division by 0. In particular, it is necessary to confirm that $p_\Theta(A, E \mid F)$ is not 0, otherwise the partial derivative of the log will introduce division by 0.

2.4 The Decoder and Language Model

In the previous section, statistical modeling using a source channel paradigm was described. To build a complete system, one still has to implement a decoder and a language model. These are major efforts unto themselves, and not undertaken in this thesis. Nevertheless, a few comments are in order.

Generally, the designer of an NLU system writes a decoder that searches the meaning space for the formal language that maximizes equation 2.2. The most common search procedures are the *stack* search[33] and the Viterbi beam search[44], both variants of the $A^\star$ search[72].

As the decoder searches the space of F, the translation model parameters

are used to calculate either the maximum likelihood value:

$$p_\Theta(E \mid F) = \sum_A p_\Theta(A, E \mid F) \tag{2.69}$$

or the Viterbi value:

$$p_\Theta(E \mid F) \approx \max_A p_\Theta(A, E \mid F) \tag{2.70}$$

This translation model value is multiplied by a language model score for the formal language. The search procedure over the space of F is called a decoder.

Language models are usually implemented as n-gram models, which have been proven to be successful[2]. Recently researchers have leaned toward maximum-entropy based language models[67, 71, 40, 73].

One enhancement is that the translation and language model values can be weighted to increase performance:

$$p(F)^{\lambda_{LM}}\ p_\Theta(E \mid F)^{(1-\lambda_{LM})} \tag{2.71}$$

The weights are calculated empirically using held-out data.

While this section briefly described the most common decoders and language models for statistical NLU, this thesis uses even a simpler search strategy for ATIS. This is called the pattern matcher, and is described in section 4.4.

Chapter 3

Air Travel Information Service

So far, this thesis has presented statistical natural language understanding in a very theoretical way. The basic elements of source-channel modeling and maximum likelihood estimation have been presented, but not applied to a domain. The domain used in this thesis is the Air Travel Information Service (ATIS) [69], used by ARPA for the last five years in the Human Language Technology workshops. The primary reasons for selecting ATIS are:

- There has been a concerted effort to collect data for this domain, called the MADCOW initiative[27]. Thus, no additional data collection was required.
- The results of this thesis can be compared with the results of the other ARPA participants, most notably AT&T, BBN, CMU, MIT, SRI, and Unisys.

3.1 The ATIS Corpus

The ATIS corpus used in this thesis contains 25,483 sentences, of which there are 23,412 unique ones. These data do not include three test sets of December 1993, December 1994, or the development test set for the December 1994 evaluation. These test sets are being reserved for testing purposes only, for it is "cheating" to test a system on data used in its development. The difference between the total number of sentences and the unique ones is some sentences are ambiguous, and are thus included with more than one interpretation.

For each sentence, the ATIS corpus contains many types of data. The data relevant to this thesis are:

sro The speech recognition output, which has been hand-annotated to mark "exceptional" speech events. An example of the text in an .sro file is:

> [pop] . [smack] . [inhale] . i'd like to find a f:light . betwee:n,
> [um] . Dallas, and Philadelphia

There are two types of acoustic phenomena annotated, the non-speech phenomena as shown above (e.g. [smack]), and speech phenomena like false starts. Using lex [39], the data were cleaned to remove the acoustic and false start annotations, to produce:

> i'd like to find a flight between Dallas and Philadelphia

cat The "category" or "class" for this sentence. Each sentence is categorized as being context-independent(A), context-dependent(D), or X [5].

Category X queries are unanswerable for a variety of reasons, including truncated speech (i.e. the user turned off the microphone too early) and queries about data not in the database.

win The NL-Parse for the sentence. NL-Parse is an unambiguous English-like language that can be parsed with a context free grammar[26]. In fact, NL-Parse can be parsed with an LALR(3) grammar[1]. The corpus contains NL-Parse for all category A and D queries, but most category X queries do not have NL-Parse. The NL-Parse for the above query is:

List flights from Dallas and to Philadelphia

sql SQL for the query that generates the "minimal" answer. The minimal answer includes all the rows from the appropriate tables that satisfy the query. Only columns needed to answer the query are displayed. For example, the minimal answer for the above query would include just flight ids.

sql2 Similar to above, this SQL generates the "maximal" answer. This answer includes more columns than were explicitly requested by the user, but are reasonable to present. For instance, in the above example, it would be reasonable to display the departure and arrival airports. If the user asked for the earliest flight, then it would be reasonable to display the departure and arrival times as well.

ref This is the answer from the database if one runs the .sql query. This is

stored using a tuple notation similar to a Lisp s-exp [82].

ref2 This is the answer from the database if one runs the .sql2 query, also stored using tuples.

Unfortunately, not all the data in the corpus have been annotated with NL-Parse. This is relevant because NL-Parse forms the basis for the formal language. There are 5627 class A sentences for training and 600 class A sentences for smoothing. This leaves approximately 6000 sentences that are class A and lacking NL-Parse.

3.2 The ATIS Database

The ATIS database contains 27 tables. The most important ones are shown in table 3.1.

Most queries are relatively easy to evaluate, requiring joins between tables where appropriate. Queries that are difficult to answer are ones that require complex grouping in the SQL. For example, asking the busiest hour for each airport would require grouping the departing and arriving flights into airports and hour of departure or arrival, then counting these, then finding the hour that has the maximum count for each airport. This is extremely difficult to do in SQL.

Since most queries are about flights and fares, the columns for these are presented in tables 3.2 and 3.3.

Table Name	*Num Cols*	*Description*
aircraft	15	aircraft performance statistics.
airline	3	abbreviated airline codes and official airline names.
airport	7	the airport name and an indication of its location. an airport may serve more than one city.
airport_service	5	a list of the airports that serve a city.
city	5	a description of cities. Cities are distinct from airports.
fare	9	a table of the fares listed by class, and where applicable by airline and by restriction.
fare_basis	9	a table describing the modified booking classes that determine a fare.
flight	15	a table containing the primary information for flights.
food_service	4	meals included under meal codes.
ground_service	4	fares for different types of ground transportation between a city and an airport.
restriction	8	this table describes the restrictions that apply to restricted fares.

Table 3.1: Important ATIS Relations

Column Name	*Description*
flight_id	a unique identifier for a flight.
flight_days	a string of codes for days of the week, with 'not' to exclude days, and 'daily' for all.
from_airport	the origin airport for a flight or fare.
to_airport	the destination airport for a flight or fare.
departure_time	the departure time for a flight.
arrival_time	the arrival time for a flight.
airline_flight	the sequence of airline codes and flight numbers for a flight
airline_code	the code for an airline.
flight_number	the flight number for a direct flight.
aircraft_code_sequence	the sequence of codes for the aircraft used on a flight.
meal_code	a code indicating meal service.
stops	the number of intermediate stops on a flight.
connections	the number of connections on a flight.
dual_carrier	'yes' if a flight has a dual carrier, 'no' otherwise.
time_elapsed	the total elapsed time for a flight.

Table 3.2: Columns in the FLIGHTS Relation

Column Name	*Description*
fare_id	a unique identifier for a fare.
from_airport	the origin airport for a flight or fare.
to_airport	the destination airport for a flight or fare.
fare_basis_code	a modified booking class that determines a fare.
fare_airline	the code for an airline that has its own fare for a fare code.
restriction_code	the code for a set of restrictions that apply to a fare.
one_direction_cost	the cost of a fare in one direction.
round_trip_cost	the cost of a fare for a round trip.
round_trip_required	'yes' if a fare applies only to round-trip travel, 'no' otherwise.

Table 3.3: Columns in the FARES Relation

A close inspection of the flight and fare tables will reveal some interesting facts. First, the only columns they have in common are from_airport and to_airport. In order to do joins[77], there is another table which lists flight_id and fare_id pairs that go together. A flight can have more than one fare, and a fare might be applicable to more than one flight. Second, cities are not mentioned in either of these, since flights and fares are between airports. Thus, if you wish to fly from a city, one must join the airport code from the flight or fare table to the city table, using the airport service table as an intermediate table for the join.

3.3 NL-Parse

NL-Parse is an English-like language for expressing ATIS queries [26] that is nearly unambiguous. Indeed a yacc [39] program was written to parse NL-Parse statements, and generate SQL. Yacc can handle LALR(1) grammars. It is well known that LALR(k) grammars are unambiguous[1]. The LEX tokenizer required 2 additional characters of lookahead, making the implementation LALR(3). However, yacc found three shift-reduce conflicts. Fortunately, these ambiguities result from pathological cases, and yacc's default behavior of preferring a shift action to a reduction is the correct parsing recovery action[1]. An extended Backus-Naur Form of NL-Parse is given in appendix A. This appendix also describes the ambiguity in NL-Parse. A more complete grammar is given in appendix B. This contains the yacc

grammar (stripped of the semantic actions that generate SQL and control the error recovery). In order to understand this thesis, a few example NL-Parse statements should suffice:

- List earliest flights from Boston and to Denver
- List early morning flights from Boston and to Denver
- List United flights from Boston and to Denver and whose flight number is 201 and serving breakfast
- List food services whose meal description is breakfast and served on United flights from Boston and to Denver and whose flight number is 201
- List fares charged for United flights (from Boston and to Denver and whose flight number is 201)
- List aircraft equipping United flights from Boston and to Denver and whose flight number is 201

Chapter 4

Statistical NLU for ATIS

4.1 Analysis of English

In order to reduce the number of parameters in the translation models, the English is analyzed into *tagged* English using a statistical tagger[18]. This did require annotating some of the data manually. The key point is that one can't possibly train a statistical system with just a few thousand sentences if the English vocabulary contains an extra 1500 words for city names, airport names, dates, prices, times and so forth. Too many parameters will not be seen in the training data, and hence have 0 probability associated with their events. Parameters can be tied together, but since tagging is better understood, this was preferred over tieing.

No attempt was made to determine classes automatically. Though the availability of the SQL and NL-Parse data would make this a much easier

Tag Name	*Description*
ACODE	an airline code, like DL.
AIR	an airline name, like Delta.
ARP	an airport name, like LAX.
CITY	a city name, like Boston.
CODE	a generic code, like DC10.
DATE	a date, like 6/21/90.
DAY	a day, like Sunday.
NUM	a number, like one hundred twenty.
STATE	a state, like Texas.
PRICE	a dollar amount for a ticket, like four hundred dollars.
TIME	a departure or arrival time, like 4 p m.
COUNTRY	a country, like Canada.
MONTH	a month when not part of a date, like December.

Table 4.1: Tag set for ATIS

problem than determining classes given English only. Upon determining some basic class instances, this could be used to generate training data for a statistical tagger. Then, this process could be iterated. Although hand tagged data were used to train the statistical tagger, this should not be considered a weak link in the portability claims of this thesis.

The tags used are shown in table 4.1.

Without these tags, there would be vastly too many parameters, and significantly more training data would be needed. As it is, numerous taggable classes were not included, and this probably affects performance. Examples include MEALs (breakfast, lunch, etc) and TIMERANGEs (evening, morning, etc).

The translation models can make valuable use of these tags, besides just reducing the number of parameters. The tags were designed so that whenever

one of these tags appears in the English, the corresponding tag also appears in the formal language. Thus, these tags are valid in both the English and formal language vocabularies. One can enforce the constraint that a tag in the English must be generated from the corresponding tag in the formal language. This is done at training by setting $p(e \mid TAG) = 0$ for all $e \neq TAG$. This also helps decoding (see section 4.4).

4.2 Synthesis of Meaning

The formal language used are the nodes contained in the pre-order traversal of the parse tree for *cleaned and tagged* NL-Parse. Tagging the NL-Parse is much simpler than tagging English, as there are strong clues in the highly regular NL-Parse. For example, a CITY must follow "from", "to", or "stopping in". Of course, an AIRPORT has the same requirement. Thus, if there were a CITY and AIRPORT with the same name, then there would be a problem. Fortunately, it was possible to resolve all ambiguities of this sort.

One issue in tagging the formal language is that improvements can be attained in most models by adding thematic roles, sense suffixes, or both. For example, a CITY can be tagged as a FROM_CITY or a TO_CITY if this is built into the statistical tagger's training set. For most of the results presented in this thesis, the tags contain sense suffixes. For example, the first CITY in the English becomes CITY_1, the second one CITY_2 and so forth. When the NL-Parse is tagged, upon discovering a tag, the association list of

CITY names already found in the English is consulted to decide which sense to use.

An inspection of NL-Parse shows that it highly resembles SQL. Between tables, there is always an English substring denoting the join, as in "List aircraft equipping flights $\cdots$". Often NL-Parse contains constructs which resemble the derivation from SQL, and leads to NL-Parse which is significantly more awkward and verbose than the corresponding English. This does not lend itself well to a statistical natural language understanding system in which English words are generated from formal language words, since now clauses in the formal language will have to generate words in the English. Thus, the original NL-Parse was cleaned. Some examples of the problems fixed are:

- Weekday flights in NL-Parse require the conjunction of each day, for example, "flying on Monday and flying on Tuesday and $\cdots$ and flying on Friday". This became "flying on weekdays".

- Arrival days in NL-Parse are handled in an ugly way, for example "(flying on Monday and overnight) or (flying on Tuesday and not overnight)". This became "arriving on Tuesdays".

- Days mentioned by the user were converted to dates. These were converted back to days. For example, if the user said "give me a flight on Saturday", the original NL-Parse might be "flying on 2/3/96". This became "flying on Saturday".

- Approximate times are handled by a time interval in NL-Parse. If an English query asked for a flight "around noon", the NL-Parse contains "departing between 1130 and 1230". This became "departing around 1200".

- Flights and fares were merged together. The original NL-Parse made this distinction even though users do not. For example, if someone wants a "first class flight from Boston to Denver", original NL-Parse requires "List flights from Boston and to Denver and having prices of first class fares". This became "List first class flights from Boston and to Denver".

The tagged and cleaned NL-Parse is then parsed with the grammar given in appendix B. The parse tree has some modifications to what the "pure" parse tree would produce:

- Only semantically relevant nodes are kept. Many of the nonterminals are used to make the rules more compact, and contain no semantic relevance. For example, the nodes "corpora", "query", and "just" are all semantically irrelevant. When a semantically irrelevant node is excluded, one of the children is promoted to the parent location in the parse tree, and dominates the remaining children.

- NL-Parse requires full Boolean expressions. These are implemented as binary operators in the grammar, but are converted to n-ary operators in the parse tree.

- Nonterminal nodes that ambiguously accept different children, are given a suffix to indicate the sense. For example, "from" can be used either to designate a departure city tag, a departure airport tag, a departure city expression (e.g. "cities located in Massachusetts"), or a departure airport expression. The created parse tree uses "from:city", "from:airport", "from:cities", or "from:airports" to distinguish these cases.

- Default nodes are sometimes inserted when an optional NL-Parse word is omitted, if convenient to do so in yacc. For example, one can say "List all departure time of flights" or "List departure time of flights". In either case, the parse tree will have an "all" node as the parent of "departure time". In addition, all queries that request a column be displayed, have "Extract" in them in addition to "List" in the cleaned version.

For statistical NLU to be successful, the synthesized formal language must contain the semantic concepts present in the English, and not contain redundant or meaningless words in it. This will become more evident in chapter 8. Thus, an open area of research is how to get a formal language representation with minimal human effort. One can begin from an SQL representation, but learning when the SQL is verbose due to the database design and not the English is a challenge. While not perfect, the formal language used for ATIS in this thesis is the preorder traversal of the cleaned

NL-Parse parse tree.

4.3 Hand Alignments

The models used by AT&T[65] and BBN[54] required hand aligned data for training. If they trained the models using the EM algorithm without this supervision, then the resulting parameters did not work nearly as well. In order to see if the models investigated in this thesis suffer from the same fate, hand alignments were created manually for 3575 of the 5627 class A sentences.[1] This was done by four different annotators, using a tool that allows one to connect each English word with the formal word most likely to generate it. The decisions on how ambiguous words should align were left to each annotator. One guiding principle was given, basically, to try to keep strings of words together. Since most of the models used in this thesis are based on substrings (see chapter 6), one should try to keep consecutive words aligned to the same meaning word whenever possible. Using this tool, I was able to align approximately 100 sentences per hour, if an initial alignment is created automatically from one of the models described in this thesis (trained on 100% hidden data of course). It should be emphasized that the goal of this thesis is to find a model which works with no hand aligned data. Having the hand aligned data will facilitate analyzing the strengths and weaknesses of the models. In particular, one hopes to discover that some models can be

[1] See section 5.4 for how these hand aligned sentences are used.

trained with no hand aligned data at all.

4.4 The ATIS Decoder/Pattern Matcher

Having described the analysis and synthesis stages of the statistical NLU used in this thesis, this section describes the decoder used in this thesis. The initial development of a stack or Viterbi decoder for a natural language application can take one or two man years, and the research to improve the search is never ending. This effort is warranted if a good model is discovered. To focus on a decoder and LM before finding such a model would be premature. A poor man's version of these was implemented for this thesis, which is called the *pattern matcher*.

The ATIS pattern matcher simply tries all formal language patterns that are seen in the training set that have the exact same tags as the English. Since training requires an English tag to come from the corresponding formal tag, only patterns that contain the exact same tags will have a non-zero probability $p(E \mid F)$. This significantly speeds the pattern matcher, and reduces the error rate.

While there are 25,483 sentences in the ATIS corpus, formal language exists for approximately 13,000 of them, of which there are approximately 4,000 unique formal language sentences. So, rather than developing a complex search procedure through the meaning space, the pattern matcher scores each of the 4,000 meanings in its pattern vocabulary using $p(E \mid F) * p(F)$,

where F is the formal language and E is the tagged English. The translation models provide $p(E \mid F)$. The language model score for $p(F)$ is just the unigram probability distribution of the patterns. These two values are multiplied together to get the final score for a pattern. The pattern with the highest score (and matching tags) is selected. An initial experiment to determine a weighting factor found no improvement over equal weighting (see section 2.4).

In order to select the best pattern, two techniques are often used:

- Maximum Likelihood Decoding - This considers all possible ways of generating an English sentence from a formal language pattern. Even though most sentences have only a few sensible ways of doing this, all possible ways contribute to the probability. This is only computationally feasible for some of the models.

- Viterbi Decoding - This considers only the most probable way to generate an English sentence from a formal language pattern. If one has a well designed formal language and a good model, then most of the maximum likelihood probability will come from the Viterbi alignment probability. But if the modeling has flaws, then summing over all possible alignments, not just the Viterbi alignment, can lead to improvements in performance.

There is one disadvantage to using a pattern matcher. The pattern vocabulary will certainly be incomplete. For example, in the DEV94 set of

410 class A sentences, 70 of them are missing from the pattern matcher vocabulary. It is possible to solve some of these coverage problems with good classes. For example, MEALs were not tagged. Thus, there are patterns that explicitly contain "serving breakfast", "serving lunch", or "serving dinner". Hence a missing pattern might actually exist, but with a different MEAL in it. This accounted for 25% of the missing patterns. To reduce the missing patterns by 75%, it was necessary to allow for the insertion or deletion of meaning fragments. For example, often a pattern was "close" to the reference pattern, in that it differed by one or two NL-Parse conjuncts. Rather than trying to spend several weeks making a fragment-based pattern matcher, one can get an upper bound on performance by augmenting the pattern matcher vocabulary with the test set pattern if it is missing from the pattern vocabulary. The results will naturally be much better with the augmented pattern vocabulary. The true measure of performance, unfortunately, has to use the training pattern vocabulary only. But the results of using the augmented pattern vocabulary still have value. If one had a decoder that searched the formal language space in a reasonable way, and used a good language model for $p(F)$ in equation 2.2, then hopefully the system could get results close to those attained by using the augmented pattern vocabulary.

4.5 Evaluation

Once the pattern matcher has selected the most likely pattern for a test sentence, this needs to be evaluated according to an evaluation metric.

4.5.1 Evaluation Metrics

There are many ways to evaluate the output of an NLU system. In fact, throughout the five year HLT workshop, this has been an ongoing debate. In the end, NIST decided to use the *Common Answer Specification* (CAS) as its evaluation metric[9]. This requires comparing the output tuples from the "reference" answer, to the output tuples produced by a system. Of course, the comparison program must allow for different permutations and the inclusion of additional reasonable columns. Most of the participants in the HLT workshops have been unhappy with the CAS. In fact, no fewer than eleven papers discussed alternate evaluation metrics or argued against using the CAS metric[69, 28, 4, 68, 70, 79, 59, 29, 31, 36, 57].

This thesis will present the results of the models using using two metrics:

- Exact Match - When a sentence is decoded into its most likely meaning, this can be compared for equality to the reference answer. This is a very strict metric, as often the wrong answer might still have the same meaning. But nevertheless, there seems to be a correlation between the exact match scores and the CAS scores. Exact match scores are presented along with each model.

- Common Answer Specification - When a sentence is decoded into its most likely meaning, the CAS performance is attained as follows:

  1. Convert the meaning into tagged NL-Parse.
  2. Use an association list obtained during the analysis of English to fill in the tag values.
  3. Process the NL-Parse by the yacc parser to produce SQL.
  4. Submit the SQL to DB2 to produce the answer tuples.
  5. Run the NIST comparator to compare the answer tuples to the reference answer tuples.

  Since running the SQL on the database is a lengthy process, this is done for only a subset of the experiments. The CAS results are presented in chapter 10.

4.5.2 Evaluation Test Sets

There are three test sets:

- DEV94 - This is the development test set given in mid 1994 to the participants in the HLT workshop. One can use this set repeatedly to get results, but the ARPA participants were not allowed to look at the data. This contains 410 class A sentences.
- DEC93 - This is the official December 1993 evaluation set. This contains 448 class A sentences.

- DEC94 - This is the official December 1994 evaluation set. This contains 445 class A sentences.

There are several ways that one could use these three test sets. Most participants looked at the December 1993 test data, so that it could be adjudicated for the final results. That is, they were allowed to look at the test data, and report errors they found in the test data to NIST. NIST would then consider the requests, and when appropriate, modify the reference answer for re-evaluation. Thus, most participants have had the benefit of looking at the DEC93 data while developing their DEC94 systems, even though they could not look at the DEV94 data that they already had.

In the development of this thesis, I decided to look at the DEV94 data instead. This allows me to report valid numbers for the DEC93 and DEC94 official evaluations, which can then be compared to other participants.[2] The DEC93 results would not be valid if these data were examined. Thus, while other research groups had the benefit of looking at DEC93 in the development of their DEC94 systems, the development of the models in this thesis was facilitated by the decision to look at the errors made in DEV94.

In this thesis, DEV94 is used to test each model using maximum likelihood decoding using the exact match criterion. These results are presented with each model using various amounts of training data. In the final chapter, CAS results are presented for all three test sets, with and without augmenting the

[2]Though technically one could argue that I had the benefit of looking at DEV94.

pattern matcher vocabulary to include missing patterns.

Chapter 5

Basic Word Alignment Model

5.1 Introduction

This chapter presents the first model, called model 1. Being the first model discussed, model 1 is presented in greater depth than the other models in this thesis. Notation is introduced, count update formulae for the parameters are derived, the issues affecting the EM algorithm's performance are discussed, and the results are analyzed with respect to the amount and composition of the training data. This is not done for subsequent models.

Model 1 is a basic word alignment model, in which each English word is generated by a formal language word independently from the other English words, much like the die tossing experiment outlined in section 2.3. This is the same as model 1 used by IBM in machine translation[11, 12, 13]. There is a many-to-one mapping from the English words to the formal language. For

example, the formal word "flights" might generate the English words "show me the flight".

The generation of English words from formal language yields an *alignment* between the English and meaning. An example alignment is shown in figure 5.1. If the English sentence has $\ell(E)$ words, and the formal language sentence has $\ell(F)$, then there are $\ell(F)^{\ell(E)}$ possible alignments between words in E and words in F, since each $e \in E$ must be generated from an $f \in F$. Each alignment will have a probability associated with it. In figure 5.1, "the" is aligned to "List". It is nearly equiprobable for "the" to be aligned to "flights". But alignments that have the English word "flights" aligned to anything but the formal language word "flights" have very low probability. In the source-channel model, the speaker's thoughts in the formal language space are sent through a noisy channel. The observed English is the output from the channel. Alignments are the basis of the statistical models for the channel developed in this thesis. Each alignment is possible; there is no "correct" alignment. In figure 5.1, there are two possible alignments that make sense for this sentence. For other examples, hundreds of alignments make sense. But for all sentence pairs, each of the exponentially large number of alignments contribute to the likelihood.

Let E denote an English sentence, which consists of $\ell(E)$ words, denoted $e_1 \cdots e_{\ell(E)}$. Thus, E is a tuple of order $\ell(E)$, whose individual elements are the words e_i. Similarly, let F denote the formal language. F is a tuple of order $\ell(F)$, whose individual elements are the pre-order traversal of the

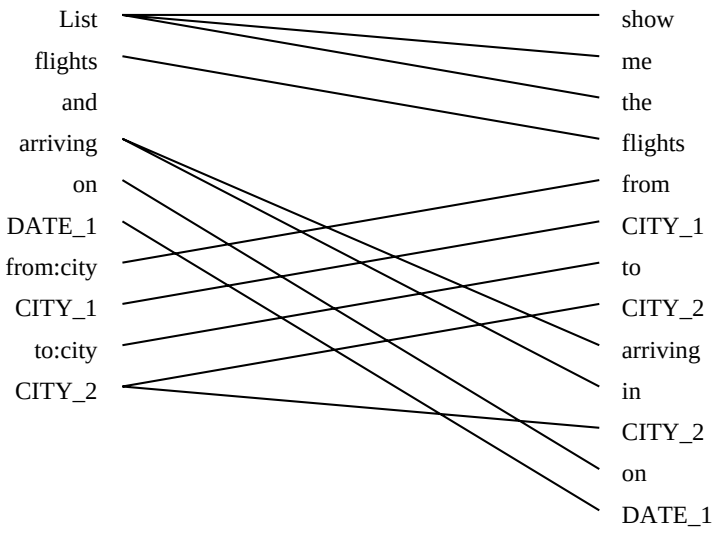

Figure 5.1: Example Alignment

parse tree for the cleaned and tagged NL-Parse. The individual elements of F are the formal words f_i. An alignment A is a tuple of order $\ell(E)$. The elements a_i are integers in the range $1 \cdots \ell(F)$. Each e_i is aligned to one and only one formal language node f_{a_i}. Thus, in figure 5.1, $E =$ [show, me, the flights, from, CITY_1, to CITY_2, arriving, in, CITY_2, on, DATE_1]. $F =$ [List, flights, and, arriving, on, DATE_1, from:city, CITY_1, to:city, CITY_2]. $A =$ [1, 1, 1, 2, 7, 8, 9, 10, 4, 4, 10, 5, 6].

5.2 Formulae

For any (E, F), there are $\ell(F)^{\ell(E)}$ possible alignments, each of which can have non-zero probability, though the Viterbi alignment[78, 21] should comprise the bulk of this probability for a good model. One can derive the formula for $p(E, A \mid F)$ by repeated use of Bayes rule[13]:

$$p(E \mid F) = \sum_A p(E, A \mid F) \tag{5.1}$$

$$p(E, A \mid F) = p(E, A \mid F, \ell(E))p(\ell(E) \mid F) \tag{5.2}$$

$$= p(E \mid A, F, \ell(E))p(A \mid F, \ell(E))p(\ell(E) \mid F) \tag{5.3}$$

The assumptions in model 1 are:

- The length of the English sentence E is determined entirely by the length of F, not the words in F. Hence, $p(\ell(E) \mid F)$ can be modeled by the parameter $p(\ell(E) \mid \ell(F))$. Intuitively, the length of E is likely

to be proportional to the length of F.

- Each alignment A between E and F is equally likely. Hence $p(A \mid \ell(E), F)$ is replaced by $\frac{1}{\ell(F)^{\ell(E)}}$. Intuitively, this says that word order in the English and formal language does not matter. In figure 5.1, if the English were permuted to "The flights arriving in CITY_2 on DATE_1 from CITY_1 to CITY_2 show me", then model 1 yields the same likelihood as for the original English. While word order does matter for virtually all of the sentences, the assumption that it does not still gives a model that attains reasonable results. This is also important in order to provide initial statistics for the more powerful models presented in subsequent chapters.

- The probability of generating E given A and F depends only upon the words to which E is aligned. Each $e \in E$ is generated independently. Hence $p(E \mid A, F, \ell(E))$ is replaced by $\prod_{i=1}^{\ell(E)} p(e_i \mid f_{a_i})$. Future models relax this strong assumption. This assumption would be analogous to a machine translation system that did word for word translation, yet allowing for different word orders. This approach is clearly insufficient for machine translation, as it is for natural language understanding. But given enough training data, or a limited domain like ATIS, reasonable results can be attained.[13].

Applying these substitutions, model 1 becomes:

$$p(E, A \mid F) = \frac{p(\ell(E) \mid \ell(F))}{\ell(F)^{\ell(E)}} \prod_{i=1}^{\ell(E)} p(e_i \mid f_{a_i}) \tag{5.4}$$

One particularly nice feature of this model is that:

$$\begin{aligned} p(E \mid F) &= \sum_A \frac{p(\ell(E) \mid \ell(F))}{\ell(F)^{\ell(E)}} \prod_{i=1}^{\ell(E)} p(e_i \mid f_{a_i}) && (5.5) \\ &= \frac{p(\ell(E) \mid \ell(F))}{\ell(F)^{\ell(E)}} \sum_A \prod_{i=1}^{\ell(E)} p(e_i \mid f_{a_i}) && (5.6) \\ &= \frac{p(\ell(E) \mid \ell(F))}{\ell(F)^{\ell(E)}} \prod_{i=1}^{\ell(E)} p(e_i \mid F) && (5.7) \\ p(e \mid F) &= \sum_{i=1}^{\ell(F)} p(e \mid f_i) && (5.8) \end{aligned}$$

The exchanging of the sum and products between equation 5.6 and 5.7 is because the sum and the product use all possible alignments. Hence, using associativity and distributivity of addition and multiplication, the sums and product can be interchanged[13]. This is important because for a given E and F, this leads to an $O(\ell(E)\ell(F))$ time algorithm to compute $p(E \mid F)$.

There are two types of parameters in model 1:

sentence lengths These are the $p(\ell(E) \mid \ell(F))$ parameters. These are not hidden, and can be trained to their maximum likelihood estimates by counting the times that an English sentence of length $\ell(E)$ occurs with a formal language sentence of length $\ell(F)$.

translation probabilities These are the $p(e_i \mid f_{a_i})$ parameters. These are hidden parameters (unless the training data are entirely hand aligned). Using equation 2.57, the count update formulae can be derived, and the EM algorithm run.

5.3 Count Derivation

To derive the count of a parameter $p(e' \mid f')$, one can apply formula 2.57 to equation 5.7:

$$
\begin{aligned}
c(e' \mid f') &= p(e' \mid f') \frac{\partial}{\partial p(e' \mid f')} \log p(E \mid F) && (5.9)\\
&= p(e' \mid f') \frac{1}{p(E \mid F)} \frac{\partial}{\partial p(e' \mid f')} p(E \mid F) && (5.10)\\
&= p(e' \mid f') \frac{1}{p(E \mid F)} \frac{p(\ell(E) \mid \ell(F))}{\ell(F)^{\ell(E)}} \prod_{e \neq e'} p(e \mid F) && (5.11)\\
&= p(e' \mid f') \frac{1}{\prod_e p(e \mid F)} \prod_{e \neq e'} p(e \mid F) && (5.12)\\
&= \frac{p(e' \mid f')}{p(e' \mid F)} && (5.13)
\end{aligned}
$$

As mentioned in section 2.3, one must be careful when applying formula 5.13. It is tempting to apply this formula blindly. But one must only accumulate counts for a sentence if $p(E \mid F)$ is not 0, since this appears in the denominator in equation 5.10.

5.4 Training

The model 1 translation probabilities were trained for 512 iterations. The sentence length probabilities were trained on one iteration, since they are not hidden. The initial translation probabilities were set to uniform distributions, except that $p(e \mid TAG)$ is set to 0 for all $e \neq TAG$. This then forces the EM algorithm to only utilize alignments consistent with the tag constraints. Hence, bad alignments due to tags being misaligned are excluded. These zeros also mean that the pattern matcher only needs to try meaning patterns that have the exact same set of tags as the English, since no other meaning words can generate the English tags. See section 4.4 for more on the pattern matcher.

The algorithm to perform an iteration of the EM is trivial:

1. Initialize a two dimensional array holding counts for $c(e \mid f)$ to 0. Naturally, this array has size $\ell(Voc_E) \times \ell(Voc_F)$, where Voc_E and Voc_F denote the size of the vocabularies for values of e and f.

2. For each pair of sentences E and F in the training data, compute $p(E \mid F)$, and if non-0, accumulate the counts for the translation probabilities using:

$$\forall_{e \in E f \in F} c(e \mid f) += \frac{p(e \mid f)}{p(e \mid F)} \tag{5.14}$$

3. For each $e \in Voc_E$ and $f \in Voc_F$, normalize the counts back into

probabilities using:

$$p(e \mid f) = \frac{c(e \mid f)}{\sum_{\forall e' \in Voc_E} c(e' \mid f)} \tag{5.15}$$

In order to examine the performance of the EM algorithm 810 sets of parameters were generated as follows. The training corpus consists of 5627 sentences, of which hand alignments exist for 3575 of them. The hand and hidden data were divided into 9 different sets of logarithmic size (all the data, $\frac{1}{2} \cdots \frac{1}{128}$, none of the data). This leads to 81 separate training experiments, each using a different amount of hand aligned and hidden data. For each of these experiments, the parameter sets were saved after exponential numbers of iterations (1, 2, 4, $\cdots$, 512). Thus, there are 810 parameter sets to test for model 1. Note that this scheme means that some sentences will be used more than once, once as hidden data, and once as hand aligned data. This does have the affect of biasing the data in favor in the direction of the sentences used twice. But since these additional instances have "perfect" alignments, this additional bias does not pose problems.

Model 1 has a unique global maximum to which the EM algorithm will eventually converge [13]. This is because $\log p(E, A \mid F)$ is just the sum of $\ell(E) + 1$ logs. Since the log function is concave by Jensen's inequality, and since the sum of concave functions is concave, Model 1 has a concave objective function. Thus, the EM algorithm will eventually converge to the global maximum.

5.5 Smoothing

Smoothing is the process of creating a new probability distribution from an existing one that is less sharp than the original. The probability is increased for low probability events, and decreased for high probability events. This is often necessary because of insufficient training data, over training on the training data, or a mismatch between the training and test data. The method used in this thesis for smoothing is called *deleted interpolation*[2]. The original estimate $p(e \mid f)$ is smoothed by a linear combination:

$$p_s(e \mid f) = \alpha_{1,f} p(e \mid f) + \alpha_{2,f} p(e) + \alpha_{3,f} \frac{1}{\ell(Voc_E)} \tag{5.16}$$

Each parameter is smoothed using three αs, and these are conditioned upon f. For a particular f, $\alpha_{1,f} + \alpha_{2,f} + \alpha_{3,f} = 1$. To improve smoothing, the conditioning on f is done by using the count of f, that is, the number of times f appears in the training and held-out smoothing corpora. The f are *binned* so that f that occur approximately the same number of times share a single set of three alphas. This is intuitively appealing. The more often an f has occurred, the more reliable the estimate of $p(e \mid f)$ will be. If f occurs infrequently, $p(e \mid f)$ will not be accurate, so the estimate can "back-off" to $p(e)$ or a constant.[1]

The exact algorithm to determine the bins is as follows:

[1] In order to preserve 0 values created by the tag constraints, $p(e)$ and $\frac{1}{\ell(Voc_E)}$ have to also be conditioned on f

1. The f are counted for the training set and the held-out smoothing set. This produces $\ell(Voc_F)$ tuples $(F_{training\_count}, F_{heldout\_count})$.

2. The tuples are sorted by increasing order of training count.

3. The tuples are then partitioned into bins while satisfying three criteria:

   - Tuples with the same training count value must appear in the same bin.
   - If a bin contains tuples with a training count $c2 > c1$, then all tuples for training count $c3$ between $c1$ and $c2$ are also in the bin.
   - The sum of the heldout counts of the tuples in a bin must be the minimum possible value ≥ 50.

The binning can be programmed by "visiting" the tuples in increasing order of training count size. A running sum of the heldout count is accumulated. When this exceeds 50, the first bin is defined.[2] All subsequent tuples with the same training count are included in this class, and then the next bin is started.

This approach to smoothing is similar to the cat-cal method of Church[16]. The idea is that one should decide in which bucket to place an f based on a large corpus (hence the sorting by training count size). However, to make sure that the training counts are representative of some new sample, a held-out corpus is used to determine the cutoff point for the bucket.

[2] 50 is selected so that the three alphas have enough smoothing counts so that they are well estimated.

To train the α_f values, the 600 held-out smoothing sentences are used. Uniform values are given to $\alpha_{n,f}$, and the EM algorithm is run. This time, the probabilities $p(e \mid f)$ and $p(e)$ are constants, and the α values are the parameters.

The count accumulation formula for $\alpha_{1,f}$ is derived by applying formula 2.57 to equation 5.16:

$$
\begin{aligned}
c(\alpha_{1,f}) &= \alpha_{1,f} \frac{\partial}{\partial \alpha_{1,f}} \log p_s(E \mid F) && (5.17)\\
&= \alpha_{1,f} \frac{1}{p_s(E \mid F)} \frac{\partial}{\partial \alpha_{1,f}} p_s(E \mid F) && (5.18)\\
&= \alpha_{1,f} \frac{1}{\prod_{i=1}^{\ell(E)} p_s(e_i \mid F)} \frac{\partial}{\partial \alpha_{1,f}} \prod_{i=1}^{\ell(E)} p_s(e_i \mid F) && (5.19)\\
&= \alpha_{1,f} \frac{1}{\prod_{i=1}^{\ell(E)} p_s(e_i \mid F)} \sum_{i=1}^{\ell(E)} p(e_i \mid f) \prod_{e' \neq e_i} p_s(e' \mid F) && (5.20)\\
&= \sum_{i=1}^{\ell(E)} \frac{\alpha_{1,f} p(e_i \mid f)}{p_s(e_i \mid F)} && (5.21)
\end{aligned}
$$

An important observation from equation 5.21 is that the counts for $\alpha_{1,f}$ can be attained by summing over each e in the sentence. The contribution from each e is:

$$
\begin{aligned}
c(\alpha_{1,f}, e) &= \frac{\alpha_{1,f} p(e \mid f)}{p_s(e \mid F)} && (5.22)\\
&= \frac{\alpha_{1,f} p(e \mid f)}{p_s(e \mid F)} \frac{p_s(e \mid f)}{p_s(e \mid f)} && (5.23)\\
&= \frac{p_s(e \mid f)}{p_s(e \mid F)} * \frac{\alpha_{1,f} p(e \mid f)}{p_s(e \mid f)} && (5.24)
\end{aligned}
$$

Equation 5.24 consists of two terms. The first term is nearly identical to that of equation 5.13. The only difference is that the smoothed estimates are used instead of the unsmoothed ones, as defined by equation 5.16. These counts are then multiplied by the second term, which is the amount of $p_s(e \mid f)$ that is due to the $\alpha_{1,f}$ component of equation 5.16. This demonstrates the *chain rule* principle in count derivations. The alpha update formulae are derivable from the parameter update formulae by prorating according to their contribution to $p_s(e \mid f)$.

When all counts have been accumulated for the held-out corpus, the new estimate for $\alpha_{1,f}$ is attained by:

$$\alpha_{n,f} = \frac{c(\alpha_{n,f})}{\sum_{i=1}^{3} c(\alpha_{i,f})} \tag{5.25}$$

Since there are vastly fewer parameters, only 8 iterations of the EM algorithm are needed to train the α_f values.

There are 810 model 1 parameter sets. Rather than smoothing each of these, only the final models after 512 iterations were smoothed. Thus, there are 81 smoothed parameter sets for model 1.

5.6 Results

As mentioned in the preceding sections, there are 810 unsmoothed model 1 parameter sets and 81 smoothed parameter sets. Each of these was used

in the pattern matcher on the DEV94 test set, using both the maximum likelihood and Viterbi decoding criteria (see section 4.4). This yields 1782 experimental results.

5.6.1 Exact Match Maximum Likelihood Results for DEV94

Since 1782 results are too many to present, 162 of the experiments are presented in the form of two tables. The numbers shown in the tables are the number of times out of 410 class A DEV94 sentences, that the pattern matcher found the answer given in the DEV94 corpus for the query. The formal language must exactly match.

In table 5.1, the maximum likelihood decoding results for the DEV94 test set using the smoothed model 1 parameters (512 iterations) are given. In table 5.2, the maximum likelihood decoding results for the DEV94 test set using the unsmoothed model 1 parameters (512 iterations) are given. In order to allow comparison with maximum likelihood and Viterbi decoding, appendix C contains the Viterbi decoding results for 512 and 2 iterations of training.

The DEV94 test set has 410 class A sentences. The best result is when all of the hand data and one sixteenth of the hidden data are used. The results for this experiment are 284/410, or 69%. This is without augmenting the pattern vocabulary to include the 70 missing patterns from the DEV94

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	283	276	273	273	271	269	270	268	267
2813	282	275	269	272	270	268	267	269	262
1406	282	275	271	266	262	251	249	251	243
703	278	273	264	256	244	242	229	231	235
351	284	278	265	261	255	235	200	209	202
175	275	269	265	259	236	227	204	204	197
87	276	276	268	268	237	215	154	168	149
43	276	270	270	249	235	189	165	168	127
0	274	273	267	259	223	184	116	119	154

Table 5.1: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model 1 as a Function of Amount of Hand and Hidden Training Data

test set.[3]

Some general observations from the tables presented in this chapter and in appendix C are:

- Maximum likelihood decoding does better than Viterbi decoding when there are ≥ 351 hidden training sentences. This is true for either the smoothed or unsmoothed parameter sets. This makes sense. The hand aligned data give one alignment, the correct alignment, which is used to update counts for the parameters used in the alignment. The hidden data give counts to all the parameters used in any alignment, which is equivalent to using the maximum likelihood decoding criterion.

- The unsmoothed parameters generally produce better results when fewer iterations are run if the amount of hand data is less than 446

[3]This augmentation is done for the CAS results.

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	279	276	266	268	268	265	263	263	261
2813	279	274	264	265	262	260	251	258	252
1406	283	278	266	260	248	247	241	248	235
703	278	270	255	247	222	217	214	217	201
351	277	273	254	251	233	213	154	186	138
175	271	265	254	247	199	207	150	191	111
87	N/A	275	258	247	210	186	114	138	81
43	273	269	261	245	212	184	112	141	47
0	275	269	263	248	207	165	70	112	154

Table 5.2: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 512 Iterations of Model 1, as a Function of Amount of Hand and Hidden Training Data

sentences. The only exception is if all the hidden data are used. The fact that many results are better when only two iterations are run implies that the training is probably moving the parameter values away from their "true" values, even though the likelihood may be improving.

- When ≤ 446 hand sentences were used, adding more hidden data always helped. In fact, the unsmoothed parameters in table 5.2 still show a sizeable gain in performance when the amount of hidden data are doubled from 2813 to 5627 sentences (without using any hand data). When the number of hand aligned sentences is 3575, the results only got a little better in going from 0 to 5627 hidden sentence. But if no hand data are used, the results improve dramatically. As the amount of hand training data increases, adding more hidden data does not help.

- Smoothing provided little to no advantage when all the hand data were

used. This is because the trained parameters are so good. When less hand data were used, smoothing nearly always helped. In particular, the worst decoding results were improved the most.

- Smoothed statistics computed with no training data (i.e. a linear combination of $p(e)$ and $\frac{1}{\ell(Voc_E)}$), do better than when just ≤ 87 hidden training sentences of ≤ 55 hand training sentences are used. This is probably because the model is overtraining to this limited amount of training data.

- With smoothing, there is a huge dynamic range in the results for varying amounts of hand data when little hidden data are included. But as the amount of hidden data increases, this dynamic range shrunk. If you look at the last column of the smoothed results, you see that the increase in the hidden data allows fewer sentences of hand data to be used. For example, when 5627 hidden sentences and 27 hand sentences are used, one gets nearly identical performance as to when 3575 hand sentences and 43 hidden sentences are used. Thus, this lends some credence to the thesis that completely automatic statistical methods can do as well has hand supervised statistical methods if given enough data.

5.6.2 Cross Entropy Results for DEV94

The *cross entropy*[35], which is a measure of a model's uncertainty on some test data, is calculated as:

$$H = -\sum_{E,F} \tilde{p}(E,F) \log_2(p(E \mid F)) \tag{5.26}$$

$$= -\frac{1}{N} \sum_{E,F} \log_2(p(E \mid F)) \tag{5.27}$$

For each pair of sentences, the sum of the log of the model's prediction $p(E \mid F)$ is normalized by the number of sentences N. Often, the cross entropy is normalized by the number of English words in the test corpus. In this case, the normalizing denominator N is replaced by the sum of the lengths of E, $\sum_E \ell(E)$.

The reason that it is useful to display the entropy of the model on the test data, is that it measures the uncertainty of a model on the test data. The higher the entropy, the more uncertainty the model contains. The relationship between entropy and decoding performance is tentative at best. While a model with a significantly higher entropy generally does worse, models with similar entropies may have very different performance.

The cross entropies of DEV94 using maximum likelihood scores calculated with smoothed model 1 parameters are shown in table 5.3.

Some observations on using the cross entropy as a qualitative measure of performance are:

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	5.61	5.61	5.62	5.61	5.61	5.60	5.60	5.60	5.60
2813	5.64	5.63	5.64	5.64	5.63	5.63	5.62	5.62	5.62
1406	5.68	5.68	5.71	5.70	5.74	5.74	5.74	5.74	5.73
703	5.69	5.69	5.73	5.73	5.77	5.79	5.80	5.80	5.80
351	5.70	5.71	5.75	5.76	5.81	5.85	5.92	5.92	5.92
175	5.71	5.73	5.78	5.81	5.90	5.98	6.05	6.07	6.13
87	5.71	5.73	5.81	5.81	5.90	6.07	6.18	6.21	6.26
43	5.72	5.74	5.81	5.82	5.95	6.12	6.28	6.28	6.56
0	5.72	5.74	5.82	5.85	5.98	6.22	6.48	6.50	7.19

Table 5.3: Maximum Likelihood Cross Entropy for DEV94 Using Smoothed Model 1 as a Function of Amount of Hand and Hidden Training Data

- For maximum likelihood entropy calculations, adding more hidden data to the training improves the cross entropies. For viterbi entropy calculations, adding more hidden data to the training hurts the cross entropy results.

- Adding more hand data improves the cross entropy results when there are not a lot of hidden training data. But when a lot of hidden training data are used, using no hand training data is marginally better.

- The Viterbi cross entropies, which are not shown, are about .25-1 bit worse. When there is little data(the lower right corner of the table), then maximum likelihood decoding produces better cross entropies than Viterbi decoding, which uses a single alignment. As more and more hand data are used in the training, then the benefit of using maximum likelihood over Viterbi cross entropies diminishes.

- For both Viterbi and maximum likelihood unsmoothed cross entropies, the entropies get better for about 8 iterations of training. Then, as more iterations are done, the cross entropies get worse. These indicate that the model is probably over training on the training data. The unsmoothed maximum likelihood cross entropies for 512 iterations are shown in appendix C. In some cases, the unsmoothed entropies are over 20 bits worse than the corresponding smoothed ones.

5.6.3 Viterbi Percentage Results for DEV94

Another qualitative measure of performance is how much of the overall maximum likelihood probability is given by the Viterbi alignment. If the Viterbi alignment is a very small percentage, then this could indicate problems with the model or the training. But if the Viterbi probability is a large percentage, this says that the model is learning how to focus probability to the more likely alignments.

The Viterbi word percentages of DEC94 using smoothed model 1 parameters are shown in table 5.4. What this means is that on a per word basis, the Viterbi probability is n% of the maximum likelihood probability. This is calculated using:

$$Vit\% = 2^{\frac{VitLogProb - MLLogProb}{numWords}} * 100 \tag{5.28}$$

where $VitLogProb$ and $MLLogProb$ are the sums of the log of the model's Viterbi or maximum likelihood prediction of each sentence in DEV94.

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	79.0	76.5	74.3	73.5	71.9	69.5	69.0	69.3	67.8
2813	81.5	79.3	76.4	74.5	73.3	71.3	70.0	69.7	68.2
1406	82.6	81.0	78.4	76.3	73.4	72.3	69.9	69.7	68.9
703	83.1	82.1	80.1	78.7	75.8	73.7	72.5	71.1	69.2
351	83.5	82.7	81.2	80.0	76.9	73.6	69.8	68.8	64.2
175	83.8	83.4	82.2	81.2	78.7	74.5	69.9	69.2	59.9
87	84.1	83.6	81.9	81.3	79.1	74.0	68.8	65.3	53.3
43	84.2	84.0	82.7	82.1	79.7	73.6	68.6	67.0	54.5
0	84.2	84.0	83.1	82.1	79.8	72.3	66.3	61.6	19.9

Table 5.4: Viterbi Percentage of Maximum Likelihood Cross Entropy for DEV94 Using Smoothed Model 1 as a Function of Amount of Hand and Hidden Training Data

Some observations on Viterbi percentages are:

- The Viterbi percentage increases as the amount of hand training data increases. This is intuitively obvious, as having more hand data concentrates more probability on the correct alignment.

- The Viterbi percentage increases as the amount of hidden data increases, only if there are very little hand training data. If there are a lot of hand training data, then adding hidden data hurts the Viterbi percentage.

- The unsmoothed Viterbi percentages show only about a 12% effect from using no hand data to using all the hand data, and this is consistent regardless of the amount of hidden data. But the smoothed Viterbi percentages show that there is a large drop when the hidden and hand

data are low. This is because the smoothing affects the training more drastically when there are little training data.

- The unsmoothed Viterbi percentages improve as more iterations of the EM algorithm are run. Though, to have a big affect, there needs to be a large percentage of hidden data.

Chapter 6

Basic Clumping Models

6.1 Introduction

In model 1, the English words e are generated independently of each other. This leads to problems in decoding when one pattern is a subset of the other. Assuming the sentence length parameter is about the same, the model usually prefers the longer formal language sentence. This is because the longer formal language sentence includes additional formal language words, some of which might help generate rare English words. Consider the sentence "Show me the flights out of CITY_1 into CITY_2". The correct formal language pattern for this is "List flights from:city CITY_1 to:city CITY_2". But, the probability distributions for "from:city" and "to:city" are heavily biased toward the English words "from" and to". Some of the unsmoothed translation probabilities for "from:city" and "to:city" are shown in table 6.1.

Parameter	Value
$p(out\|from:city)$	5.47×10^{-61}
$p(out\|one\_direction\_cost)$	.09
$p(of\|from:city)$	.097
$p(of\|equipping)$	.16
$p(from\|from:city)$	.62
$p(into\|to:city)$	3.44×10^{-12}
$p(to\|to:city)$	.60
$p(into\|late\_evening)$	.09

Table 6.1: Some Model 1 Translation Probabilities

What has happened here? The EM algorithm has to accumulate counts whenever it sees "into" and "out of" in the training set. While these are almost always generated from "to:city" and "from:city", the alignment is hidden. The EM algorithm accumulates counts for all possible ways of generating these words, including incorrect paths. Virtually every flight and fare query includes two cities, but many include other modifiers constraining meals, airlines, aircraft, and so forth. These are much rarer than city constraints. If the training sentences containing "into" and "out of" also contain some of these rarer constraints, the EM algorithm can give more of the counts for these rare words to the rarer formal language word. Thus, the distributions for "from:city" and "to:city" remain sharp. The model has learned improperly how to generate "into" and "out of", but does this hurt? The answer is yes. Suppose in the test data, this same sentence appears. The pattern matcher has to find the most likely pattern to generate the sentence containing these rare words. The correct answer does not contain the rare formal language words which have learned to generate "into" and "out

of". The pattern matcher is free to select a different pattern that contains additional, superfluous constraints. While there is a slight penalty in terms of the sentence length and the denominator exponential term, there could be a large win in term of the $p(e|f)$ translation parameters. Thus, the sentence is decoded incorrectly. An additional constraint or table like "aircraft equipping" is added so that "equipping" can generate the word "of", even though "aircraft" generates nothing. This is called the *spurious word* problem.

So one needs models which are less apt to suffer from spurious words. Clearly, generating each English word e independently from each formal language word f is a bad assumption. The user of a natural language uses syntax and semantics to generate meaningful phrases, and the words within these phrases are highly correlated. Thus, the first enhancement made to model 1 is to model the English generation in terms of one or more substrings called a *clump*. All the words in a clump are required to align to a single f. Clumps are not predefined idioms or positions, rather they are a different way of viewing the alignments of Model 1. For example, in figure 5.1, a plausible 10-clump clumping is:

> [show me the] [flights] [from] [CITY_1] [to] [CITY_2] [arriving in] [CITY_2] [on] [DATE_1]

The hope is that clumping models will help solve spurious word problems, by forcing substrings of words to align to particular f during training. For example, if "out of" in the previous example are kept together in a clump, the

EM algorithm will have less freedom to give the counts for the occurrences of "out" and "of" to separate formal language words. Incrementing the counts of both of these for the same formal language word (hopefully "from:city"), will decrease the chance of them being spuriously generated later.

More formally, a clumping for a sentence E partitions E into a tuple of clumps C. The number of clumps in C is denoted by $\ell(C)$, and is an integer between $1 \cdots \ell(E)$, inclusive. A particular clump is denoted by c_i, where $i \in \{1 \cdots \ell(C)\}$. The number of words in c_i is denoted by $\ell(c_i)$. c_1 begins at the first word in the sentence, and $c_{\ell(C)}$ ends at the last word in the sentence. The clumps form a proper partition of E. All the words in a clump c must align to the same f. To capture this notion, A is now redefined so that an alignment position in F is specified for each c. Thus, $\ell(A)$ is now $\ell(C)$ instead of $\ell(E)$, and each a_i denotes the formal language node to which each e in c aligns.

Model 1 is a special case of model A in which each word is assumed to be in its own one word clump. Thus, the number of clumps is assumed to be $\ell(E)$. Model 1 has $\ell(F)^{\ell(E)}$ possible alignments, since each of the $\ell(E)$ words has $\ell(F)$ possible alignments. Model A has many more alignments than model 1. First consider the case where the number of clumps, $\ell(C) = 1$. There is only one way to clump $\ell(E)$ words into one substring, namely the whole sentence. This can be aligned to any of the $\ell(F)$ formal language nodes. Hence, there are F ways to align E with F using one clump. For 2

clumps, there are $\begin{pmatrix} \ell(E)-1 \\ 1 \end{pmatrix}$ ways to chose 2 clumps, corresponding to picking one of the $\ell(E)-1$ boundaries between the clumps. Each of these 2 clumps can be aligned to any of $\ell(F)$ formal language nodes. This leads to $\begin{pmatrix} \ell(E)-1 \\ 1 \end{pmatrix} \ell(F)^2$ ways to align E to F using 2 clumps. The number of ways to clump E in $\ell(C)$ clumps, and align them to F is:

$$\begin{pmatrix} \ell(E)-1 \\ \ell(C)-1 \end{pmatrix} \ell(F)^{\ell(C)} \tag{6.1}$$

Using the Binomial Theorem:

$$(1+x)^n = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} x^i \tag{6.2}$$

the sum of these terms using $\ell(C) \in \{1 \cdots \ell(E)\}$ is $\ell(F)(\ell(F)+1)^{\ell(E)-1}$.

By introducing clumps, the modeling becomes better in a variety of ways:

- Rather than modeling the number of English words as a function of the number of formal words, model A allows us to model the number of English clumps for each formal word. Some formal words like "List" can generate very long clumps, whereas other formal words, like "to:city", generate very short clumps. If a particular formal language word likes to generate long clumps, then it is less apt to be able to generate short clumps spuriously.

- If the formal language only contains words for which there is linguistic evidence in the English, then it is reasonable to expect each f to generate at least one clump. Thus, modeling the number of clumps as a function of the size of the formal language should be more reliable than modeling the number of English words. Thus, for example, "out" and "of" may be spuriously generated by different formal language words. But modeling the overall number of clumps might help to reduce this effect.

- The general notion of a clump will allow modeling $p(c_i|f_j)$, the probability that formal language node f_j generates all the words in clump c_i. This will allow more powerful models that can attack the spurious word problem.

- Since model A introduces the general notion of a clump, having model A will facilitate bootstrapping future models that also utilize clumps.

6.2 Formulae

$$p(E \mid F) = \sum_{L=1}^{\ell(E)} p(L \mid \ell(F))\, p(E \mid L, F) \tag{6.3}$$

where L is the total number of clumps for the sentence E, and:

$$p(E \mid L, F) = \sum_{C} p(E, C \mid F, L) \tag{6.4}$$

$$p(E, C \mid F, L) = \sum_{A} p(E, C, A \mid F, L) \tag{6.5}$$

$$p(E, C, A \mid F, L) = \frac{1}{\ell(F)^{\ell(C)}} \prod_{i=1}^{\ell(C)} p(c_i \mid f_{a_i}) \tag{6.6}$$

$$p(c \mid f) = p(\ell(c) \mid f) \prod_{i=1}^{\ell(c)} p(e_i \mid f_c) \tag{6.7}$$

The last formula introduces a change in notation. For all clump based models, e_i denotes the i-th word of clump c, not the i-th word in E.

As in Model 1, the sum and the product can be interchanged, giving:

$$p(E \mid L, F) = \sum_C p(E, C \mid F, L) \tag{6.8}$$

$$p(E, C \mid F, L) = \frac{1}{\ell(F)^{\ell(C)}} \prod_{l=1}^{\ell(C)} q(c_l) \tag{6.9}$$

$$q(c) = \sum_f p(c \mid f) \tag{6.10}$$

This is important because it gives a computationally efficient algorithm for computing $p(E \mid F)$. The $q(C)$ values for all possible clumpings can be calculated in $O(\ell(E)^2\ell(F))$ time if the maximum clump size is unbounded, and in $O(\ell(E)\ell(F))$ if bounded. The Viterbi decoding algorithm[21] can then be used to calculate $p(E \mid L, F)$ from $p(E \mid L-1, F)$ in time $O(\ell(E)^2)$ if the maximum clump size is unbounded, and in $O(\ell(E))$ time if bounded. Since there are $\ell(E)$ possible values for L, the Viterbi or maximum likelihood value of $p(E \mid F)$ can be calculated in $O(\max(\ell(E)^2\ell(F), \ell(E)^3))$ time if the maximum clump size is unbounded, and in $O(\max(\ell(E)\ell(F), \ell(E)^2))$ if bounded. The algorithms to achieve these bounds are discussed in the next section.

There are three types of parameters in model A:

translation probabilities These are the $p(e_i \mid f_c)$ parameters.

clump lengths These are the $p(\ell(c) \mid f)$ parameters.

number of clumps These are the $p(\ell(C) \mid \ell(F))$ parameters.

All of these parameters are hidden (unless the training data are manually clumped and aligned), and the values are trained using the EM algorithm.

A variation of model A, called model AHW, is an elegant enhancement which adds the concept of headwords and nonheadwords to clumps. This model should make major strides at solving the spurious word problem if given enough training data. Headwords will hopefully model semantically relevant words for a formal language word, so semantically meaningless words will have to be generated by nonheadwords. But since each clump is required to generate a headword,[1] this should significantly reduce spurious word errors. Thus, it should be much harder for "of" to be generated spuriously from "equipping", when "of" has a low headword probability. The headword in a multi-word clump is hidden and the EM algorithm will hopefully converge to a solution that learns headwords and nonheadwords well.

In order to implement this model, the translation probabilities are replaced by headword and nonheadword probabilities, $p_{head}(e_i \mid f_c)$ and $p_{nonhead}(e_i \mid f_c)$. Each clump is required to have a headword, and can also have optional

[1] When a clump aligns to different f, as is the case in maximum likelihood scoring, each "f" can select a different word in the clump to be the headword.

nonheadwords. Thus, clumps of length one just contain a single headword, and clumps of length $\ell(c)$ contain one headword and $\ell(c)-1$ nonheadwords. Model AHW replaces equation 6.7 with:

$$p(c \mid f) = \frac{p(\ell(c) \mid f)}{\ell(c)} \sum_{i=1}^{\ell(c)} p_{head}(e_i \mid f_c) \prod_{j \neq i} p_{nonhead}(e_j \mid f_c) \tag{6.11}$$

This headword model formula assumes that the headword can appear in any position within a clump with equal probability (i.e. the $\frac{1}{\ell(c)}$ factor). If desired, one could instead use $p(headpos \mid \ell(c), f)$ to allow the probability to be conditioned upon the length of the clump and the formal language word to which the clump is aligned.

The headword model appears to add an extra factor of $\ell(E)$ into the time complexity of $q(c)$ (if the maximum clump size is unbounded). However, this is not true. The same dynamic programming algorithm to compute $q(c)$ efficiently for model A can be used for model AHW with no increase in time complexity.

Model A and AHW suffer from their inability to model word order. For example, in models A and AHW, the clump containing the words "in the morning" will have the same probability as the word string "in morning the" and "morning the in". This problem can be fixed by modeling the string of words using an N-gram language model. In this thesis, due to limited training data, I consider only bigram models, conditioned by the f to which the clump is aligned. This model is called model ALM. Model ALM does not

model headwords, but for one or two word clumps, a bigram language model prediction is apt to be just as sharp as a headword/nonheadword model, and provide the additional power of knowing which words commonly start and end clumps for a given f. This model is also intended to solve spurious words. The hope is that spurious phrases like "out of" will align to the same "f", and given enough training data, this will align to the proper "f".

Model ALM has parameters $p(e_i \mid e_{i-1}, f)$. The first word in the clump uses a special marker called the *boundary word* (i.e. $p(e_1 \mid boundary, f)$). The last word also uses the boundary word (i.e. $p(boundary \mid e_{\ell(c)}, f)$). Model ALM replaces equation 6.7 with:

$$p(c \mid f) = p(\ell(c) \mid f)p(e_1 \mid boundary, f_c)p(boundary \mid e_{\ell(c)}, f_c)\prod_{i=2}^{\ell(c)} p(e_i \mid e_{i-1}, f_c) \tag{6.12}$$

6.3 Count Derivation

The algorithm to train models A, AHW, and ALM consist of four basic steps:

- Compute the clump probabilities $q(c)$ for all possible clumps.
- Use the Baum-Welch[6] algorithm to compute the counts for each possible clump, since the hidden clumping can be represented by a hidden Markov model.

- Using the chain rule, distribute the clump counts to the model A, AHW, or ALM parameter counts.
- Renormalize the counts of the model parameters.

The last of these steps is trivial. Each of the other three steps is discussed in greater detail below.

6.3.1 Computing the Clump Probabilities

The clump probabilities for model A ,$q(c)$, are calculated by summing the clump probabilities for each formal word f, as shown below:

$$q(c) = \sum_f p(c \mid f) \tag{6.13}$$

$$p(c \mid f) = p(\ell(c) \mid f) \prod_{i=1}^{\ell(c)} p(e_i \mid f_c) \tag{6.14}$$

A simple implementation, which examines each possible clump and computes $q(c)$ using the above formulae, will have complexity $O(\ell(E)^3 \ell(F))$ if there is no bound on the clump size, since there will be $\ell(E)$ clumps of size 1, $\ell(E)-1$ clumps of size 2, $\cdots$, and 1 clump of length $\ell(E)$. This can be improved by a factor of $\ell(E)$, by noticing that a clump of length l beginning at position n can compute the product in equation 6.14 from the product used in the computation of the clump of length $l-1$ (also beginning at position n). First, all the products in equation 6.14 using one term are computed in $O(\ell(E))$ time. Then all the products using two terms are calculated from

these in $O(\ell(E)-1)$ time. Then all the products using three terms are calculated, and so forth. In this manner, all the product terms can be calculated in $O(\ell(E)^2)$ time. If the maximum clump size is bounded, then this requires only $O(\ell(E))$ time. Once all the product terms have been calculated, $q(c)$ can be calculated in either $O(\ell(E)^2\ell(F))$ or $O(\ell(E)\ell(F))$ time depending on whether the maximum clump size is bounded. The algorithm is basically the same for model A or ALM, the only difference is that model ALM has to multiply in the boundary word probabilities, and condition the value for each subsequent English word in a clump on the previous English word. This is trivial to do.

To compute the clump probabilities for model AHW, a similar dynamic programming algorithm can be used, though it is trickier than model A or ALM. Two temporary values are needed for each clump, the product of just the nonhead terms and the value $p(c \mid f)$ in equation 6.11 without the $p(\ell(c) \mid f)$ component. To extend the result from a clump of length $l-1$ to a clump of length l, one can then multiply the product of the nonhead term by the probability that the new word is a head. To this, one adds the adjusted $p(c \mid f)$ term multiplied by the probability that the new word is not a head.

6.3.2 Using the Baum-Welch Algorithm For Clump Models

In models A, AHW, and ALM, the clumping and alignment are hidden. But because of the fortunate interchange of the sum and product terms, the probability of $p(E, C \mid F)$ is simply a product of the $q(c)$ values for the clumps in C, adjusted by a normalizing factor (see equation 6.9). This product can be computed by creating an order-1 hidden Markov model. In this HMM, there is a state for each e in E and one for the end of the sentence, giving a total of $\ell(E) + 1$ states. An arc from state e_i to state e_j $j > i$, means that words $e_i, \cdots, e_{j-1}$ are generated in a single clump. The arc has probability $q(c)$ of being taken, where c is the clump containing words $e_i, \cdots, e_{j-1}$.

Suppose that there is no upper bound on the maximum clump size. Then state e_1 will have $\ell(E) - 1$ arcs, one to each of the other $\ell(E) - 1$ states. State e_2 will have $\ell(E) - 2$ arcs into states $e_3, \cdots, e_{\ell(E)}$, and so forth.

This HMM, with parameters $q(c)$ can be used to run one iteration of the Baum-Welch algorithm[6], which will compute the counts $c(q(c))$. One technical detail is that the Baum-Welch algorithm can only be run for a fixed number of clumps, say L. Thus, it is necessary to run this $\ell(E)$ times, for $L = 1, 2, \cdots, \ell(E)$. This is really computing $c(q(c) \mid L)$. Since the number of clumps for a given $p(L \mid \ell(F))$ is also a hidden variable, it is necessary to use the chain rule to prorate each $c(q(c) \mid L)$ according to the overall score

for $p(E \mid F, L)$:

$$c(q(c)) = \frac{\sum_{L=1}^{\ell(E)} p(E \mid F, L) p(L \mid \ell(F)) c(q(c) \mid L)}{\sum_{L=1}^{\ell(E)} p(E \mid F, L) p(L \mid \ell(F))} \tag{6.15}$$

The Baum-Welch algorithm can calculate $c(q(c) \mid L)$ in time $O(\ell(E)^2 L)$. Since L ranges from 1 to $\ell(E)$ if the maximum clump size is not bounded, the overall training time complexity of using the Baum-Welch algorithm is $O(\ell(E)^4)$. Note that the $p(E \mid F, L)$ terms in equation 6.15 are the forward pass scores of the final state $e_{\ell(E)}$ computed by the Baum-Welch algorithm. For a discussion of how to compute $c(q(c) \mid L)$ for an HMM, see[2, 44].

6.3.3 Count Update Formulae for Models A, AHW, and ALM Using the Chain Rule

Having calculated $c(q(c))$ by normalizing and summing $c(q(c) \mid L)$ for $L = 1, \cdots, \ell(E)$, it is straightforward to apply the chain rule to distribute the counts to the model parameters. The results are stated without derivation. But the derivation would be similar to the one done for model 1.

For model A, the count update formula for the number of clumps parameter for a particular training pair (E, F) is:

$$c(\ell(C) = L \mid \ell(F)) += \frac{p(E \mid F, L) p(\ell(C) = L \mid \ell(F))}{p(E \mid F)} \tag{6.16}$$

The numerator is the probability of generating E from F using $\ell(C) = L$.

This is the product of the forward pass score from the Baum-Welch algorithm $p(E \mid F, L)$ and the probability of generating a clumping of size L given F of size $\ell(F)$. These terms are normalized by the overall probability of generating E from F. Note that while this formula does not depend on any $q(c)$, it still requires the Baum-Welch algorithm to compute $p(E \mid F, L)$ and $p(E \mid F)$.

The count update formulae for the clump length and translation probabilities distribute the value $c(q(c))$ via the chain rule. Any parameter used in the calculation of $q(c)$ will receive a count proportional to its contribution to $q(c)$:

$$c(c \mid f) = c(q(c))\frac{p(c \mid f)}{q(c)} \tag{6.17}$$

The count of a clump given f is just the normalized value of generating this clump from a particular f. This value, is finally given to each parameter:

$$c(\ell(c) \mid f) \quad += \quad c(c \mid f) \tag{6.18}$$

$$c(e \mid f) \quad += \quad c(c \mid f) \tag{6.19}$$

Thus, for each clump c, one distributes the value $c(q(c))$ to all the parameters used in the computation of $q(c)$. There will be $\ell(F)$ contributions to the clump length probabilities, one for each f. There will be $\ell(E)\ell(F)$ contributions to the translation probabilities, one for each (e, f) pair, where $e \in c$ and $f \in F$.

For model AHW, the count update formulae for the number of clumps and clump lengths remain the same. However the translation probabilities

must take into account the headword and nonheadwords. One first calculates $c(c \mid f)$, but using $p(c \mid f)$ and $q(c)$ calculated according to equation 6.11. Then, for each different possible headword $h \in c$, one calculates:

$$c(h \mid c, f) = c(c \mid f) \frac{p_{head}(h \mid f) \prod_{e \neq h} p_{nonhead}(e \mid f)}{\sum_{i=1}^{\ell(c)} p_{head}(e_i \mid f) \prod_{j \neq i} p_{nonhead}(e_j \mid f)} \tag{6.20}$$

Now, the parameter counts can be updated:

$$c_{head}(h \mid f) \quad += \quad c(h \mid c, f) \tag{6.21}$$

$$c_{nonhead}(e \mid f) \quad += \quad c(h \mid c, f) \;\; \forall e \in c, e \neq h \tag{6.22}$$

In order to efficiently implement the count updates, many of these normalizing terms are calculated in the computation of $q(c)$. This is tricky to do for model AHW, but not impossible.

The count update formulae for model ALM are the same as model A, with one minor variation. The count computed for $c(c \mid f)$ is given to each bigram in c, including the two bigrams containing boundary words at the end points.

6.4 Training

Model A, AHW, and ALM were trained using the same set of experiments as in model 1. The initial parameter values for model A are bootstrapped from model 1. The initial parameter values for models AHW and ALM

are bootstrapped from model A. The headword distributions are sharpened by squaring the translation probabilities from model A, and renormalizing. The nonhead distributions for model AHW are set to uniform. For model ALM, the bigram distributions are initialized to just the unigram distribution $p(e \mid f)$. The counts are accumulated for the bigram events though, so subsequent iterations make use of the bigram probabilities.

Since the hand alignments contain just the f to which each e aligns, and neither the clump nor headword information, there are many ways one could use the hand aligned data in training. The results presented here assume that any consecutive English words aligned to the same f are in one clump. This assumption is certainly not valid 100% of the time. For example "Please show which flights $\cdots$" will have "Please show which" aligned to "List". There are many training sentences in which "Please" is a 1-word clump aligned to "List". Likewise, there are sentences beginning "Show flights" and "Which flights". Hence, it might be more reasonable to assume that the clumping is still hidden, and accumulate counts over clumpings consistent with an alignment. This was not done though. To a first approximation, the "maximal" clump assumption works most of the time. The hidden headwords are handled by the EM algorithm. Thus, each time a hand aligned sentence is processed, each headword and nonheadword is given a fractional count based on the current parameter values.

Since models A, AHW, and ALM do not have unique global maxima, it is necessary to get good initial statistics. In order to achieve this, model A

is run for 20 iterations with the translation probabilities fixed. This gives the clump length and number of clump parameters the opportunity to move to better locations in probability space. At this point, since the parameters are reasonably estimated, 5 more iterations of the EM algorithm are run, in which all parameters are allowed to vary.

For models AHW and ALM, the headword/nonheadword and bigram distributions have poor estimates. Thus, for 20 iterations, the clump lengths and number of clumps parameters are held fixed, while the headword/nonheadword or bigram distributions are trained. Then, 5 more iterations are run allowing the clump lengths and number of clumps parameters to vary.

6.5 Smoothing

The model A and AHW clump length and translation probability parameters are smoothed as in model 1. Deleted interpolation is used, allowing a parameter estimate to backoff to an unconditional estimate. Since these cannot be directly observed in the training data, like $p(e)$ can, one has to calculate these values using the training data. For example:

$$p_s(\ell(c) \mid f) = \alpha_{1,f} p(\ell(c) \mid f) + \alpha_{2,f} p(\ell(c)) + \alpha_{3,f} \frac{1}{M} \quad (6.23)$$

$$p(\ell(c)) = \sum_f p(\ell(c) \mid f) p(f) \quad (6.24)$$

where $p(f)$ is computed using the training set, and M is the maximum clump size. Note that the alphas used to smooth the clump lengths are a different set than the alphas used to smooth the translation probabilities.

In model ALM, the clump lengths are smoothed in the same way as models A and AHW. But since model ALM uses conditioning on both f and the previous e in the bigram models, the binning is conditioned upon the count of the pair $(f, e-1)$, and one can back off to $p(e_i \mid f)$ when $p(e_i \mid e_{i-1}, f)$ has low count:

$$p_s(e_i \mid e_{i-1}, f) = \alpha_{1,e_{i-1},f} p(e_i \mid e_{i-1}, f) + \alpha_{2,e_{i-1},f} p(e_i \mid f) + \alpha_{3,e_{i-1},f} p(e_i) + \frac{\alpha_{4,e_{i-1},f}}{\ell(Voc_E)} \tag{6.25}$$

6.6 Results

The most interesting observation from models A, AHW, and ALM is that performance gets *worse* during the final five iterations (see appendix D). Through the first 20 iterations, model A holds the translation probabilities fixed at the model 1 estimates, while the clump lengths and number of clumps parameters are being trained. For model AHW, the clump length parameters are fixed, and the headword/nonheadword probabilities are trained during the first 20 iterations. For model ALM, the clump lengths are fixed, while the bigram probabilities are trained for 20 iterations. From iterations 21 through 25, all model A parameters are re-estimated. Models AHW and

ALM fixed the headword/nonheadword or language model probabilities, and re-estimated only the clump lengths and number of clumps parameters during the final five iterations. While the likelihood on the training set is increasing, this does not generalize to the test sets. It is hard to say why. I don't believe that the EM algorithm is over training. Two possible explanations are:

- The models themselves might not be sufficiently rich to accurately model the domain. So as mentioned in [14], MLE is not well suited for these models.

- The models could be undertrained. When all parameters are allowed to vary, the EM algorithm finds a way to maximize the likelihood on the training data that is "wrong". Some evidence of this is provided by using hand aligned training data, as this phenomenon does not occur. The hand aligned data keeps the EM algorithm in check. But, if there were more training data, then the EM algorithm would not be able to find obtuse ways to make the likelihood better.

The results presented here use the smoothed, 20 iteration parameter sets using maximum likelihood decoding for models A, AHW, and ALM. The unsmoothed parameter sets for 20 and 25 iterations are presented in appendix D for each model. In appendix D, the reader can see the effect of allowing a new set of model parameters to vary for the final five iterations.

6.6.1 Exact Match Maximum Likelihood Results for DEV94

The following tables contain the exact match results using maximum likelihood decoding for smoothed models A, AHW, and ALM. The unsmoothed results (shown in appendix D) show the following:

- Models A, AHW and ALM are helped significantly by hand training data.
- Model A almost always does better than model ALM. The probability matrix $p(e_i \mid f, e_{i-1})$ is very sparse for model ALM, and hence does not do well on test data unless smoothed.
- When all the hidden and no hand training data are used, Model A is nearly at the limit of its performance. In moving from 2813 to 5627 hidden training sentences, Model A only got 4 more DEV94 sentences right. Model AHW improved by 28 sentences and model ALM improved by 16. It is clear that models AHW and ALM can benefit from more training data.

The smoothed results shown for models A, AHW, and ALM are better than their unsmoothed counterparts. The smoothed results shown in tables 6.2-6.4 also show virtually no difference between these models when using just hidden data (the last column). They all show that the results can be further improved by more training data. But this is most evident

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	283	281	276	274	273	275	272	275	273
2813	285	284	275	275	273	271	269	264	261
1406	283	281	272	274	260	250	245	252	244
703	281	276	267	270	260	248	241	244	235
351	281	282	267	264	251	224	218	218	192
175	282	275	261	263	249	226	204	202	183
87	282	279	258	264	246	202	187	179	155
43	280	282	266	270	254	211	204	194	134
0	279	280	265	263	240	206	173	173	153

Table 6.2: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model A as a Function of Amount of Hand and Hidden Training Data

for models AHW and ALM, where the unsmoothed results show dramatic increases in performance as the number of hidden sentences is doubled from 2813 to 5627. In comparison to model 1, the results are 1-1.5% better (in terms of absolute error rate) when all the hidden and no hand training data are used. But model 1 shows that it is near its performance limit, as the accuracy improved by less than 1% when the number of hidden sentences was doubled from 2813 to 5627. Thus, while models A, AHW, and ALM appear only minimally better than model 1, and appear virtually identical amongst themselves, models AHW and ALM clearly have more potential for improvement if given more training data. Until the data are collected and the experiments run, the full power of these models will not be known.

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	287	283	283	278	277	277	277	274	273
2813	288	285	285	272	270	279	278	262	266
1406	284	282	269	269	254	262	246	248	254
703	277	277	265	279	270	245	260	244	250
351	278	281	272	275	254	226	222	226	167
175	276	276	274	265	254	252	232	211	188
87	274	276	263	273	256	217	201	192	157
43	279	277	266	269	252	211	221	178	129
0	278	278	264	265	253	206	192	168	153

Table 6.3: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model AHW as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	288	284	290	280	276	279	282	273	273
2813	287	276	279	270	270	277	275	278	263
1406	285	283	273	271	263	252	257	252	251
703	286	280	273	272	260	252	242	244	236
351	286	283	273	257	251	221	209	220	149
175	282	282	269	263	249	216	203	207	189
87	287	282	273	263	247	209	201	173	156
43	283	279	270	268	253	198	198	153	131
0	285	280	269	261	246	196	172	166	50

Table 6.4: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model ALM as a Function of Amount of Hand and Hidden Training Data

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
A	5.63	5.45	5.50
AHW	5.33	5.14	5.27
ALM	5.32	4.93	5.06

Table 6.5: Cross Entropy Results for DEV94 Using Model A Variants and Different Amounts of Hand and Hidden Training Data

6.6.2 Cross Entropy Results for DEV94

The cross entropy results are shown in table 6.5 using the smoothed models, for the most important experiments.

The good results of models AHW and ALM using hand aligned training data are also reflected in the cross entropy. While not guaranteed to be true, they are true in these cases.

6.6.3 Viterbi Percentage Results for DEV94

The viterbi percentages for models 1, A, AHW, and ALM are shown in table 6.6. As expected, model A does better than model 1, and either model AHW or ALM does better than model A. The Viterbi percentages shown are calculated using unsmoothed parameter sets.

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	73.5	80.8	87.5
A	81.0	90.2	92.7
AHW	89.4	93.7	94.7
ALM	86.0	94.4	95.3

Table 6.6: Viterbi Percentages of Maximum Likelihood Cross Entropy for DEV94 For Various Models

Chapter 7

Clumping Models With Fertilities

7.1 Introduction

The basic clumping model A improves over model 1 in that it models the English sentence by substrings of words called clumps. By selected a formal language that contains approximately the semantic concepts mentioned in the English, one expects that the number of clumps and clump length parameters of model A will model the length of the English better than the sentence length parameters of model 1. Using headwords or language models helps to solve spurious word problems. This modeling is not intuitively satisfying in that modeling $p(\ell(C)|F)$ by $p(\ell(C)|\ell(F))$ totally ignores the identify of the formal words in F. It is easy to see that "from:city" usually generates one

clump, where "List" frequently generates 2 or 3 clumps. One should model the number of clumps in the English by conditioning on the identity of the formal language words. The number of clumps generated by a formal word is called its *fertility*. This is an extension of the word fertility used in machine translation[13] to clump models.

The hope is that this will better model the number of clumps needed in the English, and spurious word problems associated with breaking up clumps will be reduced. One way this happens is that introducing formal language words will require multiplying the likelihood by their fertility in the English. Thus, the likelihood is reduced each time a formal language word is added that does not generate a clump with greater probability than some other formal language word. For example, to spuriously add "aircraft equipping" to a flight query, the fertilities of both "aircraft" and "equipping" are now multiplied into the likelihood. If these do not generate words with great enough probability to offset the fertility, then the model will not insert them. The hope is that fertilities will be strong enough to compensate for weakly generated spurious words.

This thesis describes two different ways to model fertility. These two ways can be used with any of the three ways of generating the translation probabilities (directly, with headwords/nonheadwords, or with a bigram language model). This leads to 6 different fertility models. The general fertility models, called models C, CHW, and CLM, allow an arbitrary fertility distribution $p(n \mid f)$ for the number of clumps n generated by formal word f.

Notationally, the fertility of f_i is the number of clumps in C that are aligned to f_i in A. That is, the fertility of f_i is n if i appears n times in A.

There is one problem in implementing models C, CHW, and CLM however. In models 1 and A, AHW, and ALM, the summation over all A of $p(E, A \mid F)$ was exchangeable with the product used to compute $p(E, A \mid F)$. This converted a seemingly exponential problem into a polynomial one. In models C, CHW, and CLM this exchange can not be done mathematically. The maximum likelihood training and decoding appear to be exponential. All hope is not lost though. If one assumes that the fertility n of f is a Poisson process with parameter λ_f:

$$p(n \mid f) = \frac{e^{-\lambda_f} {\lambda_f}^n}{n!} \tag{7.1}$$

then a polynomial time algorithm exists. This will become obvious when the formulae are given in the next section. The fertility models assuming a Poisson distribution are called models B, BHW, and BLM.

One can still use models C, CHW, and CLM, but it is not possible to examine every possible alignment. Hence, one can use any of the previous models to produce a set of "candidate" alignments. These alignments are then rescored using the model C, CHW, or CLM parameters, and the counts accumulated for the EM algorithm. In particular, one can use the top-N Viterbi algorithm[74] to guarantee that the best N alignments according to the candidate model are used in model C. If the model has a high Viterbi per-

centage (see section 5.6.3), then hopefully the millions of alignments omitted will not contain any appreciable amount of probability. Note that in order for the EM algorithm to produce an MLE estimate, the same candidate alignments must be used from one iteration to the next. Thus, this top-N alignment can be done just once, and then used repeatedly for each iteration of the EM algorithm.

7.2 Formulae

In this section, I first give the formulae for the general fertility model C. I then describe why the "trick" of exchanging a summation and product cannot be done. While another factorization might exist that can lead to a polynomial algorithm, I have not found it. Next, the formulae for model B are given, attained by applying the Poisson assumption to the general fertility term of model C. The simplified formulae reveal that the polynomial time algorithms exist to perform the maximum likelihood estimation. I do **not** give the formulae for models BHW, BLM, CHW, and CLM. These are straightforward extensions of the models B and C, and are attained analogously to how models AHW and ALM are derived from model A (see section 6.2).

The formulas for this model are:

$$p(E \mid F) = \sum_{C,A} p(E, C, A \mid F) \tag{7.2}$$

$$p(E, C, A \mid F) = \frac{1}{L!} \prod_{i=1}^{\ell(F)} p(n_i \mid f_i) n_i! \prod_{j=1}^{\ell(C)} p(c_j \mid f_{a_j}) \tag{7.3}$$

$$p(c \mid f) = p(\ell(c) \mid f) \prod_{i=1}^{\ell(c)} p(e_i \mid f_c) \tag{7.4}$$

where $p(n_i \mid f_i)$ is the fertility probability of generating n_i clumps by formal word f_i. Note that $\sum n_i = L$. The $L!$ denominator is the number of ways of arranging L clumps. The $n_i!$ terms in the numerator are the number of ways of arranging the n_i clumps aligned to an f_i. It is not possible to exchange the sum and products, because each alignment has a different fertility term $\prod_{i=1}^{\ell(F)} p(n \mid f_i) n!$. In model A, this term is simply a constant, and can be factored out of the summation. For the exchange to be valid, $p(E, C, A \mid F)$ must be expressible as a constant times $\prod_{j=1}^{\ell(C)} p(c_j \mid f_{a_i})$.

It appears that the computation of the likelihood for model C, the sum of $\ell(F)(\ell(F)+1)^{\ell(E)-1}$ product terms, is exponential. Dynamic programming can perform some of the factoring, but because there are an exponentially large number of different fertility terms (regardless of the clump probability term), factoring by the fertility terms still leads to an exponential number of addends. The only hope to reduce the computation to polynomial time would be to factor according to the clump length and translation probabilities. I have not discovered how to do this factoring.

If one assumes that the fertility is modeled by a Poisson distribution, then a polynomial time algorithm exists. The Poisson assumption is motivated by the intuition that most formal language words have a specific number

of clumps that they like to generate. Generating more or less clumps has lower probability. The Poisson distribution is one of an infinite number of unimodal distributions that has this property. The selection of a Poisson distribution was done for its mathematical properties, and has worked well in practice. While many physical processes are provably Poisson[51, 37], the underlying assumptions about fixed numbers of events occurring in a limited time or space do not apply here.

To show that polynomial time algorithms still exist, one substitutes equation 7.1 into equation 7.3:

$$
\begin{aligned}
p(E, C, A \mid F) &= \frac{1}{L!} \prod_{i=1}^{\ell(F)} \frac{e^{-\lambda_f} {\lambda_f}^{n_i}}{n_i!} n_i! \prod_{j=1}^{\ell(C)} p(c_j \mid f_{a_i}) && (7.5)\\
&= \frac{1}{L!} \prod_{i=1}^{\ell(F)} e^{-\lambda_f} {\lambda_f}^{n_i} \prod_{j=1}^{\ell(C)} p(c_j \mid f_{a_i}) && (7.6)\\
&= \frac{1}{L!} \prod_{i=1}^{\ell(F)} e^{-\lambda_f} \prod_{j=1}^{\ell(C)} q(c_j \mid f_{a_i}) && (7.7)\\
q(c \mid f) &= \lambda_f p(c \mid f) && (7.8)
\end{aligned}
$$

The last simplification moves the ${\lambda_f}^n$ term into the $q(c \mid f)$ term. This is because ${\lambda_f}^n$ means that in the alignment, f has n clumps aligned to it. This means that in the product term of each $\ell(C)$ clumps, exactly n of these will be aligned to f. Hence, one can multiply the $p(c \mid f)$ term by λ_f. This will then add a factor of ${\lambda_f}^n$ into the product of the clumps. Now it should be obvious that the summation and product can be exchanged. The first

component of equation 7.7:

$$\frac{1}{L!}\prod_{i=1}^{\ell(F)} e^{-\lambda_f} \tag{7.9}$$

is a constant for each fixed L, since the product term uses ALL λ_f, regardless of the number of clumps aligned to f. Hence, this can be factored out, and the polynomial time expressions given:

$$\begin{aligned}
p(E \mid F) &= \sum_L p(E \mid F, L)p(L \mid F) && (7.10)\\
p(L \mid F) &= \frac{e^{-\sum \lambda_f}(\sum \lambda_f)^L}{L!} && (7.11)\\
p(E \mid L, F) &= \sum_C p(E, C \mid F, L) && (7.12)\\
p(E, C \mid F, L) &= \frac{1}{(\sum \lambda_f)^L}\prod_{l=1}^{\ell(C)} q(c_l \mid F) && (7.13)\\
q(c \mid F) &= \sum_f q(c \mid f) && (7.14)
\end{aligned}$$

This is polynomial time in the same way that model A was. The $q(C)$ values for all possible clumpings can be calculated in $O(\ell(E)^2\ell(F))$ time if the maximum clump size is unbounded, and in $O(\ell(E)\ell(F))$ if bounded. The Viterbi decoding algorithm[21] is then used to calculate $p(E \mid L, F)$. The score produced by the Viterbi algorithm, which is the sum over all possible clumpings for a fixed L, is then normalized by the $\frac{1}{L!}\prod_{i=1}^{\ell(F)} e^{-\lambda_{f_i}}$ constant.

In model B, the number of clumps produced by each f is a Poisson process. Under this assumption, one can mathematically show that the total number of clumps for a sentence is also a Poisson process, with parameter $\sum_f \lambda_f$,

where the summation is taken over the $f \in F$. Thus, for any fixed F, if you consider all possible clumpings of all possible E, the number of clumps will also be a Poisson process. This is shown in equation 7.11.

There are three types of parameters in either models B or C:

translation probabilities These are the $p(e_i \mid f)$ parameters. For the headword variants, these are replaced with two distributions, the headword and nonheadword probabilities, $p_{head}(e_i \mid f)$ and $p_{nonhead}(e_i \mid f)$. For the language model variants, these are replaced with bigram language models $p(e_i \mid e_{i-1}, f)$.

clump lengths These are the $p(\ell(c) \mid f)$ parameters.

fertilities These are the $p(n \mid f)$ parameters for model C, or the λ_f parameter for model B.

7.3 Count Derivation

The count derivation for model C uses the following steps, since it is not possible to train model C using all possible hidden alignments:

1. For each sentence in the training corpus, use one of the previous models and the top-N Viterbi algorithm to compute the 100 most likely alignments. Since the hidden clumping model does not know the proper number of clumps, the top-N algorithm actually produces the 100 most likely alignments for $1, 2, \cdots, \ell(E)$ clumps. Thus, one gets $100\ell(E)$

alignments.[1] From these $100\ell(E)$ alignments, the best 1000 are selected.

2. For each alignment, rescore the alignment with the current model C parameters, and accumulate the scores in a sum.

3. Create a probability distribution for the alignments by normalizing each alignment by the total score of all the alignments.

4. For each alignment, give to each parameter used in the alignment a count equal to the normalized probability of this alignment.

5. Recompute the model C parameters after the counts have been accumulated for the whole training corpus.

6. Iterate.

The count update formulae for model B are the same as model A for the clump lengths and translation probabilities. The only question is how does one update the counts for the Poisson parameter λ_f. Since each λ_f is used in the same place that the clump length is (see equation 7.8), the count λ_f receives is the same that the clumps lengths do. To compute the MLE of a Poisson process:

$$p(n) = \frac{e^{-\lambda}\lambda^n}{n!} \tag{7.15}$$

[1] If the maximum clump size is fixed, then alignments utilizing few numbers of clumps might not exist.

suppose one has a sequence of observations $n_1, \cdots n_k$, which gives rise to an empirical probability estimate $\tilde{p}(n)$. As before, one needs to maximize:

$$\begin{aligned} \frac{1}{N} \log \mathcal{L} &= \sum_n \tilde{p}(n) \log p(n) && (7.16) \\ &= \sum_n \tilde{p}(n) \log \frac{e^{-\lambda} \lambda^n}{n!} && (7.17) \\ &= \sum_n \tilde{p}(n)((n \log \lambda - \lambda) - \log n!) && (7.18) \end{aligned}$$

where N is the total number of observations n that occurred.

To maximize with respect to λ, take the partial derivative with respect to λ and set to 0:

$$\begin{aligned} \sum_n \tilde{p}(n)(\frac{n}{\lambda} - 1) &= 0 && (7.19) \\ \sum_n \tilde{p}(n)(n - \lambda) &= 0 && (7.20) \\ \sum_n \tilde{p}(n) n &= \lambda \sum_n \tilde{p}(n) && (7.21) \\ \lambda &= \sum_n \tilde{p}(n) n && (7.22) \end{aligned}$$

Thus, to maximize the likelihood of a Poisson process to produce the observed data, one weighs the observed events by their empirical probability, and sets this to the Poisson parameter λ.

Note that equation 7.22 can be rewritten as:

$$\lambda = \frac{1}{N} \sum_n \tilde{c}(n) n \qquad (7.23)$$

The value $\tilde{c}(n)n$ is the count that a parameter gets, weighted by the number of times the event occurred in the sentence. This is exactly the accumulated counts that the EM algorithm produces. Thus, to maximize the likelihood for the Poisson parameters λ_f in models B, BHW, and BLM, one divides the total count accumulated for λ_f by the number of times f appears in the training corpus.

7.4 Training

The initial parameter values for models B and C are bootstrapped from model A. The initial values for the Poisson parameters in model B are set to 1. The model C fertility parameters are trained by running one iteration of model B, and accumulating counts for the general fertility distributions.

The initial parameter values for models BHW, BLM, CHW, and CLM are bootstrapped from models B or C. Note that one could also use AHW or ALM instead. The decision to bootstrap from B or C was done arbitrarily. The headword distributions are sharpened by squaring the translation probabilities from models B or C, and renormalizing. The nonhead distributions for models BHW and CHW are set to uniform. For models BLM and CLM, the bigram distributions are initialized to to just the unigram distribution $p(e \mid f)$. The counts are accumulated for the bigram events though, so subsequent iterations make use of the bigram probabilities.

As in model AHW, the headword and nonheadwords are not known, so

this hidden event is handled by the EM algorithm. Thus, each time a hand aligned sentence is processed, each headword and nonheadword is given a fractional count based on the current parameter values.

As in models A, AHW, and ALM, the model B and C variants do not have unique global maxima. Thus, in order to get good initial parameter estimates, models B and C are first run with the translation probabilities fixed. For models B and C, 20 iterations and 10 iterations are run respectively. This gives the clump lengths and fertilities or Poisson parameters the opportunity to move to better locations. At this point, since the parameters are reasonably estimated, only 5 more iterations of the EM algorithm are run, in which all parameters are allowed to vary.

For models BHW, CHW, BLM, and CLM the headword/nonheadword and bigram distributions have poor estimates. Thus, for either 20 iterations (models BHW and BLM) or 10 iterations (model CHW and CLM), the clump lengths and fertilities or Poisson parameters are held fixed, while the headword/nonheadword or bigram distributions are trained. Then, 5 more iterations are run allowing the clump lengths and fertilities or Poisson parameters to vary.

Model C actually takes a long time to train, because the Viterbi top-N implementation of the Baum-Welch algorithm runs approximately $100 \log 100$ times slower the standard Baum-Welch algorithm. This is why only 10 iterations where run instead of 20. In addition, it would be too expensive to run all 81 experiments corresponding to using varied amounts of hand and

hidden training data. Instead, only the 3 relevant "corners" where run in which all or none of each of the hand aligned training data or the hidden training data were used.

7.5 Smoothing

Except for the Poisson and fertility parameters, all the other parameters are same as in model A, and are smoothed the same way. The Poisson parameters are not smoothed, since these are well estimated. The model C fertilities are smoothed in the standard way using deleted interpolation.

7.6 Results

The unsmoothed results for models B, BHW, and BLM also show no improvement during the final 5 iterations of training. Thus, it is sufficient to train the clump lengths and Poisson fertilities in model B for 20 iterations, keeping the translation probabilities (initialized from model A) fixed. These are used to initialize the clump lengths and Poisson fertilities for models BHW and BLM. 20 iterations of re-estimating the headword/nonheadword or language model parameters for models BHW and BLM are sufficient.

Comparing the model B results to model A results, one discovers that the results are generally 1-2% better. The impact appears to be greatest when there are few training data, either hand or hidden. When a lot of

hand training data are used, model B is 2% better. As for models BHW and BLM, Poisson fertility generally helps most of the experiments. Poisson fertility actually hurts the experiments which use a lot of hidden data and little hand data when compared to models AHW and ALM.

While it is encouraging that Poisson fertility improves model A results, it is discouraging that it hurts model AHW and ALM results (when there is mostly just hidden data, which is the goal of this thesis). This is probably due to a poor training strategy. Models BHW and BLM use the model B Poisson parameters, and retrain the headword/nonheadword and language model translation probabilities for 20 iterations. Perhaps it would have been better to initialize these from model AHW and ALM parameters instead, and train the fertilities.

7.6.1 Exact Match Maximum Likelihood Results for DEV94

The following tables contain the exact match results using maximum likelihood decoding for smoothed models B, BHW, and BLM.

As in the model A variants, hand data always help models B, BHW, and BLM. Smoothed model BLM show a vast improvement in accuracy as the training data are doubled from 2813 hidden sentences to 5627 hidden sentences. Again, this shows that more training data are needed to train model BLM. Models B and BHW can also be improved by more hidden

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	290	285	282	280	279	279	276	275	274
2813	286	281	280	281	279	279	270	270	267
1406	287	288	278	278	265	258	251	257	247
703	290	285	273	280	270	251	245	242	239
351	290	286	271	268	257	237	206	224	210
175	291	287	274	271	268	246	218	225	209
87	291	287	272	278	263	243	208	210	194
43	289	287	273	275	261	235	223	227	186
0	289	288	273	273	257	239	211	209	148

Table 7.1: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model B as a Function of Amount of Hand and Hidden Training Data

training data. Thus, it is premature to draw conclusions about the the value of fertility and utility of hand aligned training data until more hidden training data are used and the performance peaks asymptotically.

Table 7.4 gives the results for the "corner" experiments for models C, CHW, and CLM. A comparison to the model B, BHW, and BLM results shows that model CHW is better than BHW by about 4% if only hidden training data are used, 1% if only hand training data are used, and the same if trained with both hand and hidden training data. Models C and CLM improve upon B and BLM by about 1% only if the hand data are used, and they degrade by 4% if no hand data are used. Unfortunately, due to lack of resources, the experiments were not run to determine the minimum amount of hand aligned training data needed to guarantee an improvement by the models C and CLM over B and BLM. But it is encouraging that model CHW

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	295	286	282	279	278	273	272	269	269
2813	294	291	288	280	273	270	270	273	261
1406	291	290	283	280	255	251	244	253	248
703	289	287	271	278	269	256	248	242	240
351	286	288	271	271	259	232	221	216	175
175	291	288	275	273	267	239	210	217	159
87	288	289	274	282	262	238	209	202	185
43	291	287	275	275	267	239	231	223	179
0	292	289	275	278	265	241	220	208	148

Table 7.2: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model BHW as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	290	287	279	280	279	275	276	273	274
2813	292	282	282	278	281	272	266	270	255
1406	291	285	281	279	255	251	252	254	244
703	291	283	281	272	262	234	240	229	237
351	292	287	279	272	256	239	209	225	165
175	291	283	278	272	260	232	195	215	182
87	290	283	280	275	251	243	196	205	178
43	291	284	279	275	262	231	209	192	159
0	289	284	280	271	262	236	197	200	58

Table 7.3: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model BLM as a Function of Amount of Hand and Hidden Training Data

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
C	262	290	293
CHW	286	296	297
CLM	258	291	293

Table 7.4: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smooth Models C, CHW, and CLM as a Function of Amount of Hand and Hidden Training Data

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
A	5.63	5.45	5.50
AHW	5.33	5.14	5.27
ALM	5.32	4.93	5.06
B	5.56	5.49	5.52
BHW	5.29	5.19	5.32
BLM	5.25	4.98	5.11
C	5.30	4.93	4.99
CHW	4.81	4.65	4.74
CLM	4.96	4.47	4.58

Table 7.5: Cross Entropy Results for DEV94 For Various Models

showed 4% improvement with no hand aligned training data.

7.6.2 Cross Entropy Results for DEV94

The cross entropy results are shown in table 7.5 for the smoothed A and B models, for the most important experiments.

These entropies are nearly identical to the corresponding model A entropies. Many of the model C variants show an improvement in half a bit over the model B variants.

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	73.5	80.8	87.5
A	81.0	90.2	92.7
AHW	87.0	93.0	93.5
ALM	86.0	94.4	95.3
B	80.0	89.4	91.7
BHW	85.8	92.8	93.3
BLM	85.5	94.2	95.3
C	90.9	93.7	94.8
CHW	90.5	93.9	93.7
CLM	95.5	96.7	96.9

Table 7.6: Viterbi Percentages of Maximum Likelihood Cross Entropy for DEV94 For Various Unsmoothed Models

7.6.3 Viterbi Percentage Results for DEV94

The Viterbi percentages for models A and B are shown in table 7.6. The Viterbi percentages shown are calculated using unsmoothed parameter sets. As can be seen, the results nearly one percent worse for the model B variants. The model C, CHW, and CLM Viterbi percentages are much better than model B. Though this should come as no surprise. Models B, BHW, and BLM are trained to their maximum likelihood estimate using all possible alignments, whereas models C, CHW, and CLM use only the most likely alignments to compute the statistics. Hence, one would expect a much better Viterbi percentage from models C, CHW, and CLM.

Chapter 8

Discussion

So far, 10 different models have been presented. In the interest of brevity, only smoothed model B results are discussed in this chapter. The purpose is to give the reader insight as to the kinds of errors made by this statistical approach. Obviously each model will have its own strengths and weaknesses.

Recall the smoothed model B results presented in table 7.1, 290 sentences exactly matched the reference answer out of 410 for the experiment which used all the hand and hidden training data. When nothing but hidden data were used, the accuracy dropped to 274. But the last column of this table shows that the accuracy is still getting better as more hidden data are added. Thus, one could reasonably expect better results if more training data were available.

Also recall that 70 of the 410 sentences do not have the correct answer in the pattern matcher vocabulary (see section 4.4). Thus, without augmenting

Rank	Count
1	290
2	22
3	8
4	1
5	1
8	2
12	2
13	1
17	1
22	1
24	1
232	1
1220	1
1407	1
1901	1
3832	1
3833	3
3838	1
3882	1
total > 1	50

Table 8.1: Histogram of Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model B with all the Hand and Hidden Training Data

the pattern matcher vocabulary to include these 70 missing patterns, there is no way to know whether or not the statistical translation models would have worked. Thus, there are only 50 errors that can be analyzed from the experiment using all hand and hidden training data, and 66 errors from the experiment using just hidden data.

Histograms of the exact match results are given for these experiments in tables 8.1 and 8.2.

The numbers in the tables give the rank the reference answer receives

Rank	**Count**
1	274
2	25
3	13
4	7
5	2
6	1
7	4
8	1
9	1
10	1
16	1
65	1
232	1
1220	1
1407	1
1901	1
3832	1
3833	3
3838	1
total > 1	66
N/A	70

Table 8.2: Histogram of Maximum Likelihood Decoding Results for DEV94 Using Smoothed Model B with just Hidden Training Data

when the pattern matcher scores all 3882 patterns using $p(E \mid F)p(F)$. A rank of 1 means that the correct answer was found. A rank of 2 meant that one incorrect answer was found to score better than the correct answer. While the experiment using hand data has a better exact match result, the top 5 results are virtually identical. Thus, using nothing but hidden data, one loses 16 correct answers, but in total, these are shifted to "near misses". The top 5 exact match accuracy is 78%.

64 of 66 errors from the experiment using smoothed model B trained on just hidden training data are analyzed.[1] The following categories are used.

- **Translation Model Error**

  This is an error due to the smoothed model B parameters. These errors are analyzed further in the next section.

- **Language Model Error**

  This is an error due to a poor language model estimate. The translation model actually prefers the reference answer, but the pattern matcher finds an incorrect answer that has a higher score due to a better language model estimate.

- **Bad Formal Design**

  This is an error due to a poor design of the formal language NL-Parse. With a formal language better suited to the translation models, this

[1] Due to a programming bug, 2 of the errors can not be analyzed.

type of error might not occur.

- **Permutation Error** This is when the correct answer and the incorrect pattern matcher answer contain the same words, but in a different order (and hence a different meaning). Unless the translation model includes parameters to disambiguate these cases, the statistical NLU system will have to rely on the language model to predict the correct answer. This is intellectually unsatisfying, as the English is unambiguous. This is an error inherent in the modeling approach.

- **Bad Class** This error occurs if a sentence is context dependent, and is accidentally misclassified into the class A test set.

- **Tagger Error** This error occurs if the tagger generates a bad tag in the English. Since the pattern matcher only tries patterns that contain the exact same set of tags, a tagger error will necessarily cause an error.

- **Bad City/Airport Tag** The English tagger was trained to use ARP tags for airports that are mentioned by their city. For example, "the airport in Indianapolis" becomes an ARP as opposed to "the airport in CITY". The DEV94 test set includes numerous instances of English like this, and the reference answer uses CITY instead of ARP. The tagger produces a wrong tag and the sentence is not decoded correctly. This is not an error by the tagger, but rather the result of a bad design decision made several years ago.

Error Type	Count
Translation Model Error	28
Language Model Error	16
Permutation Error	8
Bad Formal Design	5
Bad City/Airport Tag	4
Tagger Error	2
Bad Class	1

Table 8.3: Errors Made by Pattern Matcher in DEV94 Using Smoothed Model B with just Hidden Training Data

The distribution of the 64 errors into these classes is shown in table 8.3.

Since the translation model is the primary focus of this thesis, the translation model errors are discussed in greater depth in the next section. The other errors are described in appendix E. In section 8.2 some strategies for reducing the error rate are discussed.

8.1 Errors Due to the Translation Model

In the error analysis presented in this section, smoothed model B parameters are used in a maximum likelihood decoding. This makes it hard to know why a particular formal language word is included. One would have to examine all possible alignments. Instead, one can use the Viterbi alignment to find the most likely formal language word to generate each English word. While this may not be the cause of the error, it often gives insight as to why a wrong pattern was selected.

In the translation model error list below, each error occurred just once,

unless a count is mentioned. The errors are loosely categorized according to why the error occurred. The prefix "E:" means "English", "A:" means the correct answer, and "P:" means the pattern matcher answer.

8.1.1 Substitution Errors

In a substitution error, the correct pattern differs from the wrong pattern found by the pattern matcher in that one formal language word is replaced by another.

E: what is the cheapest one-way flight from CITY_1 to CITY_2

A: List flights cheapest one_way from:city CITY_1 to:city CITY_2

P: List fares cheapest one_way from:city CITY_1 to:city CITY_2

The scores are very close, the wrong answer has score 7.033×10^{-9}, while the correct answer has score 7.004×10^{-9}. In both cases, one_way generates "one-way flight" and flight and fare generate nothing. Thus, the selection between flight and fare is left to the LM and Poisson fertility. While it does not seem likely, there are many cases in which "one_way" should generate the word "flight". For example, in "how much does the one way flight cost", the formal language word "fare" generates "cost", and it is reasonable to have "one_way" generate "one way flight". Flight/fare confusion happened in four sentences.

E: what is the least expensive flight from CITY_1 to CITY_2 one-way

A: List flights cheapest one_way from:city CITY_1 to:city CITY_2

P: List flights cheapest one_direction from:city CITY_1 to:city CITY_2

The language model likes one_direction a lot more than one-way, as this is the default formal language if someone requests the cheapest flight without requesting the type fare desired. The translation model score is also better for the wrong answer, (1.17×10^{-17} vs 1.10×10^{-17}). This is probably a case that would not be an error according to the CAS metric. In a similar sentence, "cheapest one-direction" substituted for "along-with flights" so that "one-direction" could generate "is" spuriously.

E: list all AIR_1 flights into CITY_1

A: List flights AIR_1 to:city CITY_1

P: List flights AIR_1 from:city CITY_1

In both answers, "into" is generated from "flights", and both "from:city" and "to:city" generate nothing. Thus, the language model picks the more likely pattern. The word "into" appears 48 times out of 58178 words in the training corpus. $p(into \mid to : city) = 3.55 \times 10^{-5}$ and $p(into \mid flights) = .0016$. The most common usage for "into" is in queries like "i want to arrive into CITY_1 by $\cdots$".

E: i want to see all flights arriving and departing ARP_1

A: List flights or from:airport ARP_1 to:airport ARP_1

P: List flights from:airports airports to:airport ARP_1

This error occurred twice, and contains two substitution errors. In the wrong answer, "from:airports" spurious generates "see all"; $p(see|List) = .00046$ and $p(all|List) = .0156$ for the correct answer, but $p(see|from : airports) = .073$ and $p(all|from : airports) = .079$ for the wrong answer. In fact, "from:airport" generates "departing" with greater probability than "to:airport", which generates "departing" in the wrong answer. Yet this win from using "from:airport" is not offset by the spurious inclusion of "from:airports airports". It should also be pointed out that this sentence is an example in which there are two formal words for the same English word. One formal word will generate the English word, and the other will generate nothing. Since this was a tag, and only formal language patterns that contain the exact same tags as the English are attempted by the pattern matcher, this does not lead to an error.

E: i need an AIR_1 flight number from CITY_1 to CITY_2 departing at about TIME_1

A: List Extract flights AIR_1 departing around TIME_1 from:city CITY_1 to:city CITY_2 Features all flight_number

P: List Extract flights AIR_1 departing at TIME_1 from:city CITY_1 to:city CITY_2 Features all flight_number

In two sentences, "at about" is generated from "at" instead of "around": This is because $p(at \mid at) = .38$ and $p(about \mid flight) = .004$, yet $p(at \mid around) =$

.044 and $p(around \mid around) = .035$. When hand aligned training data are used, $p(at \mid around) = .064$ and $p(around \mid around) = .062$, and this error does not occur. But because "about" can be spuriously generated by "flights", and since $p(at \mid at)$ is so sharp, the wrong answer is found.

8.1.2 Insertion Errors

In an insertion error, the incorrect pattern has additional formal language words besides all the words in the correct answer. These inserted words lower the likelihood in terms of their fertilities and factorials. But if one of these words has a much higher probability of generating some of the English words, then this can compensate. For infrequent English words, the probability of generating them from formal language words that are used frequently, but with other English expressions, will be low. But if there is a formal language word in the training sentence that appears less frequently, the EM algorithm can improve the likelihood by giving the counts for generating this rare English word to the less likely formal language word. The more common formal language words generate the more common English words. But should a less likely word appear in the test set, then wrong formal language is introduced just to generate a rare word. This is called the *spurious word* problem. The headword variants should help solve this problem if properly trained with sufficient data.

E: i need a flight from CITY_1 to CITY_2 that stops in CITY_3 what flight should i take

A: List flights from:city CITY_1 stopping-in:city CITY_3 to:city CITY_2

P: List flights from:city CITY_1 serving:meal lunch stopping-in:city CITY_3 to:city CITY_2

The language model for the reference answer is 30 times more likely than the wrong answer. But unfortunately, the translation model is 500 times more likely for the wrong answer. There are a whole lot of spurious word misalignments attained from adding "serving:meal lunch" to the formal language; "serving:meal" generates "flight" and "that"; "lunch" generates "should" and "take"; "flight" generates nothing.

E: please tell me what flights leave CITY_1 next DAY_1 and land in CITY_2

A: List flights flying-on DAY_1 from:city CITY_1 to:city CITY_2

P: List flights morning flying-on DAY_1 from:city CITY_1 to:city CITY_2

In this sentence, the word "morning" spuriously generates the words "leave" and "in". These English words are rare, and in the reference answer, "leave" is generated from DAY_1 and "in" is generated from "flights". Unfortunately, "leave in the morning" is very likely, and hence "leave" and "in" have high likelihood of being generated from "morning". The headword models should help this.

E: i'd like to fly from CITY_1 to CITY_2 on AIR_1 and the plane should arrive around TIME_1

A: List flights AIR_1 arriving around TIME_1 from:city CITY_1 to:city CITY_2

P: List aircraft equipping flights AIR_1 arriving around TIME_1 from:city CITY_1 to:city CITY_2

The word "plane" is strong evidence that the user wants to know about aircraft. The only way to ever get this would be to have enough training data to be able to distinguish "the plane" from "which plane" or "what type of plane", and have a model powerful enough to condition this usage in an aircraft and a flight query. It is unlikely that one could ever get this much training data.

E: list all flights out of CITY_1 on AIR_1

A: List flights AIR_1 from:city CITY_1

P: List flights AIR_1 or from:city CITY_1 to:city CITY_1

The words "out of" are rare, and are spuriously generated by "or". Though one can imagine saying "I want to fly into or out of $\cdots$", and this will give the EM algorithm the opportunity to assign "out of" to "or" to increase the likelihood. In the correct answer "out of" doesn't even align to "from:city", "out" aligns to AIR_1 and "of" aligns to "List". $p(out \mid or) = .017$, $p(of \mid$

$or) = .026$, $p(out \mid from : city) = 2.4 \times 10^{-5}$, and $p(of \mid from : city) = .00045$.

E: i need a ticket from CITY_1 to CITY_2

A: List flights from:city CITY_1 to:city CITY_2

P: List flights round-trip from:city CITY_1 to:city CITY_2

The word "round-trip" is erroneously included to spuriously generate "ticket". This is because $p(ticket|round-trip) = .037$, $p(ticket|flights) = 2.4 \times 10^{-5}$, and $p(ticket|List) = .00044$. The high probability is because people often say "I want a round trip ticket $\cdots$".

E: what flights leave CITY_1 arriving in CITY_2

A: List flights from:city CITY_1 to:city CITY_2

P: List flights arriving afternoon from:city CITY_1 to:city CITY_2

Two formal language nodes are inserted, "arriving afternoon". Since "arriving" appears in the English, this has a strong preference to be generated from "arriving". Adding afternoon helps to spuriously generate "leave" and "in" as well, $p(leave|afternoon) = .023$, but $p(leave|flights) = .007$, $p(leave|from : city) = .00015$, $p(in|afternoon) = .147$, and $p(in|to : city) = .00066$. This happened in two sentences.

E: what flights leaving CITY_1 arriving in CITY_2 have first class seating

A: List flights first-class from:city CITY_1 to:city CITY_2

P: List airlines serving:flights flights first-class morning from:city CITY_1 to:city CITY_2

This time, "morning" is added to generate "leaving" and "arriving", and "serving:flights" added to generate "have". The spurious introduction of "serving:flights" to generate "have" happened twice.

E: i'd like the flights from CITY_1 to CITY_2 with a connecting flight anywhere

A: List flights connecting from:city CITY_1 to:city CITY_2

P: List airlines serving:flights flights connecting from:city CITY_1 to:city CITY_2 serving:flights flights direct from:city CITY_1 to:city CITY_2

More spurious insertions here: "serving-flights: generates "with" with probability $p(with|serving:flights) = .0688$, yet $p(with|connecting) = .00077$.

E: i'd like to buy a round-trip ticket flying into CITY_1 and out of CITY_2

A: List fares round-trip from:city CITY_2 to:city CITY_1

P: List fares cheapest one_direction from:city CITY_1 or one_way round-trip to:city CITY_2

More spurious insertions, "or" likes to generate "and" and "into" better than "from:city" and "to:city".

8.1.3 Complex Errors

A complex error is one that involves two or more errors.

E: i wanna fly from CITY_1 to CITY_2 and be there before TIME_1

A: List flights arriving before TIME_1 from:city CITY_1 to:city CITY_2

P: List aircraft equipping flights departing before TIME_1 from:city CITY_1 to:city CITY_2

In this sentence, "arriving" was replaced with "departing" and "aircraft equipping" is inserted. The word "wanna" is unknown, and is spuriously generated from "equipping" with 3 times greater probability than "before", the most likely formal language word to generate it in the reference answer. Also, "fly" is generated from "equipping" with 46 times greater probability than flights.

E: i need a round-trip ticket from CITY_1 to CITY_2 flying with AIR_1

A: List flights AIR_1 round-trip from:city CITY_1 to:city CITY_2

P: List fares AIR_1 round-trip thrift-class from:city CITY_1 to:city CITY_2

This sentence contains an insertion and a substitution. The insertion of "thrift-class" spuriously generates "with". What is interesting here is that the wrong answer was actually 9th on the list of answers. In all the 8 patterns that scored better, each had a different formal language word added to spuriously generate "with". This is especially surprising because "flying" and

AIR_1 are both generated from AIR_1. The clumping model could have chosen to align all three words to AIR_1 in a single clump if it were more likely. Even in the correct answer, "with" is generated from "List", not AIR_1. The relevant probabilities are: $p(with|List) = .00140$, $p(with|AIR\_1) = .00020$, and $p(with|thrift - class) = .108$.

E: what flights do you have available on DATE_1 leaving CITY_1 arriving in CITY_2 by

A: List flights flying-on DATE_1 from:city CITY_1 to:city CITY_2

P: List Extract flights arriving on DATE_1 from:city CITY_1 to:city CITY_2 Features the-number-of entries

In this case, "the-number-of" spuriously generates "have" and "by, and "arriving" generates "arriving" and "in". The last two are reasonable given the usual meaning for "arriving", but in this case, "arriving" is used in a new sense.

E: what sort of ground transport is available in CITY_1

A: List Extract ground-services provided-for:city CITY_1 Features all transport_type

P: List ground-services provided-for:airports airports serving:city CITY_1 provided-for:city CITY_1

Here, "provided-for:city" generates "sort", a rare English word, with 80 times greater probability than "transport_type".

E: what is the quickest flight flying from CITY_1 to CITY_2

A: List flights shortest from:city CITY_1 to:city CITY_2

P: List flights cheapest one_direction from:city CITY_1 to:city CITY_2

This time, "is" is spuriously generated from "one_direction" with probability $p(is|one\_direction) = .155$. This is almost 20 times more likely than $p(is|List)$.

E: please list flights from CITY_1 to CITY_2 round-trip whose cost is less than PRICE_1

A: List flights from:city CITY_1 less-than round_trip_cost PRICE_1 to:city CITY_2

P: List fares round-trip from:city CITY_1 less-than round_trip_cost PRICE_1 to:city CITY_2

This sentence contains an insertion and substitution. Fares spuriously generates "is" with 4 times greater probability than any word in the answer. Also "cost" is spuriously generated from "fares" instead of "round_trip_cost" with 4 times greater probability. Though it is reasonable for "cost" to be generated by "fares".

8.2 Reducing the Error Rate

It is clear from table 8.3, the analysis of the errors in the preceding section, and the discussion about the pattern match in section 4.4 that the major errors are:

- Missing pattern errors. This accounts for 70 errors, since missing patterns are automatically counted as an error. One could ignore these 70 test sentences in computing the error rate, but this is unfair. A pattern that has not been seen in 12000 context independent sentences is likely to contain rarer phenomena, and hence might not be correctly decoded by the statistical model. Since this thesis is not focusing on a decoder or language model, the missing pattern errors are not analyzed further. Note that chapter 10 does include error rates for DEV94 that use the pattern vocabulary augmented with any missing patterns.

- Translation model errors, mostly caused by spurious words.

- Language model errors, which are due to the unigram distribution on patterns seen in the training set. Though in all fairness, if the translation models were sharper for some of these sentences, then even with a poor language model prediction, the correct answer might still be found. It should also be mentioned that the language model helps to get many sentences correct. Again, since this thesis is not focusing on language models, this is not considered further.

- Permutation errors, in which the pattern matcher has two patterns that contain the same formal language words, that are permutations of each other. All the DEV94 permutation errors were due to confusion between which cities in a sentence are the departure cities, the arrival cities, and stopover cities.

There are many ways one could try fixing the spurious word and permutation errors. The next subsections describe a few potential solutions to each of these. Except for the next chapter which describes a new model to help solve permutation errors, no attempt is made to implement these in order to present better results. I conjecture that these solutions would help reduce the errors. To what extent I cannot be certain without implementing each of them.

Before presenting a discussion on potential solutions to these problems, one should mention that the modeling presented in this thesis has one potentially large source of error, that surprisingly did not prove to be a problem. This is the problem of "unknown" words. An unknown word is one that has never been seen in the training or smoothing data, but then appears in the test data. No statistics will exist for this unknown word. Consider, for example, what happens for the word "client" in this statistical NLU system. The word "client" is never seen in the training or smoothing data. Hence, "client" is not in the English vocabulary. Should "client" be seen in the test data, it is replaced by the unknown word, which is represented by "***". Since all words in the training and smoothing data are known, $p(***|f)$ and

$p(***)$ are 0. Thus, in computing $p(E|F)$ for a sentence E that contains the word "client", there is no formal language word that can generate "***" with non-zero probability. Hence, all patterns have $p(E|F) = 0$, and the sentence can not be decoded. Smoothing saves us, but in a very poor way. The smoothing for a translation probability, given by equation 5.16, for the unknown word is:

$$p_s(*** \mid f) = \alpha_{1,f} p(*** \mid f) + \alpha_{2,f} p(***) + \alpha_{3,f} \frac{1}{\ell(Voc_E)} \quad (8.1)$$

$$= \alpha_{3,f} \frac{1}{\ell(Voc_E)} \quad (8.2)$$

Thus, there is now a nonzero probability, and the unknown word can be generated. But, the formal language words most likely to produce the unknown words will be the ones for which $\alpha_{3,f}$ is largest, which will include many f due to the binning strategy based on frequency counts. In this thesis, no effort was made to model unknown words, because this was not a major contributor to the error rate. But in some other domain, or test set, one might not be as lucky.

8.2.1 Solutions For the Spurious Word Problem

A spurious word error occurs when a formal language word is introduced to generate an English word because $p(e \mid f)$ for this f is significantly higher than $p(e \mid f)$ for any f in the correct pattern. Some of the reasons spurious word errors occur are:

- If an English word is rare, than $p(e \mid f)$ will be poorly estimated, and to a crude approximation, the EM algorithm will give the most probability to $p(e \mid f)$ for the rarest "f" that appear with e.

- If an English word e is common and semantically relevant, then the statistics for $p(e \mid f)$ should properly train for the semantically correct f. Suppose e is not semantically relevant. Then it will occur in many different contexts. If these are not uniform, then an infrequent f may learn to generate a semantically meaningless e spuriously.

The next subsections propose modeling enhancements to help reduce spurious word errors.

Replacing Rare English Words with the Unknown Word

One way to reduce the spurious word problem for rare English words, and at the same time provide a solution for the unknown word problem, is to replace all English words with low frequency counts in the training set with the "unknown word", denoted canonically by "***". Chances are that if an English word e appears only once or twice, then the statistics that will be accumulated for it will be underestimated anyways. Suppose English word e appears once in the training set, and suppose that the formal language for this sentence uses an f that rarely appears. The EM algorithm will make $p(e \mid f)$ high in order to maximize the likelihood. Now suppose the test set uses e in a different semantic meaning, then the pattern matcher is

likely to incorrectly decode the sentence using the pattern that contains f, because $p(e \mid f)$ is the only way to generate e with high probability. Hence, a spurious word error results. But had e been replaced with "***", there would be many instances of "***" in the training set, and $p(*** \mid f)$ will have non-0 probability for many f since "***" occurs in many senses. Thus, replacing low count e with "***" serves two purposes. It will allow the generation of statistics for unknown words conditioned upon f, and it will help improve the spurious word problem.

Remove Meaningless Words

Another way to reduce the spurious word problem is to use a filter to remove meaningless words from the input. For example, the word "the" rarely contains semantic information, yet after training model 1, $p(the|all) = .38$, $p(the|midday) = .28$, and $p(the|equal - to) = .28$. Yet $p(the|List) = .21$. Thus, when "the" appears, there is a chance that a pattern could be selected which includes one of these formal words so "the" can be spuriously generated. But if "the" were removed from the English in the analysis, then this would not be an issue. This is one trick AT&T used in their system[63].

How does one recognize meaningless words? This is a research topic unto itself, though here are some ways:

- Use a lexicon to filter out words according to their parts of speech. For example, determiners and articles are apt to be semantically meaningless in ATIS.

- Use a statistical measure like *mutual information*[17]. The mutual information of two words e and f is defined as:

$$I(e;f) = \log \frac{p(e,f)}{p(e)p(f)} \tag{8.3}$$

This captures the notion of how often e and f occur together versus how often they occur by themselves. If e and f are independent, then $p(e,f) = p(e)p(f)$ and $I(e;f) = 0$. If e and f occur together frequently, then $I(e;f) >> 0$. If e and f rarely occur together, then $I(e;f) << 0$. The idea behind using mutual information is that if $I(e;f)$ is low for all f given an e, then this gives some evidence that e could be meaningless.

Determining appropriate thresholds can be tricky. Measuring the mutual information on a sentence basis seems more robust than on a word alignment basis. One could consider e having occurred with f if either they occur in the same sentence, or if e is generated from f in a Viterbi alignment. Another enhancement is to make use of the *negative* evidence $I(e;\bar{f})$, how often e occurs and f does not. If e is semantically relevant for f, then $I(e;\bar{f})$ should have low mutual information.

- Use the parameters of a trained translation model. This is closely related to mutual information. If one uses the headword/nonheadword models, the most probable headwords should be semantically relevant. The nonheadwords might contain both relevant and irrelevant words. For example, the formal language word "flights" has headword and

Headword	$p(Headword\|flights)$
flights	0.76754
flight	0.12155
there	0.0238927
list	0.0231245
leaving	0.0112259
that	0.0106361
in	0.0074895
leave	0.00465136
go	0.00407161
all	0.00396288

Table 8.4: Unsmoothed Model AHW Headword Probabilities for the Formal Word "flights" Trained On Just Hidden Training Data

nonheadword probabilities shown in tables 8.4 and 8.5.

The nonheadwords are all words that tend to be next to the English word "flights". Hence, the clumping model finds clumps that include these words, but they are appropriately given to the nonheadword distribution instead of the headword distribution. While "from" is a probable nonheadword for "flights", it has probability .81 of being a headword for "from:city" and .68 of being a headword for "from:airport". Not surprisingly, the following four words account for .992 nonheadword probability for "from:city": "fly", "going", "travel", and "go". All of these words can precede "from". This leads to another approach to reducing the spurious word problem.

Nonhead Word	$p(Nonheadword\|flights)$
all	0.173597
are	0.170147
the	0.132664
available	0.0827668
list	0.0685069
of	0.0565241
any	0.041508
there	0.0373753
from	0.0333291
go	0.0238467

Table 8.5: Unsmoothed Model AHW Nonheadword Probabilities for the Formal Word "flights" Trained On Just Hidden Training Data

Constrain English Words to Align to Particular Formal Words

Often, the relevance of an English word depends on the context. In "show me the flights from Boston to Denver departing before noon", "departing" is semantically relevant in that it is the only English word that indicates that "before noon" is a departure time. But in "show me the flights departing from Boston to Denver", "departing" is semantically irrelevant. While one cannot remove "departing" from the English, perhaps one can constrain "departing" to come from a small set of formal language words. If one is using a headword model, one can do even better by allowing departing to be a headword for a small set, and a nonheadword for a different set. It makes sense to allow "departing" to be generated from the formal words "departing", "departure_time", and "flying-on" as a headword. It also makes sense to allow "departing" as a nonheadword for "from:city", "from:airport", AIR_1,

etc. Using the trained parameter values and mutual information, it might be possible to force many probabilities to be zero, and hopefully guide the EM algorithm to better parameter settings.

Tie Semantically Similar Words

A common morphological technique is to replace words like "fly", "flying", "flight", "flies", and "flights" with a canonical stem. While these words are indeed used in different contexts, it is exactly these contexts that can lead to spurious word problems. For example, "flight" and "flights" tend to be used in flight queries, where "fly", "flies", and "flying" are used in airline queries. Consequently, the rare formal language words used in airline queries learn to spuriously generate these forms of the verb "to fly". This is another trick used by AT&T.

Use Different Senses for Words When Appropriate

Often an English word has more than one sense, and the translation models are unable to distinguish between the senses. While there are 123 instances of "to fly" in the English training set, there are 3800 instances of "to CITY_*" and 100 instances of "to ARP_*" ($* = 1, 2, 3 \ldots$). While the clumping models are intended to help recognize that "to" is sometimes an infinitival marker, the maximum likelihood estimation gives so much probability to $p(to \mid to : city)$, that almost always the word "to" is aligned to "to:city". This can then lead to a spurious insertion of a "to:city CITY_1". For example, in the

sentence "I would like to fly from CITY_1", the pattern matcher might select "List flight to:city CITY_1". If "to" were sensed however, then $p(to_{infinitival} \mid to : city)$ will not be high, and this type of spurious error avoided.

Remove Unneeded Formal Language Words

So far, the proposed solutions for reducing the spurious word errors have all focused on the English. The formal language used for ATIS has design deficiencies that also contribute to the spurious word problem. If there are infrequent "f", and one of these is used in the formal language for an English sentence that contains a rare word or a semantically meaningless word, then the EM algorithm will give most of the count for seeing the rare or meaningless e to the rare f. So, if one removes redundant or superfluous f, then there is less of a chance that when a rare e occurs, that there will be a rare f that can absorb its count. Given enough training data, the EM algorithm will learn which of the available f should generate this e. Here are several ways that the ATIS formal language needs to be cleaned:

- Operators between tables are almost always redundant. For example, "List aircraft equipping flights" could be replaced with "List aircraft flights". This is less "pretty", but the presence of equipping allows it to absorb the probability for less frequent English words. One could achieve the same result by setting $p(e \mid equipping) = 0$ for all e.

- Remove superfluous formal words used to make the formal language more canonical. For example, the word "all" is added before every column request that doesn't request the "minimum" or "maximum" operator. For example, the NL-Parse for "List flight numbers of flights" and "List all flight numbers of flights" both have the word "all" in the formal language. This was done to provide symmetry, as they have an identical meaning. But introducing "all" into the sentence gives a new formal language word that learn to generate infrequent e, and hence lead to spurious word errors later.

- Tie together different pre-terminal operators. For example, the decision was made to distinguish between the following different uses of "from" in NL-Parse:

  List flights from Boston

  List flights from cities named Boston

  List flights from JFK

  List flights from airports abbreviated JFK

  This leads to 4 different "from" nodes in NL-Parse, "from:city", "from:cities", "from:airport", and "from:airports". The "from:cities" and "from:airports" are very rare and consequently lead to spurious word errors. But if these similar words were replaced with one word, then this would be less likely to happen.

8.2.2 Reducing Permutation Errors

In order to provide a different translation model score for two formal language sentences that contain the same words, but in a different order, it is necessary to reformulate one or more components of the analysis-transfer-synthesis paradigm so that two permuted F generate the same E with different probabilities. Consider the two formal language sentences:

List flights from:city CITY_1 to:city CITY_2

List flights from:city CITY_2 to:city CITY_1

The current models make no use of the implicit tree structure that was used to generate these formal language sentences. In one tree, the "from:city" node is the parent of CITY_1. In the other tree, it is the parent of CITY_2. This relevant parent-child (predicate-argument) structure can be utilized in a variety of ways.

Use Thematic Roles for Tags

The reason that permutation errors exist, at least for the ATIS domain, is because the English tags contain a sense suffix. If the sensed tag were replaced with a thematic role tag, then the two formal language sentences would be replaced by one:

List flights from:city CITY_FR to:city CITY_TO

This places the burden of identifying thematic roles on the English analysis. While this probably works very well for ATIS, it is not a very general solution.

Perhaps in other domains, other permutations not related to tags exist.

Use a Divide and Conquer Approach

In the formal language tree, "from:city" is a nonterminal node, which has either CITY_1 or CITY_2 as a child. Instead of training the models in this thesis using the tags, one could use a two pass strategy for training and decoding. The first pass would ignore the tags, and the second pass would then use the tags. If the formal language tags are removed, the English CITY tags would now have to be constrained to come from any formal language word that could be the parent of a CITY tag. This includes "from:city", "to:city", and "stopping-in:city". Now, the clumping models should hopefully learn that "from:city" generates clumps that resemble "from CITY_1", "out of CITY_2", and so forth. The strings that could align to "from:city" could be very complex though, and one would have to write rules on how to map these strings to a meaning. This is in fact what AT&T and BBN have done. However, there is another solution. Upon determining what aligns to "from:city" and "to:city", one could then rescore the model now including the tags at the leaves, subject to the constraint that the tags can only align to words that formerly aligned to the parent. For example, in the English sentence "Show me the flights to CITY_1 from CITY_2", one would expect that the "from:city" will initially align to "from CITY_2". But upon implementing divide and conquer, the English words "from CITY_2" will now have to be realigned with the original subtree and tag constraints, so "from CITY_2"

can only align to the subtree that contains "from:city CITY_2" and not the subtree containing "from:city CITY_1". Thus, the permutation problem is avoided.

This solution requires the clumping models. For model 1, there is no reason that "from CITY_2" should both align to "from:city". The EM algorithm might choose to give more probability to CITY_2 aligning to "to:city", even though CITY_2 is adjacent to "from".

The divide and conquer approach seems very general, and a reasonable question to ask is why not do this from the highest level of the the formal language tree down. In fact, this was tried. At the highest level, the meanings were replaced with "List TABLE", where TABLE is "flights", "fares", "airlines", etc. The idea was that the EM algorithm should learn that "Please show me", "I would like to see", and "What" align to "List". In model 1 however, this was not the case. Consider for example, the English tag CITY_1. CITY_1 always occurs with "List". But not all flights mention cities, some mention airports. And, other queries like ground service queries, use cities. Thus, model 1 learns that CITY tags like to be generated from "List". The model does learn the other strings, but it unfortunately makes mistakes for common words like CITY tags. Perhaps if one used a clumping model, and constrained the number of clumps to be a small number, say just 1-3, then maybe a divide and conquer approach for the whole tree could be integrated into the model.

Use A Distortion Model

Another way to score two different permutations, is to use the heuristic that formal language words that are close in the formal language tree, should align to English clumps that are close in the English. This heuristic was applied in statistical machine translation, and called *distortion*[13]. One expects with high probability, that a clump aligned to the formal word "from:city" should be very close, if not adjacent to, the clump aligned to its child formal language word in the formal language tree. Thus, one can model the clump distortion between a child clump and its parent clump. The distortion $p(d)$ will then provide different scores for permuted formal language sentences, as the distortions will be different. Distortion is the most elegant solution to the permutation problem, and is investigated in chapter 9.

Chapter 9

A Distortion Model

9.1 Introduction

All the models to this point have used a formal language that is the preorder traversal of the parse tree of the cleaned and tagged NL-Parse. This preorder traversal throws out valuable information, in particular, the parent-child relationship between nodes. The two formal language sentences:

List flights from:city CITY_1 to:city CITY_2

List flights from:city CITY_2 to:city CITY_1

cannot be distinguished by any of the models presented so far. This is because clumps are generated independently by each f. One way for a translation model to give two different scores for these two formal language sentences is to model the proximity of the clumps aligned to each f. If the tree structure of the formal language is preserved, then a model can parameterize the

proximity of a child's clumps to its parent's. The parameters that model the proximities are called *distortions*[13]. In this chapter, the foundations of distortion modeling are presented, along with one distortion model, named model DIS.

To see how distortions might help disambiguate these two patterns, consider a model that models the distortion between the first clump of a CITY tag, and the first clump of its parent, from:city or to:city in particular. Since the first clump of the parent is being used as the reference location for the distortion, this is called the *anchor*. Denote the distortion probability as $p_{anchor}(d)$, where d can be either positive or negative. One can choose to condition this upon the parent formal language word if desired, $p_{anchor}(d \mid from : city)$ for example. For this illustrative example, assume that there is only one distribution for distortion. One expects that $p_{anchor}(d)$ is greatest for $d = 1$, and much lower for any other number. In the English sentence "Show me the flights to CITY_1 from CITY_2", "from" will align to "from:city" and "to" will align to "to:city" in either of the two formal language sentences:

List flights from:city CITY_1 to:city CITY_2

List flights from:city CITY_2 to:city CITY_1

But in the first formal language sentence, when the formal word CITY_1 generates CITY_1 in the English, it will have to use a distortion of $p_{anchor}(-1)$; when CITY_2 generates CITY_2 in the English it will have to use a distortion

of $p_{anchor}(3)$. In the second formal language sentence, the distortions will use $p_{anchor}(1)$ and $p_{anchor}(1)$. The latter will be much more likely, and hence the correct permutation is found.

Modeling distortions introduces many new complexities into maximum likelihood estimation. The distortion model is like all the previous models in that each clump is generated by a formal language word which is a hidden state of the model. But unlike the other models, the order the clumps are generated will affect the probability, as distortion models parameterize the probability of generating a clump given some other clumps that have already been generated. For lack of a better way, it seems reasonable to assume either a post-order or pre-order generation order.

Now that the order clumps are generated for formal language words is defined, a generation order for multiple clumps aligned to the same word needs to be defined. If there are n clumps, then there are $n!$ ways this ordering can be done, and each could lead to a different distortion value for the sentence generation, based on how distortions are modeled. To make the computation polynomial, a model should prescribe a specific generation order of clumps aligned to the same f, for example decreasing order of probability $p(c \mid f)$ or in left to right order. Assume the clumps are generated in left to right order, using a distortion to relate the distance between the previous clump generated and the current clump being generated. Again, this can be conditioned upon the formal language word, but assume that there is a second distribution $p_{within}(d)$. These distortions are only defined for $d > 0$.

Denote the position of the i-th clump of a formal word in the clumping C as pos_i, $pos_i \in \{1, 2, \cdots \ell(C)\}$. Denote the position of the first clump of a parent formal word as pos_{anchor}. If the parent formal language word does not generate any clumps, then recursively visit parents until one is found that generates clumps. This model computes the distortion score for a formal language word as:

$$d = p_{anchor}(pos_1 - pos_{anchor}) \prod_{i=2}^{n} p_{within}(pos_i - pos_{i-1}) \tag{9.1}$$

If pos_{anchor} is undefined[1], then the first factor is replaced by $\frac{1}{N}$, where N is the number of clumps that have not yet been generated.

This appears to be a perfectly fine model. The formal language words are visited in a pre-order fashion. The first formal language word to generate a clump generates its leftmost clump with uniform probability. Remaining clumps for this word are generated according to p_{within} in a left-to-right order. The remaining formal words are visited in pre-order fashion, and the first clump of each is generated according to p_{anchor} or a uniform probability, based on whether or not it has an ancestor that has generated a clump.

This model has a few problems with it, that are not readily apparent:

- If f generates n clumps, which are placed into the clumping in left-to-right order, the first clump can only be placed in the first $N - n + 1$

[1]An anchor is undefined if no ancestor for which distortions are being applied has generated a clump.

available clump positions. If it is placed any further to the right, then there will not be enough available positions to place down the remaining $n-1$ clumps.

- The probability distributions p_{anchor} and p_{within} are improperly conditioned. Since clump positions may already be full for f already visited in the pre-order traversal, not all slots are vacant. Thus, even if $\sum_i p_{anchor}(i)$ and $\sum_i p_{within}(i)$ sum to 1, not all these positions are vacant. Also, the distortions might refer to clump positions that are beyond the end of the clumping. For example, $p_{within}(i)$ for $i \geq 2$ refer to invalid positions if the first clump of a formal is in position $\ell(C)-1$.

These problems cause the model to be *deficient*[13]. A deficient model is one in which some of the probability distributions model events that cannot occur. For example, modeling the probability that a new clump is placed 2 positions to the right of the previous clump is an impossible event if this position has already been filled. In a deficient model, $\sum_{E,C,A} p(E,C,A \mid F) \neq 1$. One might propose removing the deficiency by having p_{anchor} and p_{within} represent distortions in terms of available slots. For example, $p_{within}(1)$ would mean the first available slot to the right of the first clump generated for f. This removes the deficiency that gives probability to placing a new clump in a filled position. But this does not get around the problem of running off the end of the clumping. If one conditions p_{anchor} and p_{within} on the number of available slots, then this would be a non-deficient model. For example,

$p_{within}(1 \mid av)$ would be the probability of placing a clump in the first available slot to the right of the first clump generated for f, given that there are av slots available. Now, $p_{within}(i \mid av)$ will be non-0 only for events that can occur, and $\sum_i p_{within}(i \mid av)$ will be 1. The model is now non-deficient. This model can be trained just like the model C variants. Model B is used to unhide the alignments. Each alignment is then visited in turn, and the clumps are placed down in the prescribed order, accumulating the score for the sentence in the process.[2] The scores for these alignments are then normalized, and then each alignment is revisited, accumulating a fractional count for each event that occurred.

There are two problems with this model. Conditioning the distortions on av is strange. One expects $p_{within}(1 \mid av)$ to be the same for all av. But conditioning upon av could lead to undertrained results. Another problem is that $p_{within}(1 \mid av)$ or $p_{anchor}(1 \mid av)$ might actually place a clump very far away from the previous clump being used to model the distortion, if all available slots in between are already filled. Thus, this distortion is not really measuring proximity, it is measuring proximity conditioned upon the order that the clumps are generated.

A better solution is to make use of an *auxiliary distribution*. Suppose one keeps p_{anchor} and p_{within} as originally defined. These are only valid probability distributions when all potential distortion values i are in the sentence

[2] The score includes the translation probabilities, clump length probabilities, fertility probabilities, and the distortion probabilities.

and available, which rarely occurs. Thus, it is not even appropriate to call these probability distributions. But, whenever a distortion is needed, if one divides this auxiliary distribution's value for the distortion by the sum of the auxiliary values that are defined, this leads to valid probability distributions p^*_{within} and p^*_{anchor}:

$$p^*_{anchor}(i) = \frac{p_{anchor}(i)\delta(\text{position i is in sentence and available})}{\sum_j p_{anchor}(j)\delta(\text{position j is in sentence and available})} \quad (9.2)$$

Basically, this is making a probability distribution p^* from the auxiliary distribution p by considering only the available slots.

This is also a non-deficient model, as $p^*(i)$ will always sum to 1 for available positions during the generation of the clumps of a sentence. Unfortunately, the EM algorithm cannot be used to perform maximum likelihood estimation for this model. The problem is that the denominator can be different each time an event associated with a parameter is observed. Thus one cannot simply remember the counts for the observed events, the denominator counts must be remembered too. For each parameter $p_{anchor}(i)$, one accumulates two counts, $\tilde{c}_{anchor}(i)$ which is the number of times a particular distortion between the first parent clump and the first child clump is observed in the training data, and $c_{anchor}(j)$, which is the model's prediction for the count for each available distortion j. Consider an example in which three positions are available in which the first clump of a child can be generated. Suppose these are in distortion positions -2, 3, and 4, and that position 3 is

the one that is actually filled. The following counts would be accumulated:

$$\tilde{c}_{anchor}(3) \quad += \quad 1 \tag{9.3}$$

$$c_{anchor}(-2) \quad += \quad \frac{p_{anchor}(-2)}{p_{anchor}(-2) + p_{anchor}(3) + p_{anchor}(4)} \tag{9.4}$$

$$c_{anchor}(3) \quad += \quad \frac{p_{anchor}(3)}{p_{anchor}(-2) + p_{anchor}(3) + p_{anchor}(4)} \tag{9.5}$$

$$c_{anchor}(4) \quad += \quad \frac{p_{anchor}(4)}{p_{anchor}(-2) + p_{anchor}(3) + p_{anchor}(4)} \tag{9.6}$$

Now remember, the goal in maximum likelihood estimation is to set the model's prediction to match the training data. It is intuitively reasonable to re-estimate a parameter using:

$$p_{anchor}(i)* = \frac{\tilde{c}_{anchor}(i)}{c_{anchor}(i)} \tag{9.7}$$

If a parameter is observed more often in the training data than the model predicts, the parameter for this event is *scaled* by the ratio of the observed count to the expected count. This iterative procedure is called *generalized iterative scaling*, and was shown to produce a maximum likelihood estimate by Darroch and Ratcliff[19]. The proof of convergence to a maximum likelihood estimate is not presented here.

9.2 Formulae

There are many possible distortion models one could implement using the strategy described in the previous section. The steps in designing a distortion model are:

- Pick a specific order in which to generate the clumps. Generally this is done by picking a specific order to visit the formal words according to the tree structure of the formal language, and then picking a specific order to generate the clumps aligned to each formal word.
- Pick the set of formal words for which distortions will be used.
- Determine the auxiliary distributions to be used based on the conditioning that is desired. A distortion value for generating a clump can be conditioned upon any one or more clumps that have already been generated. The anchor for the distortion could be any parent clump or previously generated clump for this formal word for example. The auxiliary distribution would be used to get a fractional count for each potential anchor. These fractional counts are then distributed to the auxiliary distribution for each potential distortion position according to what is available at the time the clump is generated.

To illustrate this principle, the following distortion model is implemented in this thesis:

- Distortions are generated for subtrees dominated by "from:city", "to:city", "stopping-in:city", "from:airport", "to:airport" and "stopping-in:airport" in the order they are visited in a preorder traversal. Before this preorder traversal is done, all clumps generated by formal words for which distortions are not used are generated first.

- Once the clumps are generated for formal words not utilizing distortions, the remaining formal words are visited in preorder fashion. The clumps aligned to formal subtrees dominated by "from:city", "to:city", "stopping-in:city", "from:airport", "to:airport" and "stopping-in:airport" are then generated according to a distortion probability.

- The first clump aligned to each formal language word uses a distortion $p_{anchor}(i \mid f_{preorder})$. That is, the previous formal word to generate clumps is used as the anchor. The distortion is conditioned upon the identity of the anchor formal word. If this is the first clump to be generated for the subtree, then the clump is generated with uniform probability over all available clump positions. Using the previously visited word in the preorder traversal instead of the parent allows for situations when a parent has 2 children, the parent generates no clumps, and each child does generate clumps. Then the first clump of the leftmost child serves as the anchor.

- Any formal language word for which distortions are being used generate the subsequent clumps after the first one by distorting positions using

$p_{within}(i \mid f)$. The remaining clumps are distorted according to the previous clump generated for the f, and a separate auxiliary distribution is kept for each f.

The formulae for this model are very similar to the ones for model C. Some random variables in model C will now be subscripted by d or nd to indicate whether or not they are being generated by a formal word for which distortions apply or not apply. The conditioning is rather complex, as all the formal words that do not utilize distortions generate their clumps first. For example, the probability of generating C_d must be conditioned upon C_{nd}. Also, there are new anchor points each time a new subtree is encountered. Formally showing all the formulae will make them appear more complex than they really are. Thus, in the formulae below, I omit some of the conditioning involving the distortions and instead opt for a less formal notation. Assume that the preorder representation for F_d has already been generated, so that the formal word f_i is the i-th word in the preorder traversal. The distortion model presented in this thesis is:

$$p(E \mid F) = \sum_{C,A} p(E, C, A \mid F) \tag{9.8}$$

$$p(E, C, A \mid F) = p(E_{nd}, C_{nd}, A_{nd} \mid F_{nd}) p(E_d, C_d, A_d \mid F_d) \tag{9.9}$$

$$p(E_{nd}, C_{nd}, A_{nd} \mid F_{nd}) = \frac{(L - L_{nd})!}{L!} \prod_{i=1}^{\ell(F_{nd})} p(n_i \mid f_i) n_i! \prod_{j=1}^{n_i} p(c_j \mid f_i) \tag{9.10}$$

$$p(E_d, C_d, A_d \mid F_d) = \prod_{i=1}^{\ell(F_d)} p(n_i \mid f_i) \prod_{j=1}^{n_i} p(c_j \mid f_i) * d(c_j) \tag{9.11}$$

$$p(c \mid f) \quad = \quad p(\ell(c) \mid f) \prod_{i=1}^{\ell(c)} p(e_i \mid f_c) \tag{9.12}$$

These formulae contain another notation change, each clump c_j is now the j-th clump generated by formal word f_i instead of the j-th clump in C. With this in mind, it is easy to verify that if all $f \in F$ do not utilize distortions, than $L_{nd} = L$ and these formulae compute the exact same value as model C. Thus, model C is really a special case of model DIS in which no f are defined with distortion handling.

One term is not yet defined in the above formulae, that is the distortion term $d(c_j)$. This is where I opt for non-mathematical notation. The distortion value for a clump takes one of three values:

$$d(c) = \left[\begin{array}{ll} \frac{1}{L_{open} - n_i + 1} & \text{if } j = 1 \text{ and the first clump} \\ & \text{being generated for a subtree} \\ p_{anchor}(pos_{c_1} - pos_{anchor} \mid f_{anchor}) & \text{if } j = 1 \text{ and not the first clump} \\ & \text{for the subtree} \\ p_{within}(pos_{c_j} - pos_{c_{j-1}} \mid f_i) & \text{if } j > 1 \end{array} \right] \tag{9.13}$$

9.3 Count Derivation

The count accumulation formulae for the translation probabilities, clump lengths, and fertilities are the same as in model C. That is, the score of

a candidate alignment is calculated (using the model DIS formulae). The score for each candidate alignment is then normalized, and the fractional count accumulated for each of these parameters.

For the distortion auxiliary distributions, the fractional count is accumulated in $\tilde{c}_{anchor}(i \mid f_{anchor})$ for the anchor distortion events observed in the candidate alignment. This fractional count is then further prorated to each potential anchor distortion j that is available and accumulated in $c_{anchor}(j \mid f_{anchor})$.

The maximization step of the EM algorithm for the translation probabilities, clump lengths, and fertilities uses standard frequency count normalization. The maximization of the auxiliary distributions uses generalized iterative scaling.

9.4 Training

The DIS distortion parameters are started from uniform initial statistics. The rest of the DIS model parameters are initialized from model C. The translation probabilities, clump lengths, and fertilities are held fixed for 10 iterations while the distortion parameters are trained. Then five more iterations are run in which other parameters are varied and the distortion parameters fixed.

9.5 Smoothing

Smoothing is only done for the model C parameters: the translation probabilities, clump lengths, and fertilities. These are smoothed for 8 iterations starting from uniform statistics.

9.6 Results

The results from this distortion model show a slight improvement over the model C results. Unfortunately, not all of the 8 permutation errors were corrected due to the language model still being too strongly in favor of the wrong permutation. Two slight changes that could easily fix all 8 permutation errors are:

- The most common distortion error is when the departure and arrival city are "toggled" between the correct answer and the incorrect permutation. Because the model presented in this chapter generates all clumps for formal language words that do not utilize distortions first, when the final subtree's last clump is generated, there are no other available positions to contribute to the denominator of equation 9.2. For example, the motivating example given at the beginning of this chapter showed the correct permutation will use $p_{anchor}(1)$ and $p_{anchor}(1)$ versus $p_{anchor}(-1)$ and $p_{anchor}(3)$ for the incorrect permutation. But because the final clump only has one available slot, $p^{*}_{anchor}(1)$ for the

correct permutation and $p^*_{anchor}(3)$ for the incorrect permuation will both have the value 1.0. If one generated the clumps first for the nodes utilizing distortions, and then the remaining clumps, then there would be other denominator values, and true distortion values would be used, and $p^*_{anchor}(1) >> p^*_{anchor}(3)$.

- Rather than changing the model, one should really use a distortion model only for what it is intended, to help disambiguate two permutations. One could thus rebuild the pattern matchers vocabulary of patterns, and tie together permuted patterns and their counts. This reduced set of patterns would be used to find the most likely pattern. Upon discovering that this pattern is a canonical example of a tied set of permutations, one could apply the distortion model *without* the language model to disambiguate the two permutations.

9.6.1 Exact Match Maximum Likelihood Results for DEV94

The smoothed model DIS results are shown in the following table, along with smoothed model C results for comparison.

As mentioned in the preceding section, only 1 of the 8 permutation errors was corrected. Though for all 8, the translation model scores became vastly better for the correct answer. They just were unable to overcome the language model score. To illustrate the point, here is what happened for one

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
C	262	290	293
DIS	263	293	294

Table 9.1: Exact Match Maximum Likelihood Decoding Results for DEV94 Using Smooth Models C and DIS as a Function of Amount of Hand and Hidden Training Data

of the 7 permutation errors that was not corrected.

E: the flight going to CITY_1 from CITY_2 should stop in CITY_3

A: List flights from:city CITY_2 stopping-in:city CITY_3 to:city CITY_1

W1: List flights from:city CITY_1 serving:meal lunch stopping-in:city CITY_3 to:city CITY_2

W2: List flights from:city CITY_1 stopping-in:city CITY_3 to:city CITY_2

The language model scores are .000231642 for A, the correct answer. Incorrect answers W1 and W2 have language model scores .00030886 and .0100378 respectively. When run with smoothed model C, W1 was found as the most likely answer, with a translation model score of 1.76674×10^{-18}, giving a total score of 2.33596×10^{-11}. Smoothed model C gives incorrect answer W2 a translation model score of 4.46201×10^{-20} for a total score of 2.11634×10^{-11}. The correct answer has translation model score of 4.46201×10^{-20} for a total score of 3.21495×10^{-12}. Thus, for smoothed model C, the incorrectly found answer is 7.27 times more likely than the correct answer.

When run with smoothed model DIS, the correct answer has translation model score 1.39921×10^{-18} for a total score of 1.80033×10^{-11}. Incorrect answer W1 has a translation model score of 1.31229×10^{-17} for a total score of 6.3664×10^{-11}. Incorrect answer W2 has a translation model score of 9.30566×10^{-19} for a total score of 9.66482×10^{-11}. Thus, smoothed model DIS selects incorrect answer W2 as the most likely answer, which is 5.37 times more likely than the correct answer. But a closer inspection of the translation model and language model scores between the correct answer and incorrect answer W2 shows that the distortion model is doing its job. The distortion model score for the correct answer is 1.5 times more likely than the incorrect answer W2. But unfortunately, the language model estimate of incorrect answer W2 is 43 times more likely. Hence an incorrect answer is still selected.

Also note that distortions help to reduce spurious word errors. For a word to be spuriously produced, it must be done so by a formal language word for which distortions are not being used, otherwise a distortion penalty will be introduced. Thus, the incorrect answer W1 found by model C was replaced by a different incorrect answer, that is "closer" to the correct answer in that it doesn't contain any spurious word errors.

9.6.2 Cross Entropy Results for DEV94

The cross entropy results are shown in table 9.2 for smoothed models C and DIS.

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
C	5.30	4.93	4.99
DIS	5.14	4.93	4.99

Table 9.2: Cross Entropy Results for DEV94 For Smoothed Models C and DIS

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
C	90.9	93.7	94.8
DIS	91.0	94.9	94.8

Table 9.3: Viterbi Percentages of Maximum Likelihood Cross Entropy for DEV94 For Unsmoothed Models C and DIS

9.6.3 Viterbi Percentage Results for DEV94

The Viterbi percentages for models C and DIS are shown in table 9.3. The Viterbi percentages shown are calculated using unsmoothed parameter sets.

Chapter 10

Summary

In this thesis, a framework for performing statistical NLU using the source-channel model paradigm has been developed. English is analyzed with a tagger to reduce the parameter set. The formal language is derived from NL-Parse, which although not perfect for statistical understanding, was convenient to use. Statistical models were designed to generate each English word from a formal language word using various parameters. The parameters found to be most valuable were:

- Translation probabilities $p(e \mid f)$.
- Clump length probabilities $p(\ell(c) \mid f)$.
- Poisson fertilities λ_f.
- Fertility probabilities $p(n \mid f)$.
- Distortion probabilities $p_{anchor}(d)$ and $p_{within}(d)$.

It is hard to compare the results of this research to others, since a large percentage of the errors are due to aspects of NLU not examined by this thesis: the decoder and the language model. In DEV94, the smoothed model B parameters trained using all the hand and hidden training data get 290 of 410 sentences correct. But 70 of the 120 incorrect sentences are due to missing patterns and 16 are due to a bad language model prediction.[1] One potential way of measuring model performance, factoring out the effect of the decoder and language model, is to augment the pattern vocabulary with the missing patterns, and see if the translation models are good enough to pick the correct pattern, which is guaranteed to be in the search set. Augmenting the pattern matcher vocabulary in this manner will certainly the performance one could hope to get with a real decoder. A real decoder will still have search errors, and will also search many additional confusable patterns, much more so than in the pattern matcher vocabulary. In section 10.1, the CAS results are given for all the models using DEV94, DEC93, and DEC94, with and without augmentation. Section 10.2 compares these results to the other ARPA HLT participants. Since one goal of performing NLU statistically is to provide a more portable alternative to traditional linguistic approaches, section 10.3 examines the results with respect to portability. Then, a few final words conclude the thesis in section 10.4.

[1] Though as noted in chapter 8, with a sharper translation model prediction, this might have overcome a poor language model prediction.

10.1 Summary of Results

To this point, all results have used the DEV94 test set and the exact match evaluation metric. In this section, the results are presented for three test sets, DEV94, DEC93, and DEC94. The latter two are official ARPA test sets that were used to compare competing systems from the participants in the HLT workshop.

The following 6 tables contain the results for these 3 test sets, with and without augmenting the pattern matcher vocabulary to include the correct answer. From the tables, the following conclusions are drawn:

- Using just hidden training data introduces a 2.5% - 5% degradation in performance between the best models for each training scenario.
- The best augmented results are 88.05% for DEV94, 87.28% for DEC93, and 84.27% for DEC94.
- Augmented results are 3.5% - 7.5% better than the unaugmented results.
- The model DIS, CHW, and CLM results are usually the best. Since model DIS is a generalization of model C, this implies that potentially models DISHW and DISLM could improve the CHW and CLM results. Thus, future work should include developing and testing models DISHW and DISLM.

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	71.22	76.10	74.88
1 - sm	74.15	77.80	75.61
A	70.73	76.83	75.12
A - sm	75.12	77.07	77.07
AHW	75.12	77.56	74.39
AHW - sm	74.63	78.05	76.59
ALM	70.73	75.85	73.41
ALM - sm	75.37	79.76	79.51
B	71.95	76.83	76.10
B - sm	75.61	78.29	77.56
BHW	69.02	78.54	76.34
BHW - sm	74.15	80.00	79.02
BLM	71.95	76.83	76.10
BLM - sm	75.61	78.29	77.56
C	71.46	77.32	76.83
C - sm	75.37	77.56	78.29
CHW	70.24	78.54	77.56
CHW - sm	76.83	79.27	79.02
CLM	65.37	78.29	75.61
CLM - sm	72.68	80.49	79.76
DIS	69.02	78.78	78.05
DIS - sm	75.85	78.29	78.78

Table 10.1: CAS Results for DEV94

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	78.54	82.93	81.22
1 - sm	80.24	83.90	81.95
A	78.54	83.90	81.46
A - sm	81.46	84.39	83.90
AHW	82.44	83.90	80.49
AHW - sm	82.20	85.85	82.93
ALM	77.56	84.15	80.49
ALM - sm	82.20	86.59	86.34
B	78.54	84.15	82.68
B - sm	81.46	85.61	85.12
BHW	78.78	84.39	82.93
BHW - sm	81.71	85.85	85.37
BLM	78.54	84.15	82.68
BLM - sm	81.46	85.61	85.12
C	79.02	85.12	83.41
C - sm	81.95	85.85	86.10
CHW	78.54	85.85	84.15
CHW - sm	83.90	87.07	86.34
CLM	72.68	86.34	81.95
CLM - sm	79.51	88.05	87.07
DIS	76.59	86.59	85.12
DIS - sm	82.93	86.59	87.32

Table 10.2: CAS Results for DEV94 With Pattern Vocabulary Augmented with Correct Answer

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	73.66	75.67	77.23
1 - sm	75.00	75.22	78.35
A	73.88	75.45	75.89
A - sm	74.78	77.01	77.90
AHW	73.44	75.22	75.67
AHW - sm	75.89	78.35	78.35
ALM	74.33	76.34	75.00
ALM - sm	76.79	78.12	78.79
B	76.12	79.91	76.79
B - sm	78.12	81.25	79.91
BHW	76.12	79.91	76.79
BHW - sm	78.12	81.25	79.91
BLM	76.12	79.91	76.79
BLM - sm	78.12	81.25	79.91
C	75.45	80.80	78.12
C - sm	79.91	82.59	81.70
CHW	75.45	80.58	77.90
CHW - sm	79.91	79.91	81.25
CLM	70.09	82.37	79.02
CLM - sm	73.21	83.04	82.37
DIS	75.89	81.25	78.35
DIS - sm	78.35	83.04	82.37

Table 10.3: CAS Results for DEC93

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	78.12	79.91	81.47
1 - sm	77.68	78.79	81.92
A	78.79	80.36	79.91
A - sm	78.57	80.80	81.25
AHW	79.02	80.36	80.13
AHW - sm	79.69	82.59	81.92
ALM	79.02	81.25	78.79
ALM - sm	80.58	82.59	82.59
B	81.03	85.04	81.47
B - sm	82.14	85.27	84.15
BHW	81.03	85.04	81.47
BHW - sm	82.14	85.27	84.15
BLM	81.03	85.04	81.47
BLM - sm	82.14	85.27	84.15
C	81.03	85.27	81.92
C - sm	83.48	85.71	85.27
CHW	79.91	85.04	81.70
CHW - sm	82.81	83.93	84.82
CLM	75.00	87.28	82.81
CLM - sm	76.79	87.05	86.16
DIS	80.58	85.94	82.37
DIS - sm	81.92	86.38	86.16

Table 10.4: CAS Results for DEC93 With Pattern Vocabulary Augmented with Correct Answer

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	70.34	73.93	71.69
1 - sm	71.91	76.18	73.48
A	70.11	73.26	71.69
A - sm	73.71	75.73	74.61
AHW	71.46	73.93	73.26
AHW - sm	75.28	77.08	76.18
ALM	70.56	72.13	72.58
ALM - sm	74.83	75.51	76.40
B	69.66	73.26	71.24
B - sm	74.16	75.51	74.38
BHW	69.66	73.26	71.24
BHW - sm	74.16	75.51	74.38
BLM	69.66	73.26	71.24
BLM - sm	74.16	75.51	74.38
C	68.76	72.36	70.34
C - sm	72.13	75.51	75.28
CHW	65.39	76.18	72.36
CHW - sm	72.81	77.08	77.30
CLM	65.17	73.26	71.46
CLM - sm	72.13	77.08	76.63
DIS	68.09	73.48	71.69
DIS - sm	74.61	76.85	76.85

Table 10.5: CAS Results for DEC94

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
1	76.40	80.90	77.98
1 - sm	77.08	81.80	79.78
A	76.18	80.22	77.30
A - sm	78.65	81.12	79.78
AHW	78.65	81.35	79.33
AHW - sm	81.12	83.37	82.25
ALM	77.75	80.45	79.78
ALM - sm	81.12	82.70	82.47
B	75.96	80.00	77.08
B - sm	79.55	81.12	80.22
BHW	75.96	80.00	77.08
BHW - sm	79.55	81.12	80.22
BLM	75.96	80.00	77.08
BLM - sm	79.55	81.12	80.22
C	75.73	80.00	76.85
C - sm	77.98	82.02	81.80
CHW	72.58	83.37	79.10
CHW - sm	79.33	84.04	84.27
CLM	72.58	81.12	78.88
CLM - sm	78.20	83.15	83.15
DIS	75.96	81.35	78.65
DIS - sm	80.45	83.60	83.60

Table 10.6: CAS Results for DEC94 With Pattern Vocabulary Augmented with Correct Answer

System	DEC93	DEC94
AT&T	92.6	96.2
BBN-Delphi	90.4	N/A
BBN-HUM	83.9	90.5
CMU	94.0	96.2
MIT	90.0	95.5
Paramax/Unisys	71.4	76.4
SRI	89.5	93.0

Table 10.7: ARPA HLT Accuracy Rates

10.2 How Do the Results Compare to Other ARPA HLT Participants

The official CAS test results for the DEC93 and DEC94 test sets for the ARPA HLT participants are given in table 10.7. While not shown, the participants agreed at the January 1995 meeting that the DEV94 test set was a hard test set, and most participants scored approximately 85% on this test. While this is a hard test set, it was also never adjudicated. Adjudication on the previous test sets changed approximately 10% of the test data. This would certainly fix the 2 errors in which sentences were incorrectly labeled class A instead of class D. The augmented test results for DEV94 and DEC93 shown in tables 10.2 and 10.4 are comparable to the ARPA HLT participants when some hand training data are used.

The DEC94 results in this thesis are several percent worse than DEV94 and DEC93. I have no explanation for this, since DEC94 is a hidden test set. The problem could be due to unknown words, tagger errors, more com-

plicated formal language, permutation errors, spurious words, the language model, or any combination of these.

In comparing the results of this thesis to the work of others, one needs to remember that for most of these sites, DEC93 was their third or fourth evaluation, and DEV94 was their fourth or fifth. In this thesis, DEV94 was my first evaluation. From this, some problems discovered were:

- The pattern matcher coverage leads to a large percentage of errors.
- Spurious word problems are the next largest source of error.
- The language model is the third largest source of error.
- Permutation errors are the last significant source of error.

One or two iterations of this research would help reduce these errors significantly. The distortion model presented in chapter 9 always prefers the correct permutation to an incorrect one. However, because all patterns are searched with the distortion model, incorrect patterns are still found because favorable distortions outweigh incorrect translation probabilities. One could first use another model to select the most likely pattern. One could then easily generate permutations from this pattern, for example the cities could be switched. Then the distortion model could be used to select the most likely permutation. This approach will remove all the permutation errors in DEV94. Numerous suggestions were given in section 8.2 to reduce spurious word errors. A decoder and improved language model are needed to remove

the final sources of errors. I'm convinced that another turn or two of the crank and the results would improve significantly.

In addition, the results given for DEV94 in each chapter show that many of the models can benefit from more training data. Thus, the inclusion of additional training data should also help improve the results of these models.

10.3 How Portable are These Results

The statistical approach in this thesis, based on the source-channel paradigm, is similar to the work done by AT&T in their CHRONUS system, and BBN in their hidden understanding models. However, the work presented in this thesis is more portable in that:

- The formal language is very close to NL-Parse (and hence SQL). Thus the synthesis of the formal language into an answer is deterministic. No domain expert is required to write rules or grammars on how to extract the relevant SQL clauses from the strings of words aligned to a semantic concept. Though recent BBN work has accomplished the same effect using decision trees[55].

- The models can be trained with little or no hand aligned training data. If little training data are available, then hand alignments are needed. But if ample training data exists, the models in this thesis can be trained with no hand alignments. For ATIS, most models still see sig-

nificant benefit when trained with 5627 instead of 2813 hidden training sentences.

10.3.1 Formal Language

The formal language used in this thesis is the pre-order traversal of the cleaned and tagged NL-Parse. In the AT&T and BBN systems, concepts are used instead. For example, "from:city CITY_1" might be replaced by "DEPART_LOC" in the other systems. If the formal language for each English sentence is entered manually, than the formal language in this thesis has no advantage over the others. Since the formal language used in this thesis is derived from SQL, I claim a slightly higher degree of portability should SQL translations of the input exist. In porting a natural language system to a new domain for a customer, it would be entirely reasonable to require the customer to provide English and SQL. It would be less reasonable to require the customer to annotate their data with a formal language suited for a statistical system. The latter requires additional work. The former might already be available. Given the English and the SQL, it might be possible to derive an NL-Parse like formal language automatically. I'm sure that some problematic sentences might exist, but most queries without complex table recursion within the SQL should be easily convertible.

10.3.2 Synthesis

In the AT&T and BBN systems, the formal language concepts are aligned to strings of English words. These words are then passed through a grammar for each concept, to convert the English words to a meaning. For example, departure locations need to be recognized as either cities or airports, and then the SQL fragment to access the ATIS database needs to be generated. Thus, for each new application domain, new grammars will have to be written for each formal language concept.[2]

In this research, the formal language is directly mappable into SQL. No grammar needs to be written. It is true that a filter to convert the formal language into SQL needs to be written, but being deterministic, it does not depend on the identity of the English words that it is parsing. Hence, it is easier to write and their is no chance of ambiguity introducing an error.

On the other hand, this research requires the English to be tagged. To the extent that taggers for the lexical items shown in table 4.1 are portable, then the tagger can be ported from other domains. Even if a new application domain requires a new tagger, bracketing English words and giving them a tag in order to train a statistical tagger is a relatively fast procedure, and does not require a domain expert. I'll also conjecture that given the availability of SQL, one might be able to develop a statistical algorithm for doing the bracketing automatically. The SQL will indicate the presence of a time, city,

[2] As already mentioned, the most recent BBN work does this mapping statistically.[55]

airport, and so forth somewhere in the query. Given this information and the identity of the tags, it should be possible to build models to find the most probable words in the English to give rise to these tags.

10.3.3 Training

The AT&T and BBN systems are basically model ALM, except they include a bigram language model to parameterize which formal language concept is likely to generate a clump given the formal language word that generated the previous clump. This parameter is similar to distortions. In one sense it is weaker in that it makes no use of the structure of the parse tree. But it is stronger in that it will most likely remove many spurious word errors. A spurious word error causes a single word clump to be inserted in between two other clumps, solely because the translation probability is better. With a language model parameter to regulate this, spurious word errors can only occur that are consistent with probable bigrams. This additional power provided by this parameter also causes problems when trained from uniform initial statistics. The model quickly overtrains to the training data. Hence, both AT&T and BBN found it necessary to do their training using 100% hand-aligned training data. Aligning pairs of English and formal language sentences does not take too long to do, but it is less portable than not requiring aligned training data. Most of the models presented in this thesis benefit from using hand aligned data, but the degradation is only a few percent. Further, many of the models have not even reached their peak performance

as a function of training data. Thus, in terms of portability, if very little training data are available, then one will have to align the data by hand. But if 5000 or more training data are available, then no hand alignments are necessary for the models in this thesis. However, it would be interesting see if their model, bootstrapped from one of the models in this thesis, leads to an increase in performance.

10.3.4 Modeling

In this thesis, five different models were developed, model 1, A, B, C, and DIS. Each of the clumping models in addition allows the translation probabilities to be generated by unigram distributions, bigram distributions, or headword/nonhead word unigram distributions. For any new application domain, any one of these models might be "best". For example, in examining the augmented and unaugmented results earlier in this chapter, one discovers the best models for each test set shown in table 10.8.

Models CHW, CLM, and DIS are always the best when hand training data are used. But when using nothing but hidden training data, models AHW, ALM, C, and CHW are each better for different test sets. Thus, it appears necessary to have many different models. Each has its own strengths and weaknesses, which may or may not be relevant for a new application domain. In porting to a new domain, one should hold out a test set of at least 1000 sentences, and use this to determine which model works best.

Model	5627 Hidden Sent. 0 Hand Sent.	5627 Hidden Sent. 3575 Hand Sent.	0 Hidden Sent. 3575 Hand Sent.
DEV94	CHW	CLM	CLM
	76.83%	80.49%	79.76%
DEV94	CHW	CLM	DIS
w/aug.	83.90%	88.05%	87.32%
DEC93	C/CHW	DIS/CLM	DIS/CLM
	79.91%	83.04%	82.37%
DEC93	C	CLM	DIS/CLM
w/aug.	83.48%	87.28%	86.16%
DEC94	AHW	AHW/CHW/CLM	CHW
	75.28%	77.08%	77.30%
DEC94	AHW/ALM	CHW	CHW
w/aug.	81.12%	84.04%	84.27%

Table 10.8: Best CAS Results for each test set

10.4 Final Summary

This thesis has investigated using the source-channel paradigm for statistical natural language understanding. A high degree of accuracy has been attained by utilizing a hierarchy of models. The results are better than the Paramax/Unisys system in the ARPA HLT workshop, but are 5-10% worse than the other participants. Yet these systems were developed over 4-5 year periods, and had the advantage of numerous evaluations and iterations of improvements. The research presented by this thesis was performed over a 2 year period, and the first results are being presented now. The results are very promising, but show that more work is needed to achieve state of the art performance. Future work should focus on the following areas:

- The design and development of a decoder and language model.

- Reduction of spurious word errors, by one or more of the suggestions mentioned in section 8.2.

- The implementation of more models. In particular, the AT&T and BBN model that includes bigram parameters to model the probability that a clump aligned to one f follows a clump aligned to another f seems worth trying. While AT&T and BBN needed hand aligned data to train the model, it might be that this model can be bootstrapped by initializing from one of the models presented in this thesis.

  All the models presented in this thesis assume that the $p(c \mid f)$ is calculated independently for all c aligned to an f. While the fertility models parameterize the number of c aligned to f, they do not adjust the statistics used in the calculation of $p(c \mid f)$. One way to do this would be to use a link grammar[76, 43]. This might help catch some linguistic phenomena like embedded clauses, as in "late DAY_1 evening", where "late" and "evening" have to align to the single formal language word "late_evening".

- Try building a system to handle context dependent sentences by training on context dependent formal language fragments, and including fragments into the pattern matcher vocabulary.

Bibliography

[1] Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.

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

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

[4] M. Bates and D. Ayuso. A proposal for incremental dialogue evaluation. In *Fourth DARPA Workshop on Speech and Natural Language*, pages 319–322, Pacific Grove, California, February 1991. Morgan Kaufmann Publishers, Inc.

[5] M. Bates, S. Boisen, and J. Makhoul. Developing an evaluation methodology for spoken language systems. In *Proceedings of the DARPA Speech*

and Natural Language Workshop, pages 102–108, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

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

[7] Adam Berger, Peter Brown, Stephen DellaPietra, Vincent DellaPietra, John Gillett, John Lafferty, Harry Printz, and Lubos Ureš. The Candide system for machine translation. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 157–162, Plainsboro, NJ, March 1994. Morgan Kaufmann Publishers, Inc.

[8] D. Bobrow. Natural language input for a computer problem-solving system. In M. Minsky, editor, *Semantic Information Processing*, pages 135–215. MIT Press, Cambridge, MA, 1968.

[9] S. Boisen, L. Ramshaw, D. Ayuso, and M. Bates. A proposal for SLS evaluation. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 135–146, Cape Cod, MA, October 1989. Morgan Kaufmann Publishers, Inc.

[10] Eric Brill, David Magerman, Mitchell Marcus, and Beatrice Santorini. Deducing linguistic structure from the statistics of large corpora. In *Third DARPA Workshop on Speech and Natural Language*, pages 275–282, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

[11] Peter F. Brown, John Cocke, Stephen A. DellaPietra, Vincent J. DellaPietra, Frederick Jelinek, Robert L. Mercer, and Paul S. Roossin. A statistical approach to French/English translation. In E. Clementi and S. Chin, editors, *Biological and Artificial Intelligence Systems*, pages 547–562. ESCOM Science Publishers, B.V., Leiden, The Netherlands, 1988.

[12] Peter F. Brown, John Cocke, Stephen A. DellaPietra, Vincent J. DellaPietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June 1990.

[13] Peter F. Brown, Stephen A. DellaPietra, Vincent J. DellaPietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, June 1993.

[14] P.F. Brown. *The Acoustic-Modeling Problem in Automatic Speech Recognition.* PhD thesis, Carnegie-Mellon University, May 1987. Also IBM Research Division Technical Report RC 12750.

[15] S. Chen. Bayesian grammar induction for language modeling. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 228–235, Cambridge, MA, June 1995. Morgan Kaufmann Publishers, Inc.

[16] K. Church and W. Gale. Enhanced Good-Turing and cat-cal: Two new methods for estimating probabilities of English bigrams. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 82–91, Cape Cod, MA, October 1989. Morgan Kaufmann Publishers, Inc.

[17] K. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, BC, June 1989. Morgan Kaufmann Publishers, Inc.

[18] Kenneth Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136–143, 1988.

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

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

[21] G. David Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, March 1973.

[22] K. Fu and T. Booth. Grammatical inference: Introduction and survey - part i and ii. *IEEE Transactions on Systems, Man, and Cybernetics*, 5:95–111,409–423, 1975.

[23] A. Gorin. Semantic associations, acoustic metrics and adaptive language acquisition. In *Proceedings of the 1994 International Conference on Spoken Language Processing (ICSLP)*, volume 1, pages 79–82, Yokohama, Japan, September 1994. The Acoustical Society of Japan.

[24] A. Gorin, H. Hanek, R. Rose, and L. Miller. Spoken language acquisition for automated call routing. In *Proceedings of the 1994 International Conference on Spoken Language Processing (ICSLP)*, volume 3, pages 1483–1486, Yokohama, Japan, September 1994. The Acoustical Society of Japan.

[25] A. Gorin, S. Levinson, A. Gertner, and E. Goldman. Adaptive acquisition of language. *Computer Speech and Language*, 5:101–132, 1991.

[26] C. Hemphill, J. Godfrey, and G. Doddington. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 96–101, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

[27] L. Hirschman. Multi-site data collection for a spoken language corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 7–14, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[28] L. Hirschman, D. Dahl, D. McKay, L. Norton, and M. Linebarger. Beyond class A: A proposal for automatic evaluation of discourse. In *Pro-*

ceedings of the DARPA Speech and Natural Language Workshop, pages 109–113, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

[29] L. Hirshman, M. Bates, D. Dahl, W. Fisher, J. Garofol, D. Pallett, K. Hunicke-Smith, P. Price, A. Rudnicky, and E. Tzoukermann. Multi-site data collection and evaluation in spoken language understanding. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 19–24, Princeton, NJ, March 1993. Morgan Kaufmann Publishers, Inc.

[30] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[31] K. Hunicke-Smith and J. Bernstein. Annotation of ATIS data. In *Proceedings of the ARPA Human Language Technology Workshop*, page 412, Princeton, NJ, March 1993. Morgan Kaufmann Publishers, Inc.

[32] W.J. Hutchins. *Machine Translation: Past, Present, Future.* Ellis Horwood Limited, West Sussex, England, 1986.

[33] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, pages 675–685, November 1969.

[34] F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. Decision tree parsing using a hidden derivational model. In *Proceedings of the ARPA Human Language Technology Workshop*, pages

272–277, Plainsboro, NJ, March 1994. Morgan Kaufmann Publishers, Inc.

[35] Frederick Jelinek. Self-organized language modeling for speech recognition. Unpublished IBM-Internal Report, 1985.

[36] K. Sparck Jones. Towards better NLP system evaluation. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 102–107, Plainsboro, NJ, March 1994. Morgan Kaufmann Publishers, Inc.

[37] Wilbur B. Davenport Jr. *Probability and Random Processes.* McGraw-Hill, New York, NY, 1970.

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

[39] Brian W. Kernighan and Rob Pike. *The UNIX Programming Environment.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[40] Joshua Koppelman. A statistical approach to language modelling for the ATIS problem. Master's thesis, MIT, January 1995.

[41] R. Kuhn and R. De Mori. Learning speech semantics with keyword classification trees. In *Proceedings of the IEEE International Conference on*

Acoustics, Speech and Signal Processing, volume 2, pages 55–58, Minneapolis, MN, April 1993.

[42] R. Kuhn and R. De Mori. The application of semantic classification trees to natural language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):449–460, May 1995.

[43] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, MA, October 1992.

[44] Kai-Fu Lee. *Automatic Speech Recognition: The Devlopment of the SPHINX System.* Kluwer Academic Publishers, Boston, MA, 1989.

[45] E. Levin and R. Pieraccini. CHRONUS, the next generation. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 269–271, Austin, TX, January 1995. Morgan Kaufmann Publishers, Inc.

[46] D. Magerman. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, MA, June 1995. Morgan Kaufmann Publishers, Inc.

[47] D. Magerman and M. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings of the AAAI*, Boston, MA, 1990.

[48] D.M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition.* PhD thesis, Stanford University, February 1994.

[49] Bernard Merialdo. Tagging text with a probabilistic model. In *Proceedings of the IBM Natural Language ITL*, pages 161–172, Paris, France, March 1990.

[50] Bernard Merialdo. Tagging text with a probabilistic model. Technical Report RC 15972, IBM Research Division, 1990.

[51] Paul L. Meyer. *Introductory Probability and Statistical Applications.* Addison-Wesley, Reading, MA, 1970.

[52] S. Miller, M. Bates, R. Bobrow, R. Ingria, J. Makhoul, and R. Schwartz. Recent progress in hidden understanding models. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 276–280, Austin, TX, January 1995. Morgan Kaufmann Publishers, Inc.

[53] S. Miller, R. Schwartz, R. Bobrow, and R. Ingria. Hidden understanding models of natural language. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 25–32, Las Cruces, NM, June 1994. Morgan Kaufmann Publishers, Inc.

[54] S. Miller, R. Schwartz, R. Bobrow, and R. Ingria. Statistical language processing using hidden understanding models. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 278–182, Plainsboro, NJ, March 1994. Morgan Kaufmann Publishers, Inc.

[55] S. Miller, D. Stallard, R. Bobrow, and R. Schwartz. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 55–61, Santa Cruz, CA, June 1996. Morgan Kaufmann Publishers, Inc.

[56] M. Minsky. Introduction. In M. Minsky, editor, *Semantic Information Processing*, page 14. MIT Press, Cambridge, MA, 1968.

[57] R. Moore. Semantic evaluation for spoken-language systems. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 126–131, Plainsboro, NJ, March 1994. Morgan Kaufmann Publishers, Inc.

[58] A. Nadas and R. Mercer. Hidden markov models and some connections with artificial neural nets. In P. Smolensky, M. Moser, and D. Rumelhart, editors, *Mathematical Perspectives on Neural Networks*, pages 1–54. 1993.

[59] L. Norton, D. Dahl, and M. Linebarger. Recent improvements and benchmark results for the Paramax ATIS system. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 89–94, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[60] D. Pallett. DARPA ATIS test results. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 114–121, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

[61] D. Pallett. DARPA resource management and ATIS benchmark test poster session. In *Fourth DARPA Workshop on Speech and Natural Language*, pages 49–58, Pacific Grove, California, February 1991. Morgan Kaufmann Publishers, Inc.

[62] D. Pallett, J. Fiscus, W. Fisher, J. Garofolo, B. Lund, A. Martin, and M. Przybocki. 1994 benchmark tests for the ARPA spoken language program. In *Proceedings of the Spoken Language Systems Technology Workshop*, pages 5–36, Austin, TX, January 1995. Morgan Kaufmann Publishers, Inc.

[63] R. Pieraccini and E. Levin. Stochastic representation of semantic structure for speech understanding. In *EUROSPEECH 91*, pages 383–386, Genova, Italy, September 1991.

[64] R. Pieraccini and E. Levin. Stochastic representation of semantic structure for speech understanding. *Speech Communication*, 11:283–288, 1992.

[65] R. Pieraccini, E. Levin, and C. Lee. Stochastic representation of conceptual structure in the ATIS task. In *Fourth DARPA Workshop on Speech and Natural Language*, pages 121–124, Pacific Grove, California, February 1991. Morgan Kaufmann Publishers, Inc.

[66] R. Pieraccini, E. Tzoukermann, Z. Gorelov, E. Levin, C. Lee, and J. Gauvain. Progress report on the CHRONUS system: ATIS bench-

mark results. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 67–71, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[67] Stephen A. Della Pietra and Vincent J. Della Pietra. Statistical modelling with maximum entropy. Technical report, IBM Research Division, 1994.

[68] J. Polifroni, L. Hirschman, S. Seneff, and V. Zue. Experiments in evaluating interactive spoken language systems. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 28–33, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[69] P. Price. Evaluation of spoken language systems: the ATIS domain. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 91–95, Hidden Valley, PA, June 1990. Morgan Kaufmann Publishers, Inc.

[70] P. Price, L. Hirschman, E. Shriberg, and E. Wade. Subject-based evaluation measures for interactive spoken language systems. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 34–39, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[71] A. Ratnaparkhi, S. Roukos, and R. Todd Ward. A maximum entropy model for parsing. In *Proceedings of the 1994 International Confer-*

ence on Spoken Language Processing (ICSLP), volume 2, pages 803–806, Yokohama, Japan, September 1994. The Acoustical Society of Japan.

[72] Elaine Rich. *Artificial Intelligence.* McGraw-Hill, New York, NY, 1983.

[73] R. Rosenfeld. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach.* PhD thesis, Carnegie-Mellon University, April 1994.

[74] R. Schwartz and Y. Chow. The N-best algorithm: An efficient and exact procedrure for finding the N most likely sentence hypotheses. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 81–84, Albuquerque, NM, April 1990.

[75] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

[76] D. Sleator and D. Temperley. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Dept. of Computer Science, Carnegie Mellon University, 1991.

[77] Jeffrey D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, MD, 1982.

[78] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, 1967.

[79] S. Walter. Neal-Montgomery NLP system evaluation methodology. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 323–326, Harriman, NY, February 1992. Morgan Kaufmann Publishers, Inc.

[80] J. Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–44, Jan. 1966.

[81] Dominic Welsh. *Codes and Cryptogrophy*. Oxford Science Publications, Oxford, England, 1988.

[82] Patrick Henry Winston and Berthold Klaus Paul Horn. *LISP*. Addison-Wesley, Reading, MA, 1981.

AppendixA

Extended Backus-Naur Form Grammar for ATIS

This appendix gives an extended Backus-Naur Form [30] for most of NL-Parse. Uppercase words will denote non-terminals, lowercase words will denote terminals.

```
QUERY       ::= list (COLUMN of)? TABLE
                    (along with (COLUMN of)? TABLE) ? nl

COLUMN      ::= column_name (and column_name)*

TABLE       ::= (premodifier)* table_name (MODIFIERS)?
```

```
MODIFIERS  ::= (MODIFIERS or)? AND_PHRASE

AND_PHRASE ::= (AND_PHRASE and)? MOD_ITEM

MOD_ITEM   ::= ``(`` MODIFIERS ``)''
           |   table_operator TABLE
           |   terminal_operator terminal_value
```

The ambiguity in NL-Parse results from the fact that the MODIFIER strings do not require parentheses around table operators. Thus, a MOD_ITEM can modify any of the preceding table_names. Yacc points this out, as the grammar has 3 shift/reduce errors [1]. It resolves this by shifting the next modifier onto the parse stack, which has the effect of making modifiers modify the previously mentioned table. Note that most table_operators and terminal_operators can modify only one table. Thus, using yacc error recovery, one can undo the effect of an erroneous shift action. This was a lot of work, that could have been solved had the designers of NL-Parse made one minor modification, namely:

```
MOD_ITEM ::= ( table_operator TABLE )
```

That is, the parentheses encapsulate all the modifiers for a table, hence there is no ambiguity.

AppendixB

Yacc Grammar for ATIS

This chapter contains the yacc grammar for ATIS. I have omitted the semantic actions, some of which control the error recovery. Since the semantic actions are complex, I instead just include comments describing what needs to be done in the action. This way, the grammar is more readily visible. Nonterminals are written in lower case, and terminals in uppercase. The grammar includes some of the error recovery needed to handle the ambiguity in attaching modifiers to tables. Any post_modifier_item that recursively calls a table, will also have a second rule that uses "error". If an attachment error is discovered in processing the recursive table, it signals "YYERROR", which is caught by these rules.

```
corpora      : query | corpora query
```

```
query        : just LIST column_table NL
             | error NL
             {/*A syntax or semantic error was found*/}
             | NL

column_table : column table alongwith

alongwith    : /* empty */ | ALONG WITH column_table

column       : /* empty */ | column_last
             | column_start AND_T column_last

table        : flights | flight_legs | cities | days
             | date_days | class_of_serv | ground_serv
             | food_services | airport_serv | flight_stops
             | airports | fare_bases | aircraft
             | airlines_query | restrictions | fares
             | equipment_seq | dual_carriers | time_zones
             | intervals | comp_classes | column_tables
             | table_tables | states | months | code_desc
             | flight_fares

column_last  : THE NUMBER OF | column_item_number of_for
```

```
column_start : colitem_num | column_start AND_T colitem_num


colitem_num  : THE NUMBER OF column_token
             | max_min column_token
             | column_token


/* Table rules */
aircraft     : size AIRCRAFT {/*save table on stack*/}
fares        : pre_fltfare FARES {/*save table on stack*/}
flights      : pre_fltfare FLIGHTS {/*save table on stack*/}
cities       : CITIES {/*save table on stack*/}
             | city_state_country
flight_legs  : FLIGHT_LEGS {/*save table on stack*/}
days         : DAYS {/*save table on stack*/}
date_days    : DATE DAYS {/*save table on stack*/}
class_of_serv: CLASS OF SERVICES {/*save table on stack*/}
ground_serv  : GROUND SERVICES {/*save table on stack*/}
food_services: FOOD SERVICES {/*save table on stack*/}
airport_serv : AIRPORT SERVICES {/*save table on stack*/}
flight_stops : FLIGHT STOPS {/*save table on stack*/}
airports     : AIRPORTS {/*save table on stack*/}
             | airport
```

```
restrictions : RESTRICTIONS {/*save table on stack*/}
equipment_seq: EQUIPMENT SEQUENCES {/*save table on stack*/}
dual_carriers: DUAL CARRIERS {/*save table on stack*/}
time_zones   : TIME ZONES {/*save table on stack*/}
intervals    : INTERVALS {/*save table on stack*/}
comp_classes : COMPARTMENT_CLASSES {/*save table on stack*/}
column_tables: COLUMN TABLES {/*save table on stack*/}
table_tables : TABLE TABLES {/*save table on stack*/}
states       : STATES {/*save table on stack*/}
months       : MONTHS {/*save table on stack*/}
code_desc    : CODE_DESCRIPTIONS {/*save table on stack*/}
flight_fares : FLIGHT FARES {/*save table on stack*/}
airlines_query : AIRLINES {/*save table on stack*/}
fare_bases   : FARE BASES {/*save table on stack*/}

suffix       : grouped_by ordered_by post_modifiers

ordered_by   : /* empty */
             | ORDERED updown BY night_coltok_list

grouped_by   : /* empty */
             | GROUPED_T BY night_coltok_list
             | MAXIMUM GROUPED_T BY night_coltok_list
```

```
night_coltok_list : night_col_tok
             | NUMBER_T
             | LPAREN night_coltok_list2 RPAREN

night_coltok_list2 : night_col_tok
             | NUMBER_T
             | night_coltok_list2 AND_T night_col_tok
             | night_coltok_list2 AND_T NUMBER_T

/* Some pre-table modifiers */
size         : /* empty */ | SMALLEST | LARGEST | FASTEST
             | SLOWEST | LIGHTEST | HEAVIEST
             | LONGEST_AIRCRAFT | SHORTEST_AIRCRAFT
pre_fltfare  : pre_fltfare2
             | pre_fltfare2 airlines pre_fltfare2
pre_fltfare2 : /* empty */
             | pre_fltfare2 pre_fltfare_item
pre_fltfare_item : pre_cost pre_dir | pre_time | class
             | SHORTEST | LONGEST | CHEAP
pre_time     : MORNING | AFTERNOON | EARLIEST | LATEST
             | LATEST ARRIVING | LATEST DEPARTING
             | LATE_AFTERNOON | LATE_MORNING | LATE_NIGHT
```

```
              | LATE_EVENING | EARLIEST ARRIVING
              | EARLIEST DEPARTING | EARLY_MORNING | AM
              | PM | EARLY | EARLY_AFTERNOON | EVENING
              | DAILY | OVERNIGHT | NIGHT | REDEYE | DAY
              | MID_MORNING | MID_AFTERNOON | MIDDAY | LATE
pre_cost      : /* empty */ | CHEAPEST | MOST EXPENSIVE
pre_dir       : ONE_DIRECTION | ONE_WAY | ROUND TRIP
              | NONSTOP | DIRECT | CONNECTING

/* Some time related rules */
arrival_time_meal : meal | arrival_time
arr_time_del : /* empty */ | arrival_time
arrival_time : MORNING | AFTERNOON | LATE_AFTERNOON
              | LATE_MORNING | LATE_EVENING | AM | PM
              | EARLY_MORNING | EARLY | EARLY_AFTERNOON
              | EVENING | NIGHT | COLUMN_T | DAY
              | MID_MORNING | MID_AFTERNOON | MIDDAY | LATE

/* The Boolean rules for standard operator precedence */
post_modifiers : /* empty */
                 {/*pop table off of stack*/}
              | post_modifier_expr
                {/*pop table off of stack*/}
```

```
post_modifier_expr : post_modifier_expr OR_T
                     {/*save some error recovery info here*/}
                     post_modifier_term
                     {/*save modifier table as the logical
                        AND of the modifier tables for
                        post_modifier_expr and
                        post_modifier_term.*/
                      /*save some error recovery info here*/}
                   | post_modifier_term
                     {/*set modifier table as the modifier table of
                        post_modifier_term*/
                      /*save some error recovery info here*/}

post_modifier_term : post_modifier_term AND_T
                     {/*save some error recovery info here*/}
                     post_modifier_factor
                     {/*save modifier table as the logical
                        AND of the modifier tables for
                        post_modifier_term and
                        post_modifier_factor.*/
                      /*save some error recovery info here*/}
                   | post_modifier_factor
```

```
                {/*set modifier table as the modifier table of
                   post_modifier_factor*/
                 /*save some error recovery info here*/}

post_modifier_factor : not LPAREN
                       {/*save some error recovery info here*/}
                       post_modifier_expr RPAREN
                       {/*save some error recovery info here*/
                        /*set modifier table as the modifier
                          table of the post_modifier_expr*/
                        /*See if the modifier table for the
                          post_modifier_expr matches the
                          table at the top of the stack.
                          If not, raise an error.*/}
              | not post_modifier_item
                {/*save some error recovery info here*/
                 /*set modifier table as the modifier table
                   of the post_modifier_expr*/
                 /*See if the modifier table for the
                   post_modifier_expr matches the table
                   at the top of the stack.
                   If not, raise an error.*/}
```

```
/* The post table modifiers */
post_modifier_item : ASSOCIATED WITH class_of_serv
                     {/*save the modifier table that this
                        can modify in this post_modifier_item
                        and all the follow*/}
                   | ASSOCIATED WITH error
                     {/*in this post_modifier_item and all that
                        follow that contain ``error'' in the
                        rule, perform error recovery.  These
                        rules are needed for post_modifier_items
                        that recurse into a table.*/}
                   | ASSOCIATED WITH class2 SERVICE
                   | ASSOCIATED WITH fare_bases
                   | WITH ARRIVALS ON WORD_T
                   | WITH ARRIVALS ON day_modifier
                   | AVAILABLE ON WORD_T
                   | AVAILABLE ON WORD_T also_else WORD_T
                   | AVAILABLE ON day_modifier
                   | AVAILABLE ON day_modifier also_else
                        day_modifier
                   | AVAILABLE ON days
                   | AVAILABLE ON error
                   | AVAILABLE FOR fares
```

```
| AVAILABLE FOR fare_bases

| AVAILABLE FOR error

| BELONGING TO fares

| BELONGING TO error

| CHARGED FOR flights

| CHARGED FOR error

| EQUIPPED WITH aircraft

| EQUIPPED WITH error

| EQUIPPING flights

| EQUIPPING error

| FLYING ON WORD_T flying_on_del

| FLYING ON day_modifier

| FLYING ON day_modifier also_else day_modifier

| ARRIVING ON WORD_T

| ARRIVING ON day_modifier

| FLYING ON days

| FLYING ON error

| FOR flight_stops

| FOR error

| FOR flights

| HAVING PRICES OF fares

| HAVING PRICES OF error

| HAVING restrictions
```

```
| HAVING error
| CONTAINING cities
| CONTAINING airports
| CONTAINING error
| FOUND IN time_zones
| FOUND IN error
| IN cities
| IN airports
| IN error
| LOCATED IN states
| arr_dep IN MONTH_T
| NUMBERED comp_op_del NUMBER_T
| LOCATED IN error
| OF class2 SERVICE
| OFFERED BY flights
| OFFERED BY error
| OFFERING class2 SERVICE
| OFFERING fare_bases
| OFFERING error
| ON days
| ON error
| PRIMARILY SERVED BY airlines_query
| PRIMARILY SERVED BY error
```

```
| PROVIDED for_by airports
| PROVIDED for_by error
| PROVIDED FOR cities
| PROVIDED WITH airport_serv
| PROVIDED WITH ground_serv
| PROVIDED WITH error
| PROVIDING airport_serv
| PROVIDING error
| SECONDARILY SERVED BY airlines_query
| SECONDARILY SERVED BY error
| SERVED BY airlines_query
| SERVED BY airports
| SERVED BY error
| only SERVING food_services
| SERVED ON flights
| SERVED ON error
| only SERVING meal
| only SERVING error
| only SERVING cities
| only SERVING flights
| only STOPPING IN airports
| only STOPPING IN cities
| only STOPPING IN error
```

```
| THAT ARE LEGS FOR flights
| THAT ARE LEGS FOR error
| THAT ARE STOPS FOR flights
| THAT ARE STOPS FOR error
| from_to airports
| WHOSE night_col_tok IS A STRING not
      CONTAINING number_misc_string
| WHOSE night_col_tok IS BETWEEN NUMBER_T
      arr_time_del AND_T NUMBER_T arr_time_del
| WHOSE night_col_tok IS comp_op_del max_min
      night_col_tok OF table
| WHOSE night_col_tok IS comp_op_del max_min
      night_col_tok OF error
| WHOSE night_col_tok IS comp_op_del NUMBER_T
| WHOSE night_col_tok IS not_yes_no
| WHOSE night_col_tok IS not misc_string
| SCHEDULED FOR flights
| SCHEDULED FOR flight_stops
| SCHEDULED FOR error
| WITH LEGS THAT ARE flights
| WITH LEGS THAT ARE error
| WITH SCHEDULED flight_stops
| WITH SCHEDULED error
```

```
| WITH arr_dep flights
| WITH arr_dep fares
| WITH arr_dep error
| from_to cities
| from_to error
| ABBREVIATED number_misc_string
| NAMED misc_string
| NAMED A STRING not CONTAINING
      number_misc_string
| NAMED comp_op_del ANY_T night_col_tok OF table
| NAMED comp_op_del ANY_T night_col_tok OF error
| airlines
| CHEAP
| arr_dep BETWEEN NUMBER_T arr_time_del AND_T
      NUMBER_T arr_time_del
| arr_dep time_op NUMBER_T arr_time_del
      flying_on_del
| arr_dep time_op NUMBER_T arr_time_del
      also_else time_op NUMBER_T arr_time_del
| arr_dep time_op_del arrival_time_meal
| class2
| pre_dir
| column_token
```

```
            | pre_time

/* City rules */
city_state_country : city_state | city_state COUNTRY_T
city_state   : CITY_T | CITY_T STATE_T | STATE_T

/* String rules */
misc_string  : city_state_country | class2 | DAILY
             | CLASSES | FIRST_T
number_misc_string : NUMBER_T | misc_string
             | FARE | FLIGHT | COLUMN_T | string3
             | AIRPORT_T | meal
string3      : WORD_T | WORD_T WORD_T
             | WORD_T WORD_T WORD_T | QUOTE_T

/* Classes */
class        : COACH | COACH CLASS | FIRST_T CLASS
             | ECONOMY_CLASS | THRIFT ECONOMY_CLASS
             | THRIFT CLASS | BUSINESS_CLASS
class2       : class | COACH ECONOMY_CLASS

/* Operators */
comp_op      : not GREATER THAN | not EQUAL TO
```

```
              | not LESS THAN | not LESS THAN OR_T EQUAL TO
              | not GREATER THAN OR_T EQUAL TO
comp_op_del   : not | comp_op
time_op_del   : /* empty */ | time_op
time_op       : BEFORE | AFTER | FROM | AT | BY | AROUND

/* Miscellaneous */
not           : /* empty */ | NOT
only          : /* empty */ | ONLY
also_else     : ALSO | ELSE
night_col_tok: NIGHT | column_token
column_token : COLUMN_T | STOPS | COLUMNS
not_yes_no    : not YES | not NO | not KNOWN | not UNKNOWN
from_to       : FROM | TO
for_by        : FOR | BY
airport       : AIRPORT_T
just          : /* empty */ | JUST
of_for        : OF | FOR
updown        : /* empty */ | UP | DOWN
airlines      : string3
day_modifier : TODAY plus_minus NUMBER_T | TODAY
flying_on_del: /* empty */ | also_else else_chain
else_chain    : also_chain | else_chain ELSE also_chain
```

```
also_chain   : chain_item | also_chain ALSO chain_item

chain_item   : WORD_T | pre_time

plus_minus   : PLUS | MINUS

meal         : BREAKFAST | LUNCH | DINNER | SNACK

max_min      : THE | ANY_T | ALL_T | THE MAXIMUM
             | THE MINIMUM | THE AVERAGE

arr_dep      : ARRIVING | DEPARTING
             | ARRIVING AND_T DEPARTING
             | ARRIVING OR_T DEPARTING | LEAVING
```

AppendixC

More Decoding Results for Model 1

This appendix contains a few more results for model 1. It is included to facilitate comparison with the maximum likelihood results for the smoothed and unsmoothed parameters presented in chapter 5. This contains the Viterbi decoding results for the smoothed and unsmoothed parameters. It also presents the unsmoothed results after just 2 iterations of the EM algorithm are run. As in chapter 5, the results are for DEV94, which has 410 class A sentences.

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	265	266	263	261	258	258	258	257	255
2813	266	264	260	262	261	259	257	258	254
1406	277	273	268	261	257	246	241	249	245
703	278	274	267	266	255	244	246	241	244
351	275	276	256	253	243	231	191	203	196
175	272	267	265	259	238	232	194	213	201
87	272	274	264	260	232	221	156	170	157
43	271	267	263	257	232	215	182	175	140
0	271	268	265	259	223	187	119	142	154

Table C.1: Exact Match Viterbi Decoding Results for DEV94 Using Smoothed Model 1 as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	265	261	254	254	253	254	251	251	249
2813	263	259	251	252	252	251	246	250	241
1406	280	270	260	257	240	240	237	245	236
703	275	265	258	254	231	224	221	220	216
351	274	271	250	244	224	207	154	184	133
175	273	266	253	244	210	206	149	193	113
87	N/A	269	253	248	214	192	114	135	84
43	272	266	254	240	203	189	110	143	49
0	272	269	260	244	204	161	71	126	152

Table C.2: Exact Match Viterbi Decoding Results for DEV94 Using 512 Iterations of Model 1, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	266	264	264	254	251	244	243	242	242
2813	267	263	261	261	260	249	243	244	241
1406	276	272	267	262	258	244	231	224	221
703	275	266	262	256	249	246	236	226	211
351	272	269	249	243	228	226	190	205	158
175	272	267	251	246	213	217	177	201	153
87	274	269	253	243	227	205	133	155	83
43	272	269	254	241	208	190	139	152	80
0	272	269	260	244	204	161	71	126	152

Table C.3: Exact Match Viterbi Decoding Results for DEV94 Using 2 Iterations of Model 1, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	280	272	273	267	269	269	264	259	259
2813	279	276	273	275	271	268	267	264	261
1406	280	269	266	258	257	254	250	247	247
703	274	267	261	259	245	238	237	238	228
351	280	277	254	248	239	229	206	213	201
175	271	265	254	250	221	218	196	205	175
87	278	276	259	246	216	195	136	157	113
43	277	270	263	244	215	185	128	152	88
0	275	269	263	248	207	165	70	112	154

Table C.4: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 2 Iterations of Model 1, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	6.29	6.39	7.83	7.79	8.37	8.16	8.71	8.69	8.72
2813	5.49	5.77	7.14	7.99	8.19	8.44	8.02	8.82	8.78
1406	5.59	5.83	6.22	7.02	9.85	8.49	10.59	10.96	11.17
703	5.62	6.64	5.84	6.55	8.94	10.17	11.37	11.70	12.30
351	5.54	6.02	5.87	6.97	8.83	11.23	13.18	13.08	16.38
175	6.13	6.06	6.44	7.25	9.02	9.08	11.80	10.51	24.11
87	N/A	5.51	5.51	5.72	6.57	6.42	8.63	11.52	13.65
43	5.80	5.57	5.53	5.79	5.69	6.68	6.57	7.18	16.12
0	5.55	5.54	5.57	5.58	5.66	6.18	6.58	6.81	10.62

Table C.5: Maximum Likelihood Cross Entropy for DEV94 Using 512 Iterations of Model 1, as a Function of Amount of Hand and Hidden Training Data

AppendixD

More Decoding Results for Models A, AHW, and ALM

This appendix contains a few more results for models A, AHW, and ALM. This is primarily so one can see the effect of allowing all the model A parameters to vary versus keeping one set of parameters fixed.

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	283	277	270	273	266	267	264	264	260
2813	282	279	269	265	261	259	253	257	256
1406	277	276	266	266	245	242	229	235	223
703	271	268	255	252	231	228	213	221	196
351	274	274	260	256	230	202	184	200	131
175	276	270	252	250	219	199	147	179	101
87	274	272	254	246	208	173	139	124	82
43	273	273	254	255	222	178	138	139	19
0	272	271	257	251	214	176	103	122	152

Table D.1: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 20 Iterations of Model A, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	283	279	268	269	264	259	259	263	244
2813	283	280	271	254	256	261	253	249	256
1406	283	276	261	272	245	243	237	240	232
703	276	266	251	255	234	224	211	211	187
351	275	272	259	259	233	196	185	205	119
175	276	272	250	251	217	196	145	179	89
87	276	272	254	245	209	173	138	124	81
43	273	271	254	253	220	181	129	132	23
0	273	271	257	251	214	176	103	122	152

Table D.2: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 25 Iterations of Model A, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	286	280	278	277	277	279	269	272	276
2813	283	280	275	268	267	269	264	255	248
1406	279	277	260	256	239	245	211	240	222
703	269	264	251	255	225	202	202	209	185
351	272	271	256	260	220	182	177	196	115
175	271	271	260	253	216	182	138	176	100
87	268	270	252	242	215	156	135	108	64
43	272	269	255	248	217	178	134	103	19
0	273	270	260	246	218	151	98	95	152

Table D.3: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 20 Iterations of Model AHW, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	280	261	264	277	271	258	259	259	262
2813	256	254	263	270	264	260	261	247	244
1406	260	258	258	253	234	235	212	223	217
703	246	236	251	258	229	207	190	202	192
351	249	234	259	259	226	180	172	197	108
175	254	255	260	253	214	181	133	171	95
87	240	240	257	244	217	156	131	99	62
43	258	254	258	251	212	160	131	104	19
0	257	259	261	251	218	150	98	95	152

Table D.4: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 25 Iterations of Model AHW, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	275	268	264	271	268	268	262	262	258
2813	276	268	265	250	246	252	244	258	242
1406	271	269	260	250	231	233	227	232	219
703	269	262	250	237	220	204	184	193	160
351	269	265	252	236	211	182	155	182	95
175	269	260	244	236	204	170	108	141	85
87	269	262	246	235	205	146	103	83	64
43	267	259	241	233	202	144	116	57	11
0	268	262	251	238	203	140	78	62	0

Table D.5: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 20 Iterations of Model ALM, as a Function of Amount of Hand and Hidden Training Data

Hidden Amt	Hand Amt								
	3575	**1787**	**893**	**446**	**223**	**111**	**55**	**27**	**0**
5627	274	262	258	268	263	262	255	257	259
2813	274	264	260	247	242	249	236	254	245
1406	272	269	259	250	225	231	228	235	215
703	268	261	248	232	218	201	182	192	174
351	269	262	249	231	209	182	153	177	89
175	269	260	244	234	204	165	99	137	82
87	269	261	246	232	205	149	99	81	63
43	267	259	241	233	202	143	115	55	10
0	268	262	251	238	203	140	78	62	0

Table D.6: Exact Match Maximum Likelihood Decoding Results for DEV94 Using 25 Iterations of Model ALM, as a Function of Amount of Hand and Hidden Training Data

AppendixE

Errors Made in DEV94 Not Due to the Smoothed Model B Deficiencies

This appendix shows the errors made in DEV94 that were not due to a problem with smoothed model B. The prefix "E:" means "English", "A:" means the correct answer, and "P:" means the pattern matcher answer. If there are two "E:" lines, this means that one is the tagged English, and the other is the untagged English. These results were generating using a maximum likelihood decoding. This makes it hard to know why a particular formal language word is included. One would have to examine all possible alignments. Instead, one can use the Viterbi alignment to find the most likely

formal language word to generate each English word. While this may not be the cause of the error, it often gives insight as to why a wrong pattern was selected.

Tagger Error

This error occurs when the English tagger makes an error. In DEV94, this happened in three sentences.

E: i need a flight from CITY_1 to CITY_2 on AIR_1 that gets into CITY_2 at about NUM_1 in the evening

A: List flights AIR_1 arriving around TIME_1 from:city CITY_1 to:city CITY_2

P: List flights AIR_1 evening flying-on today+1 from:city CITY_1 numbered NUM_1 serving:meal dinner to:city CITY_2

In this case, NUM_1 should be a TIME_1.

E: how much does the flight m g NUM_1 cost from CITY_1 to CITY_2 on DAY_1 morning

E: how much does the flight m g three hundred cost from new york to los angeles on monday morning

A: List fares ACODE_1 morning available-on DAY_1 from:city CITY_1 numbered NUM_1 to:city CITY_2

P: List fares morning available-on DAY_1 from:city CITY_1 numbered NUM_1 to:city CITY_2

In this case, "m g" should be an ACODE. Since it is not, a pattern is found that contains just one NUM, two CITY, and one DAY tags.

Language Model Error

This error occurs when the pattern matcher finds an incorrect pattern whose language model score is better than the reference answer, and whose translation model score is worse than the reference answer. Thus, the translation model prefers the correct answer, but the language model causes the wrong answer to be selected. This happened in 16 errors, which is 25% of the errors. Though it should be pointed out that in many of these, if the translation model had an even higher score for the correct answer, this could compensate for the language model. This is tough to diagnose without having a real language model. One might suggest decreasing the language model weight (see section 4.4), but this hurt results. The language model, while often a problem, more often helps.

E: i need to fly from CITY_1 to CITY_2 one-way what are the flights

A: List flights one_way from:city CITY_1 to:city CITY_2

P: List flights cheapest one_way from:city CITY_1 to:city CITY_2

In this case, the language model notices that when "one_way" usually appears, so does "cheapest". However, in five sentences, the word "cheapest" did not generate any words, and hence the alignment was the exact same as if "cheapest" was not there. If "cheapest" generated a word, then the translation model would penalize it. The language model score for the wrong answer is preferred .00525 to .00170. The translation model score prefers the correct answer by 1.5×10^{-20} to 7.53×10^{-21}.

E: please show me which flights fly from CITY_1 to CITY_2 between TIME_1 and TIME_2 next DAY_1

A: List flights departing between TIME_1 TIME_2 flying-on DAY_1 from:city CITY_1 to:city CITY_2

P: List flights arriving between TIME_1 TIME_2 flying-on DAY_1 from:city CITY_1 to:city CITY_2

It is not entirely fair to blame the language model for not being able to distinguish between "arriving" and "departing", as arriving generates nothing in the incorrect answer. If there were a high penalty for adding formal words with 0 fertility, then this would not be found. The language model score for the wrong answer is preferred .00085 to .00038. The overall weighted score prefers the wrong answer by 5.13×10^{-15} to 4.00×10^{-15}. This error happened in two sentences.

E: show AIR_1 flights to CITY_1

A: List flights AIR_1 to:city CITY_1

P: List flights AIR_1 from:city CITY_1

In this case, the correct answer is 2 times more likely for the correct answer, but the language model is 3 times more likely for the wrong answer. This error occurred twice.

E: please list airfares from CITY_1 to CITY_2

A: List fares from:city CITY_1 to:city CITY_2

P: List flights from:city CITY_1 to:city CITY_2

In three sentences, the pattern matcher incorrectly selected a flight query to a fare query. In this case, the LM is 8 times stronger for flights than fares, and this outweighs the 6 times score the translation model gives to the correct answer.

E: show the flights from CITY_1 to CITY_2 and show the price

A: List along-with flights from:city CITY_1 to:city CITY_2 fares

P: List flights from:city CITY_1 to:city CITY_2

In four sentences, the pattern matcher made an error on queries that asked for both flights and fares. The translation model is four times more likely for the reference answer, but the wrong pattern is 8 times more likely. In one sentence, the word "costs" was used. The probability of generating "costs"

from "fares" was 3 times higher than generating "costs" from "List", but one would expect this to be higher. Unfortunately, the word "costs" is rare. If one tied together "cost" and "costs", this would solve some of the spurious effects of rare words.

Bad Formal Language Design

This error is due to a bad formal language representation, for one of many possible reasons:

- The formal language contains too many words to represent a single semantic concept. Thus, the generative assumption is invalidated, and the model does not know how to distinguish between two similar patterns. This happens in the first and third examples in the section.
- The formal language contains semantically relevant words that do not generate any semantic concepts. This happens in the second example in this section.

These errors make it clear that it is necessary to have the formal language be as close to the English as possible, yet still be unambiguous and easy to convert to an SQL query. NL-Parse is close, it just has a few problems that need to be fixed.

E: are there any flights from CITY_1 to CITY_2 that arrive within thirty minutes of TIME_1 next DAY_1

A: List flights arriving around TIME_1 arriving on DAY_1 from:city CITY_1 to:city CITY_2

P: List flights arriving around TIME_1 flying-on DAY_1 from:city CITY_1 to:city CITY_2

In this one sentence, the formal language has a deficiency. The NL-Parse requires each departure or arrival day, date, and time to be preceded by either "arriving" or "departing/flying". This makes it impossible for the model to know how to generate English sentences that contain only one word to indicate a departure or arrival. The formal language needs to be fixed so that one can have "List flights arriving around TIME_1 DAY_1 DATE_1". Note that unless the flight is an overnight flight that does not fly all days of the week, and the user happened to pick the day for which this did not apply, then this is not going to be a CAS error.

E: tell me what a CODE_1 is

E: tell me what a d nine s is

A: List Extract aircraft abbreviated CODE_1 Features all aircraft_desc

P: List Extract fare-bases abbreviated CODE_1 Features all entries

In this case, "d nine s" is properly tagged as a CODE. But there is no way to distinguish between aircraft and fare bases codes. If the user said "tell me what aircraft d nine s is", then all would be well. This deficiency can be

fixed in a variety of ways. One way is to have a different type of code for each table. Another is to have a generic formal language pattern that contains "Define CODE_1", and the back end can decide if the query is about aircraft or fare bases.

E: please list flights for CITY_1 from AIR_1

A: List flights AIR_1 or from:city CITY_1 to:city CITY_1

P: List flights AIR_1 from:city CITY_1

In this sentence, the annotator decided that a flight "for" a city meant that it could be either to or from the city. Extra formal language nodes, "or", "from:city", and CITY_1 make it hard for the translation model, which adds penalties when formal language words do not generate English words. If indeed this concept needs to be handled, then one should have a "from-or-to:city" formal language node, and this can be used. If it is used enough in the training data, then it will train properly. One could reasonably justify the claim that the formal language is wrong, and that the correct formal language should be "List flights AIR_1 to:city CITY_1".

E: please show me all flights from CITY_1 to CITY_2 and return

A: List flights or from:city CITY_1 to:city CITY_2 from:city CITY_2 to:city CITY_1

P: List flights from:city CITY_1 to:city CITY_2

The formal language has no concept of a return flight. Hence, English queries necessitate verbose formal language. This makes it hard for a generative translation model. In this case, the LM also gives strong preference to the wrong answer, as it is 350 times more likely. If the formal language had a "return" word in it, then perhaps $p(return|return)$ might be strong enough to overcome the language model. The back end would have to implement this though.

E: how much is the fare limousine CITY_1 to downtown

A: List Extract ground-services equal-to transport_type LIMOUSINE provided-for:airports airports serving:city CITY_1 provided-for:city CITY_1 Features all ground_fare

P: List Extract ground-services equal-to transport_type LIMOUSINE provided-for:city CITY_1 Features all ground_fare

In this case, the correct answer contains redundant formal language in it. The CAS answer would produce the correct result.

Permutation Error

This is an interesting type of error. Since all the models presented in this thesis use just the identities of the formal language words to model the generation of English words, the words can be entirely rearranged. In particular, if one reverses the usual meaning of CITY_1 and CITY_2, the translation

model will still predict the exact same score. The pattern matcher will then pick the more likely pattern according to the language model. The only way to overcome this is to model the order in which clumps are generated. This can be done by distortions[13], or by n-gram modeling in the meaning space, as done by AT&T[63] and BBN[54]. This type of error happened eight times.

E: i want to go to CITY_1 on DATE_1 leaving from CITY_2

A: List flights flying-on DATE_1 from:city CITY_2 to:city CITY_1

P: List flights flying-on DATE_1 from:city CITY_1 to:city CITY_2

E: CITY_1 to CITY_2 and then to CITY_3

A: List flights from:city CITY_1 stopping-in:city CITY_2 to:city CITY_3

P: List flights from:city CITY_1 stopping-in:city CITY_3 to:city CITY_2

Notes:

Bad Class

E: how much do those NUM_1 flights cost

A: List fares arriving between TIME_1 TIME_2 from:city CITY_1 to:city CITY_2

P: List fares equal-to flight_id NUM_1

In this lone sentence, the DEV94 annotators called this a context independent sentence. Obviously wrong. The pattern found by the pattern matcher, curiously enough, is a reasonable interpretation for the query. If you were talking to a travel agent, and they told you "there are 3 United flights, 2313, 2314, and 2441, which are you interested in?" Your response, "...how much do those 2314 flights cost?"

Bad City/Airport Tag

This type of error is due to an unfortunate design decision made years ago, when ATIS consisted of only one airport per city. The decision was made to tag expressions like "the Indianapolis airport" with an ARP tag. It would be better to tag this with "the CITY airport". The problem is that there is usually more than one airport servicing a city. Thus, the English tagger produces an ARP tag incorrectly, though by design. The reference answer properly uses a CITY tag. But since English tags must be generated by the same formal tag, the pattern matcher finds a pattern that matches. As it happens, this will produce the same CAS answer if there is only one airport for the specified city. This happened in four sentences.

E: how do i get from ARP_1 into the city

E: how do i get from the indianapolis airport into the city

A: List ground-services provided-for:airports airports serving:city CITY_1 provided-for:city CITY_1

P: List ground-services provided-for:airport ARP_1