Maximum Entropy Language Modeling with Non-Local Dependencies

Jun Wu

A dissertation submitted to the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland
2002

Abstract

Stochastic models of natural language are an important component in a wide variety of natural language processing applications, such as automatic speech recognition and machine translation. A language model is a probability measure on word-sequences in a language and for most practical purposes takes the form of a left-to-right conditional probability of the "next" word w_k in a sentence given the preceding "word-history" w_1, w_2, ... , w_{k-1}. The probabilities are estimated from sample text, and the most widely used models are N-gram models, which treat word sequences as a Markov process and predict w_k from the preceding N-1 words. For reasons of data sparseness, N is typically 2-4. Due to this Markov assumption, N-gram models successfully "learn" frequent phrases and local lexical dependencies but fail to capture syntactic well-formedness in sentences and semantic coherence within and across sentences.

To improve the performance of language models, two critical problems need to be solved: first, deciding what kinds of long-range dependence in natural language should be used in langauge models, and second, determining how dependencies from different information sources can be incorporated in a sound probability model. In this dissertation, a new language model is presented that overcomes some of the shortcomings of N-gram model by combining collocational dependencies with two important long-range dependence sources of: the syntactic structure of a sentence and the topic of a discourse. Maximum entropy techniques, which are particularly well suited for modeling diverse sources of statistical dependence, are used.

Previously known parameter estimation procedures for maximum entropy models have a computational cost that makes them impractical for most large-scale applications, including the two language modeling tasks mentioned above. Some fundamental algorithmic improvements in the parameter estimation procedure for maximum entropy models are presented: a hierarchical decomposition of computation and a divide-and-conquer strategy. The computational complexity of the parameter estimation algorithm in these tasks is reduced by 2-3 orders of magnitude using these improvements, which are generally useful for all applications of maximum entropy models.

Significant improvements due to the new language model over a trigram model are demonstrated in perplexity and in word error rate for the automatic transcription of spontaneous telephone conversations (Switchboard) and of radio and television programs (Broadcast News). An effort is also made to understand the role of each kind of long-range dependency in improving performance . A breakdown of performance on different types of words or histories in sentences is given. The breakdown results show that topic information is most helpful on content-bearing words, and syntactic structure is more useful when meaningful predicting words cannot be captured by N-grams. Experimental results on both Switchboard and Broadcast News show that the topic dependence and the syntactic dependence are complementary and their gains are nearly additive. A comparison of maximum entropy models with other models proposed in the literature is provided throughtout the dissertation.

Acknowledgements

The ideal research center for Ph.D. students, in my opinion, should be a place with many bright people and providing students the freedom of working on their own interesting topics. Center for Language and Speech Processing (CLSP) at The Johns Hopkins University is no doubt one of the best labs in the world from this point of view. I thank all people in CLSP, the department of computer science and elsewhere who have helped me to accomplish my Ph.D. program. This dissertation work would not have been successfully completed without the help and support of my advisor, colleagues and family.

First and foremost, I would like to thank my advisor, Sanjeev Khudanpur, who consistently gave me good advice and also the freedom to pursue what I believe to be the best way to approach the optimal solutions in the world of speech recognition and natural language processing. Sanjeev was an outstanding advisor, who was not only one of the most talent scientists in the speech recognition society but also an excellent mentor teaching me methods in scientific research and helping me improving presentation and writing skills . It was he who brought me to the beautiful world of maximum entropy and led me towards success in this area. I also thank Sanjeev for many valuable suggestions, in both personal and professional during my study in Hopkins.

I would like to thank David Yarowsky, who was my academic advisor in the department of computer science for many years, for his incredible commitment to me and my study. As one of the investigators in the NSF STIMULATE project which I worked for, David gave me many valuable advice and help. His support and help was extremely important for me to complete the program.

I would like to thank Fred Jelinek, the director of CLSP, who provided me the best research environment where I had the chance to meet and discuss with best scientists in the world. The first time I understood speech recognition was by reading his papers when I was in China. An important reason I chose Hopkins from many universities was because of his reputation. I got many valuable suggestions and comments about my dissertation research from Fred.

I would like to thank Eric Brill, Gerald Masson, Jason Eisner and Jan Hajik for their support and help. In particular, Eric, who was my first academic advisor in Hopkins, advised me on one of my qualification projects.

I would like to thank Ciprian Chelba, Radu Florian, Peng Xu, Shankar Kumar, Woosung Kim, John Henderson, Gideon Mann, Grace Ngai, Lidia Mangu, Vlasios Doumpiotis, Silviu-Petru Cucerzan, Veera Venkataramani, Murat Saraclar, Ahmad Emami, Richard Wicentowski, Charles Schafer, Noah Smith and many other students from the department of computer science and CLSP, from whom I got many valuable suggestions and feedbacks in my research.

I would like to thank Laura Graham, Nancy Scheeler, Linda Rorke, Conway Benishek, Kimberly Petropoulos and Amy Berdann for making administrative matters run efficiently , and Jacob Laderman for making our computer systems work smoothly.

I would like to thank Roberto Pieraccini, Esther Levin, Allen Gorin and other members of AT&T Research, where I worked as an intern in 1997, for their help in my Ph.D. program and career.

No research today can be accomplished without cooperating with other people. Many tools and data I used in my research were provided by many people inside and outside Hopkins. In particular, Ciprian Chelba and Fred Jelinek provided the SLM parser and Peng Xu helped me generating data for training syntactic models; Radu Florian and David Yarowsky provided the topic detection algorithms and the initial classification data for Broadcast News; Shankar Kumar, Vlasios Doumpiotis , William Byrne and Dimitra Vergyri provided the Nbest lists and lattices for rescoring; Andreas Stolcke provided the SRI LM toolkits; Michael Riley provided the AT&T

FSM decoder; Adwait Ratnaparkhi provided the ME part-of-speech tagger. I owe them many thanks.

I would also like to thank Google Inc, which provided me opportunities to apply my knowledge to real applications and supported me to finish my dissertation after my leaving school.

Most of all, I thank my family for their role in my education. I thank my parents for teaching me the value of education at a young age and instilling in me a desire for higher education, and my brother, Zining, for his help of all kinds when I studied in the states.

Finally, I cannot thank enough my beloved wife Yan for her consistent support, encouragement and love throughout the past six years.

This dissertation work was partly sponsored by NSF STIMULATE grant.

Yan Zhang

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We begin with an introduction to language modeling, the maximum entropy principle, non-local dependencies and measures of language model performance that will be used throughout this dissertation. Work related to this dissertation is also addressed here. Readers who are familiar with language modeling may skip Sections 1, 2 and 5 of this chapter.

1.1 Language Modeling

Language modeling is a crucial technique in automatic speech recognition (ASR), optical character recognition (OCR), handwriting recognition, machine translation (MT) and spelling correction. In a statistical system that converts, *e.g.*, speech A, to text W, the most probable string $\hat{W}$ is sought to maximize the posterior probability $p(W|A)$,

$$\hat{W} = \arg\max_{W} p(W|A).$$

Applying Bayes rule, we rewrite the formula above as

$$\hat{W} = \arg\max_{W} \frac{p(A|W) \cdot p(W)}{p(A)} = \arg\max_{W} p(A|W) \cdot p(W)$$

where $p(A)$ may be ignored because it is the marginal probability of A. The model to estimate the probability $p(A|W)$ is called the acoustic model in speech recognition; the model for estimating the probability $p(W)$ is called the language model.

Let $W = w_1, w_2, \cdots, w_L$, where w_i is the i^{th} word in the word string (sentence) W, and L is the length in words of W. $p(W)$ can be decomposed as

$$p(W) = p(w_1, w_2, \cdots, w_L) = \prod_{i=1}^{L} p(w_i|w_1^{i-1})$$

using the chain rule. Sometimes, $w_1^{i-1} = w_1, w_2, \cdots, w_{i-1}$ is called the *history* of w_i and is denoted by h_i.

In large vocabulary speech recognition systems, the probability $p(w_i|w_1^{i-1})$ is difficult to estimate non-parametrically even for a moderate value of i, due to limitations on memory, computation and available data. Therefore, histories h_i are clustered into history classes $\phi(h_i)$ with a manageable size.

1.2 N-gram Models

The most commonly used language modeling technique represents the word string W as a Markov chain - the word w_i depends on at most $N-1$ preceding words $w_{i-N+1}, \cdots, w_{i-1}$, *i.e.*, $\phi(h_i) = w_{i-N+1}, \cdots, w_{i-1}$. $p(W)$ is then approximated as

$$p(W) \approx p(w_1) \cdot p(w_2|w_1) \cdots p(w_{N-1}|w_1, \cdots, w_{N-2}) \cdot \prod_{i=N}^{L} p(w_i|w_{i-N+1}, \cdots, w_{i-1}).$$

In practice, N is very small - typically 2 to 4 (a bigram, trigram or 4-gram model, respectively).

1.2.1 Estimating an N-gram model

The N-gram probability $p(w_i|w_{i-N+1}, \cdots, w_{i-1})$ can be estimated by the relative frequency with which w_i follows $\phi(h_i)$ as

$$f(w_i|w_{i-N+1}, \cdots, w_{i-1}) = \frac{\#[w_{i-N+1}, \cdots, w_{i-1}, w_i]}{\#[w_{i-N+1}, \cdots, w_{i-1}]}$$

where $\#[x, y, z]$ denotes the number of times one observes the tuple (x, y, z) in a training corpus. Unfortunately, this estimate suffers from sparse data problem even for moderate values of N. For instance, to train a trigram model for a small task (*e.g.*, Switchboard[1]) with a chosen vocabulary size of 20,000[2], there are 8 trillion free parameters to be estimated but only 3 million samples (or words) available for training. For example, a possible word triple *John likes apples* never occurs in the training data even though the three constituent words occur 56, 190 and 14 times respectively. But probability of *John likes apples* should clearly not be zero. Hundreds of millions of unseen bigrams and trillions of unseen trigrams need to be similarly assigned a non-zero probability.

1.2.2 Smoothing Techniques

Some smoothing techniques, such as back-off, due to Katz (1987) and Kneser & Ney (1995), and deleted interpolation, due to Jelinek & Mercer (1980), have been proposed to assign proper probabilities to the unseen events in the train data.

Deleted Interpolation

An N-gram model may be smoothed by interpolating N-gram, N-1-gram,$\cdots$, unigram relative frequencies. For example a trigram probability $p(w_i|w_{i-2}, w_{i-1})$ may be estimated as

$$\begin{aligned} & p(w_i|w_{i-2}, w_{i-1}) && (1.1) \\ = \ & \lambda_3(w_{i-2}, w_{i-1}) f(w_i|w_{i-2}, w_{i-1}) + \lambda_2(w_{i-2}, w_{i-1}) f(w_i|w_{i-1}) \\ & + \lambda_1(w_{i-2}, w_{i-1}) f(w_i) + \lambda_0 && (1.2) \end{aligned}$$

where the history-dependent non-negative weights λ satisfy

$$\sum_{j=0}^{3} \lambda_j = 1$$

for each history. The training of an interpolation model is a two-phase process: estimating the relative frequencies using a large set of training data and tuning the interpolation weights to maximize the likelihood of some “held-out” data. It is worth

[1]Casual conversations on assigned topics recorded over telephone.

[2]There are about 20,000 different words in the Switchboard corpus, even though the vocabulary of English is much larger than this number.

mentioning that this linear interpolation method can be used to combine other models besides the relative frequency estimate described above.

Back-off

The method of backing-off is very popular in state-of-the-art language models. Several back-off methods have been proposed by Katz (1987), Kneser & Ney (1995) and Chen & Goodman (1999) and others. These back-off schemes are similar in principle and have close performance in speech recognition. Therefore, we only show the principle of the Katz method, which is used for the baseline models in our work.

The basic idea of back-off is to use lower-order Markov models with enough evidence to approximate higher-order ones with insufficient evidence. For example, a trigram model is estimated as

$$p(w_i|w_{i-2}, w_{i-1}) = \begin{cases} f(w_i|w_{i-2}, w_{i-1}) & \text{if } \#[w_{i-2}, w_{i-1}, w_i] \geq T \\ Q_T(w_i|w_{i-2}, w_{i-1}) & \text{if } 1 \leq \#[w_{i-2}, w_{i-1}, w_i] \leq T \\ \alpha(w_{i-2}, w_{i-1}) p(w_i|w_{i-1}) & \text{otherwise} \end{cases} \quad (1.3)$$

using Katz back-off, where Q_T is a Good-Turing[3] (Good, 1953) type function, T is a threshold (typically 5 to 8) and α is the remaining probability mass for the all unseen w_i. $p(w_i|w_{i-1})$ can be recursively backed-off to a unigram probability estimate, etc.

N-gram models are good in the sense that they are easy to implement, computationally efficient to use and accurate much of the time. However, these models cannot take long-distance correlations between words into account, since N is very small. This drawback may lead to inaccurate predictions when the actual predicting information is beyond the local range. In the later chapters, we will discuss how to compensate for this drawback by using long-distance dependence.

[3]Counts r are discounted as $\frac{N_{r+1} \cdot (r+1)}{N_r}$ where N_r is the number of distinct samples occur r times.

1.3 Non-Local Dependencies

In recent years, several attempts to capture long-distance dependencies for language models have been made in either of two directions: incorporation of semantic content and of syntactic structure in natural language. Topic-(or domain-)related information has been studied, *e.g.,* by Bellegarda (1998); Clarkson & Robinson (1997); Chen & Rosenfeld (1998); Florian & Yarowsky (1999); Iyer & Ostendorf (1996); Kneser *et al.* (1997) and Martin, Liermann & Ney (1997). The essential idea in these papers comes from the information retrieval (IR) literature where extensive use is made of weighted term-frequencies (TF) to discern the topic or domain of a document. To see the idea in these papers, consider the words *Financial, contract, revival, futures* and *exchange* in the following sentence:

> *Financial* officials in the former British colony consider the *contract* essential to the *revival* of the Hong Kong *futures exchange.*

Their probability in this specific sentence, given that it addresses the topic *financial news*, should be much greater than their probability in the overall corpus. Most schemes (Clarkson & Robinson, 1997; Iyer & Ostendorf, 1996; Martin, Liermann & Ney, 1997) exploit such differences in word frequencies across topics by constructing separate language models for each individual topic or domain. Such an implementation, however, results in fragmentation of the training text by topic, for which the usual remedy is to interpolate each topic-specific N-gram model with a topic-independent model constructed using all the available data. The work in Florian & Yarowsky (1999) differs from the above methods only in that the topic structure is hierarchically self-constructed in a bottom-up manner. An alternative presented in Chen & Rosenfeld (1998) starts off being similar to the technique that will be presented in this dissertation, but then makes ad hoc changes to an exponential model with the limited objective of fast rescoring. The work on read speech in Kneser *et al.* (1997) is also somewhat similar to our work. However, dynamic shifts in topic are modeled by a unigram-cache rather than a semantic notion of topic. Lafferty & Suhm (1996) describe a topic-dependent model very similar to the topic-dependent model presented here and demonstrate improvement in perplexity over a bigram model with a relatively small 5000-word vocabulary. The approach based on latent semantic analysis (LSA) proposed in Bellegarda (1998) is a refreshing departure from these methods. A matrix of co-occurrences between words and documents is accumulated from the training data by simply keeping track of which words are found in what documents. The dimension of the matrix is reduced by singular value decomposition (SVD).

Relatively few models that utilize syntactic structure for improving over N-gram models have been proposed (c.f. Lafferty *et al.* (1992) and Chelba *et al.* (1997)). Notable among them are Chelba & Jelinek (1998, 1999), who have used a left-to-right parser to extract syntactic heads and used them to enhance trigram models. To illustrate the kinds of dependencies they attempt to capture, consider the following sentence that illustrates the mechanism used by these models.

> Financial **officials** *in the former British colony* **consider** the contract essential to the revival of the Hong Kong futures exchange.

The word *consider* is clearly more at home as part of the bigram *officials consider* as opposed to *colony consider.* This intuitive dependence is incorporated into the abovementioned models by identifying that the prepositional phrase "in the former British colony" modifies the noun phrase "Financial officials" and by suppressing its role in predicting the following word, *consider.* This is achieved, *e.g.*, by a predictor that uses the preceding syntactic head-words instead of preceding words, where a *syntactic head word* is the kernel word in a constituent of a sentence assigned by the syntactic parser. For instance, *officials* is the head work for the noun phrase *Financial officials in the former British colony.* Furthermore, noun phrases such as "Bank managers", "Securities traders" or "Stock brokers" play exactly the same syntactic role as "Financial officials". Therefore, if the bigram *officials consider* happens to be unseen, the *nonterminal label* NP' of the plural noun phrase provides a coarser classification of the history from which to predict the word *consider.* This is achieved via deleted interpolation in Chelba & Jelinek (1998, 1999). Thus the syntactic heads (including both lexical heads and non-terminal labels) bring both syntactic structure and an improved *back-off* scheme to bear on the prediction problem. An alternative to Chelba & Jelinek (1998) is the work of Roark (2001). More profound syntactic knowledge is studied and more candidate parses are considered for each sentence in the latter. A better perplexity reduction and a similar word error rate reduction are achieved compared to those in Chelba & Jelinek (1998).

The methods mentioned above consider either topic or syntactic dependence but not both, due to the difficulty of combining dependencies of different forms in one model and also due to the sparseness of the training data. However, semantic content, *e.g.,* topics, and syntactic structure of a sentence provide complementary information. This fact leads us to propose a new language model that can incorporate several kinds of constraints from different information sources. Maximum entropy (ME) techniques are efficient means of combining information from different sources in a single, unified model. Various kinds of statistical constraints can be treated as features in a maximum entropy model.

1.4 Maximum Entropy Language Modeling

The maximum entropy method was first successfully used in language modeling by Rosenfeld (1994). Language models integrating triggers[4] and N-gram features were presented in this dissertation. A substantial perplexity reduction was reported on the Wall Street Journal (WSJ) task. Several endeavors have been made in building maximum entropy language models with non-local constraints in recent years. In Stolcke, *et al* (1996) and Berger & Printz (1998), grammatical features from the link grammar (Sleator & Temperley, 1993) were incorporated with N-grams in the maximum entropy framework. In Chen & Rosenfeld (1998); Kneser *et al.* (1997) and Khudanpur & Wu (1999), topic specific language models were built using the maximum entropy method. A further attempt to incorporate several sources of information in one language model were presented in Wu & Khudanpur (1999) and Khudanpur & Wu (2000). The smoothing techniques for maximum entropy language models have been studied by Ney, Martin & Wessel (1997).

[4]If a word is significantly correlated with another word, they are considered as a *trigger pair*.

We illustrate the maximum entropy language modeling method by considering a trigram model. A trigram model should meet the following unigram, bigram and trigram constraints:

$$\begin{aligned}
p(w_i|w_{i-1}, w_{i-2}) \cdot p(w_{i-1}, w_{i-2}) &= \frac{\#[w_{i-1}, w_{i-2}, w_i]}{\#[\textit{training data}]}, && (1.4)\\
\sum_{w_{i-2}} p(w_i|w_{i-1}, w_{i-2}) \cdot p(w_{i-1}, w_{i-2}) &= \frac{\#[w_{i-1}, w_i]}{\#[\textit{training data}]}, && (1.5)\\
\sum_{w_{i-2}, w_{i-1}} p(w_i|w_{i-1}, w_{i-2}) \cdot p(w_{i-2}, w_{i-1}) &= \frac{\#[w_i]}{\#[\textit{training data}]}. && (1.6)
\end{aligned}$$

These constraints define a linear family of probability mass functions (*pmfs*), and we choose among them the model that has the maximum entropy, corresponding to the least additional assumptions on the model. It is well-known that such an ME model has an exponential form with a parameter λ corresponding to each linear constraint placed on the model:

$$p_{\underline{\lambda}}(w_i|w_{i-1}, w_{i-2}) = \frac{e^{\lambda_{w_i} \cdot g(w_i) + \lambda_{w_{i-1}, w_i} \cdot g(w_{i-1}, w_i) + \lambda_{w_{i-2}, w_{i-1}, w_i} \cdot g(w_{i-2}, w_{i-1}, w_i)}}{z(\underline{\lambda}, w_{i-1}, w_{i-2})}. \qquad (1.7)$$

where z is a normalization factor. The numerator terms from left to right correspond to unigram, bigram and trigram constraints in that order where the g's are feature indicator functions and λ's are their corresponding weights.

There are two important questions we have not answered yet. First, why the maximum entropy model satisfying constraints (1.4)-(1.6) has the above exponential form (1.7). Second, how the feature functions g are defined and how the model parameter λ's are obtained. Readers may keep these two questions in mind, and we will answer them later.

1.4.1 Advantages of the Maximum Entropy Method

There are many advantages of maximum entropy methods. For example, the data sparseness problems in language modeling can be avoided by setting only reliable marginal constraints. However, the most distinct advantage of a maximum entropy framework is its ability to combine different kinds of statistical dependencies in one unified framework.

For instance, the trigram model (1.7) can be easily augmented by adding topic-dependent unigram constraints

$$\sum_{w_{i-1},w_{i-2}} p(w_i|w_{i-1},w_{i-2},t_i)\cdot p(w_{i-1},w_{i-2}) \quad = \quad \frac{\#[t_i,w_i]}{\#[t_i]} \tag{1.8}$$

to achieve the topic-dependent trigram model

$$p_{\underline{\lambda}}(w_i|w_{i-1},w_{i-2},t_i) = \frac{e^{\lambda_{w_i}\cdot g(w_i)+\lambda_{w_{i-1},w_i}\cdot g(w_{i-1},w_i)+\lambda_{w_{i-2},w_{i-1},w_i}\cdot g(w_{i-2},w_{i-1},w_i)+\lambda_{t_i,w_i}\cdot g(t_i,w_i)}}{Z(\underline{\lambda},w_{i-1},w_{i-2},t_i)}, \tag{1.9}$$

where t_i is the "topic" of the utterance.

The above topic-dependent model (1.9) differs from the trigram model (1.7) only in the fourth numerator term corresponding to a topic-dependent salient constraint[5].

Also note that only topic-dependent *marginal* constraints and topic-independent N-gram constraints are set in the above model, and that this avoids the data sparseness problems in estimating a separate conditional probability mass function *pmf* for each $[w_{i-1},w_{i-2},t_i]$ since both tuples $[w_{i-2},w_{i-1},w_i]$ and $[w_i,t_i]$ may be seen enough times even though their conjunctions $[w_{i-2},w_{i-1},w_i,t_i]$ may not.

1.4.2 Disadvantages of the Maximum Entropy Method

A fundamental difficulty in implementing such a maximum entropy model is the heavy computational cost of the training procedure. In state-of-the-art training algorithms (Della Pietra, Della Pietra & Lafferty, 1997; Jelinek, 1997), the computational complexity for each iteration in the training is at least $O(|\hat{X}|\cdot\bar{C})$, where $|\hat{X}|$ is the number of observed "histories" as will be defined later, and $\bar{C}$ is the average number of words with any non-unigram constraint[6] for each history. Although this algorithm is much better than a straightforward implementation with the complexity of

[5]Of course, the value of parameters λw_i, $\lambda w_{i-1}, w_i$ and $\lambda w_{i-2}, w_{i-1}, w_i$ in the topic-dependent model differs from those in the trigram model even though feature functions $g(w_i)$, $g(w_{i-1},w_i)$ and $g(w_{i-2},w_{i-1},w_i)$ are the same.

[6]Unigram constraints are also called *marginal constraints* in the literature

$O(|\hat{X}| \cdot |V|)$, where V is the vocabulary, it is still not practical for complex language models or for large tasks. For instance, even in a small task such as Switchboard (Godfrey *et al.*, 1992) with only 3 million words in the training data, $|\hat{X}| \sim 10^6$ and $\bar{C} \sim 10^3$ for the model with both topic and syntactic dependencies. The problem is more severe for larger corpora, *e.g.,* Broadcast News (BN), in which both the number of histories and the average number of words with conditional features go up by one to two orders of magnitude. We overcome this computational challenge by means of an efficient hierarchical training method that will be introduced in later chapters.

1.5 Measuring Language Model Performance

A direct measure of the quality of a language model in speech recognition is the *word error rate* (WER) of the output of the speech recognizer using this model. We align the reference transcription (truth) with the output of the speech recognizer (hypothesis) for each utterance and count the number of mismatches (errors). The word error rate is the ratio of the number of errors to the total number of words in the reference. The following table provides an example of aligning the reference answer

Some of it runs off right away in to the streams and rivers

with a candidate hypothesis

Some of the runs off right away in terms of the streams rivers.

REF:	Some	of	IT	runs	off	right	away	in	*****	TO	the	streams	AND	rivers
HYP:	Some	of	THE	runs	off	right	away	in	TERMS	OF	the	streams	***	rivers
Eval:	C	C	S	C	C	C	C	C	I	S	C	C	D	C

In the third row of the table, "C" means correct while "S", "I" and "D" correspond to three kinds of errors in continuous speech recognition - substitution (S), insertion (I) and deletion (D), respectively. The word error rate is defined as

$$WER = \frac{S + I + D}{\#\text{ of tokens in the reference transcripts}} = \frac{S + I + D}{C + S + D}. \tag{1.10}$$

In the above example, $WER = 4/13 \approx 31\%$.

The performance of a speech recognizer depends on the quality of both the acoustic model and the language model. To compare the performance of two different language models, the acoustic model must be fixed and these two language models applied separately in the recognizer. In our experiments, we generate N-best[7] candidate sentences for each utterance using a speech recognizer with a baseline (trigram) model, and then re-score these N-best hypotheses using the new language model.

When a speech recognizer is not available, an alternative measurement *perplexity* (Jelinek, 1990) is used instead. The perplexity measures the *branching factor* of a language model, which shows the average number of words that will follow a given history.

Let $p(x)$ be the real probability distribution of x, and $p_M(x)$ be the probability estimate of x based on a statistical language model M estimated from the training data. The entropy of $p(x)$ defined as

$$H(P) = -\sum_x p(x) \cdot \log p(x)$$

can be interpreted as the difficulty of the task of predicting x. The *cross-entropy* of p_M and p is defined as

$$H_p(p_M) = -\sum_x p(x) \cdot \log p_M(x).$$

It can easily be proved that

$$H(p) \leq H_p(p_M).$$

The cross-entropy measures the similarity of these two distributions p and p_M - the smaller p_M is, the better the model M approximates p. The *perplexity* (PPL) of the test data with respect to the model M is then defined as

$$PPL_M = 2^{H_p(p_M)} \tag{1.11}$$

and is used as a quantitative measurement for both the quality of the language model and the complexity of the text source. For the same text source, the smaller the

[7]where $N = 100$ in this dissertation

perplexity, the better the model; for the same model, the test data that has larger perplexity is more difficult to process. Lower perplexity on a given test set usually, but not necessarily, implies lower WER in speech recognition.

The remainder of the dissertation is organized as follows. Chapter 2 introduces the maximum entropy principle and the maximum entropy language modeling techniques. Chapter 3 focuses on the efficient training methods for maximum entropy models. Chapters 4 and 5 discuss how to incorporate semantic, in particular topic, dependencies and syntactic dependencies, respectively, with N-grams in a language model. Chapter 6 illustrates how to combine all N-gram, syntactic and semantic constraints together. Experimental results with detailed analysis will be presented in each of these chapters. Chapter 7 addresses efficient ways of computing probabilities at recognition time given a maximum entropy model. Even though we focus on application of maximum entropy language modeling techniques to automatic speech recognition (ASR) in this research, maximum entropy methods can be easily used in many other applications in natural language processing. We conclude this dissertation in Chapter 8.

Chapter 2

Maximum Entropy Modeling

In this chapter, we introduce the maximum entropy principle and its application to language modeling. We will show a concrete example of building a trigram model using the maximum entropy method. We also address the computational issues and challenges in estimating model parameters.

2.1 The Maximum Entropy Principle

When we try to model the behavior of a random variable or a stochastic process from data, we should choose through intuition a model that satisfies all observed properties of this variable or process while making no "unwarranted" assumptions. This results in a *maximum entropy model* in the sense of information theory. We introduce the maximum entropy principle by providing simple examples of dice. Readers will see that the maximum entropy results meet our intuitions.

2.1.1 Examples: Dice

Let p_i be the probability of the event that the facet with i dots faces up, where $i = 1, \cdots, 6$. We want to estimate p_i with no empirical knowledge. The intuitively appealing model is the uniform model, in which $p_i = \frac{1}{6}$ for $i = 1, 2, \cdots, 6$. Actually, it is the maximum entropy model! Now let us find the maximum entropy distribution mathematically and check whether it matches our intuitive uniform model.

Problem 1:

Find the probability distribution $P = (p_1, p_2, \cdots, p_6)$, which maximizes the entropy

$$H(P) = -\sum_{i=1}^{6} p_i \log(p_i).$$

Solution:

An implicit constraint for a probability distribution is $\sum_{i=1}^{6} p_i = 1$.

We use the method of *undetermined Lagrangian multipliers.*

Let

$$\mathcal{L}(P, \alpha) = \alpha(\sum_{i=1}^{6} p_i - 1) + (-\sum_{i=1}^{6} p_i \log(p_i)).$$

Setting the partial derivatives with respect to p_j to zero, we get

$$\frac{\partial L}{\partial p_j} = -1 - \log p_j + \alpha = 0.$$

Solving the above equations we obtain the following solution:

$$p_i = \frac{1}{6}, \text{ for } i = 1, \cdots, 6.$$

∎

Next, we assume that the die is loaded, and we observe that the facet with five dots has a $\frac{1}{3}$ chance of facing up. We want to find the values of p_i that satisfy this empirical constraint. Our intuition tells us to choose $\frac{1}{3}$ for p_5 and an equal value for the rest of the p_i. Once again, we use the maximum entropy method to solve this problem.

Problem 2:

Find the probability distribution p_i, for $i = 1, 2, \cdots, 6$, which maximizes $H(P)$ subject to the constraint

$$P[f] = \sum_{i=1}^{6} p_i f(i) = \frac{1}{3},$$

where f, defined as

$$f(i) = \begin{cases} 1 & \text{if } i = 5, \\ 0 & \text{otherwise,} \end{cases}$$

is a partition function describing the specific constraint for the facet with five dots.

Solution:

Creating the Lagrangian

$$\mathcal{L}(P, \alpha) = \alpha_1(\sum_{i=1}^{6} p_i - 1) + \alpha_2(\sum_{i=1}^{6} f(i) \cdot p_i - \frac{1}{3}) + (-\sum_{i=1}^{6} p_i \log(p_i)),$$

and setting 0 to its partial derivatives $\frac{\partial L}{\partial p_j}$, we obtain the following equations:

$$\begin{cases} \alpha_1 + \alpha_2 - \log p_5 - 1 & = 0, \\ \alpha_1 - \log p_i - 1 & = 0 \text{ for } i = 1, 2, 3, 4, 6. \end{cases}$$

Solving the above equations, we obtain the solution

$$p_i = \frac{2}{15}, \text{ for } i = 1, 2, 3, 4, 6, \text{ and } p_5 = \frac{1}{3}.$$

$\blacksquare$

From the two examples above, we can see that maximum entropy distributions match our intuitions. Shore & Johnson (1980), Jaynes (1982) and Csiszár (1991) showed why the maximum entropy distribution is the optimal among all probability distributions obtained from (incomplete) data. The examples above also show a way of solving ME problems using Lagrangian multipliers. Unfortunately, the method of Lagrangian multipliers is not practical for most applications since the number of constraints is huge, resulting in an enormously large number of nonlinear equations to be solved. The analytical solution is either non-existent or hard to find. However, maximum entropy models can be estimated by numerical methods, which will be introduced in the following sections.

2.1.2 The General Problem

Problem 2 in the previous section is a concrete example of inferring a probability distribution from some observations. A typical problem of statistical modeling is to estimate a probability measure $p(s)$ on $\mathcal{S}$ from observations $s_1, s_2, \cdots, s_L$, where L is the number of samples. In the maximum entropy framework, a distribution $p*$ is chosen to maximize the entropy $H(p) = -\sum_{s\in\mathcal{S}} p(s)\log p(s)$ from a *linear family* of probability distributions

$$\mathcal{P} = \{p : \sum_{s\in\mathcal{S}} p(s) g_k(s) = a_k, 1 \le k \le K\} \tag{2.1}$$

for a given set of real-valued functions $G = \{g_1, g_2, \cdots, g_K\}$ defined on $\mathcal{S}$. It is easy to check that $\mathcal{P}$ is closed and convex.

The above problem is equivalent to finding a distribution p^* to minimize the *I-divergence*

$$D(p||q) = \sum_{s\in\mathcal{S}} p(s) \log \frac{p(s)}{q(s)}$$

between p and the uniform distribution $q = q(x) = \frac{1}{|\mathcal{X}|}, \forall x$,*i.e.*,

$$p^* = \arg\min_{p\in\mathcal{P}} D(p||q).$$

We define an *exponential family*

$$\mathcal{R} = \left\{ p : p(s) = \frac{1}{z} q(s) \exp\left[\sum_{k=1}^{K} \lambda_k g_k(s)\right], \text{ for some } \lambda_1, \lambda_2, \cdots, \lambda_K \right\} \tag{2.2}$$

for the given functions $g_1, g_2, \cdots, g_K$ and the reference distribution q, where

$$z = \sum_{s\in\mathcal{S}} q(s) e^{\sum_{k=1}^{K} \lambda_k g_k(s)}$$

is a normalization constant.

2.1.3 The Maximum Entropy Solution

Csiszár (1975) showed that intersection $\mathcal{P} \cap \bar{\mathcal{R}} = \{p^*\}$ where $\bar{\mathcal{R}}$ is the closure of $\mathcal{R}$. This means that the maximum entropy distribution p^* of the linear family $\mathcal{P}$ is unique and has an exponential form, and conversely, if a *pmf* p^* has the exponential form and satisfies the linear constraints $\{\sum_s p(s)g_k(s) = a_k, 1 \leq k \leq K\}$, then it has maximum entropy.

Therefore, instead of finding the maximum entropy distribution from the linear family $\mathcal{P}$, we look for the model satisfying these linear constraints from the exponential family $\mathcal{R}$, because the latter can be solved by the generalized iterative scaling method (Darroch & Ratcliff, 1972; Csiszár, 1989).

Generalized Iterative Scaling

Darroch & Ratcliff (1972) proposed the generalized iterative scaling (GIS) algorithm for seeking the probability distribution with an exponential form,

$$p(s) = c \cdot q(s) \prod_{k=1}^{K} \alpha_k^{g_k(s)}, \tag{2.3}$$

which satisfies linear constraints of the form

$$\sum_{s \in S} p(s) \cdot g_k(s) = a_k, \text{ for } k = 1, 2, \ldots, K, \tag{2.4}$$

where the g_k's are real-valued functions. Darroch & Ratcliff (1972) proved that starting with $p_0 = q$, the estimate $p_{n+1}(s)$ of p in the $(n+1)^{th}$ iteration computed by

$$p_{n+1}(s) = p_n(s) \prod_{k=1}^{K} (\frac{a_k}{a_{i,n}})^{g_k(s)}, \tag{2.5}$$

where $a_{i,n}$ is the expectation of the i^{th} feature function in the n^{th} iteration

$$a_{i,n} = \sum_s p_n(s) g_k(s), \tag{2.6}$$

converges, i.e., $p_n(s) \to p^*(s)$

$$p^*(s) = \frac{1}{z} exp\{\sum_{k=1}^{K} \lambda_k^* g_k(s)\}.$$

It is worth emphasizing that there exists a unique solution to (2.3) and (2.4).

Starting from some arbitrary q^1, one converges to a unique solution for maximum entropy models by the GIS algorithm. However, GIS has some practical problems. First, the convergence rate is quite slow for large models. Furthermore, if the linear constraints are not consistent[2], intermediate values of model parameters may even overflow/underflow the range of double precision representation before they would converge. An improved iterative scaling algorithm has been proposed by Della Pietra, Della Pietra & Lafferty (1997) to overcome these computational issues.

Improved Iterative Scaling

We need additional notation

$$g_{\#}(s) = \sum_{k=1}^{K} g_k(s)$$

to describe the IIS algorithm. If $g_k(s)$ is binary, $g_{\#}(s)$ is the total number of features that are active on the s.

Compared to Equation (2.5), a more general iterative solution for p^* has the following form:

$$p_{n+1} = p_n \cdot \mathbf{u} \tag{2.7}$$

where $\mathbf{u} = u_1,\ u_2,\ ...,\ u_K$ is a K dimensional vector of positive constants. Della Pietra, Della Pietra & Lafferty (1997) showed that if $\mathbf{u}$ is the unique solution [3] of

$$\sum_{s \in \mathcal{S}} p_n(s) \cdot {u_k}^{g_{\#}(s)} \cdot g_k(s) = a_k$$

for all k, then the iterative equation (2.7) provides a solution for $p_n \to p^*$. In practice, the IIS algorithm converges faster than GIS [4].

[1] Usually initialized as a uniform distribution.

[2] Very likely in real applications.

[3] The computational cost of solving these equations numerically using Newton's method may be ignored when compared to the total training time.

[4] If $g_{\#}(s)$ is a constant for all s, IIS is the same as GIS.

2.2 Maximum Entropy Language Modeling

In this section, we apply the maximum entropy principle to language modeling. In order to formalize inference for maximum entropy language models that satisfy constraints we set, we need to define some concise notation. We use x and y, respectively, to represent the history and the word following that history in language modeling. For instance, $x = w_{i-2}, w_{i-1}$ and $y = w_i$ in a trigram model $p(w_i|w_{i-2}, w_{i-1})$. We use $\mathcal{X}$ and $\mathcal{Y}$ to represent the set of histories and the set of (future) words, respectively. A typical problem in language modeling [5] is to estimate a conditional probability measure $p(y|x)$ on a probability space $\mathcal{X} \times \mathcal{Y}$. Here, $\mathcal{Y} = V$ where V is the vocabulary. In the trigram model $p(w_i|w_{i-2}, w_{i-1})$, $\mathcal{X} = \mathcal{Y} \times \mathcal{Y}$.

In the maximum entropy framework, a set of K real-valued feature functions $\{g_1, \cdots, g_K\}$ need to be defined on $\mathcal{X} \times \mathcal{Y}$. However, for convenience, only binary feature functions

$$G = \{g_k : (\mathcal{X} \times \mathcal{Y}) \to \{0, 1\}, k = 1, \dots, K\}$$

are considered in language modeling. [6]

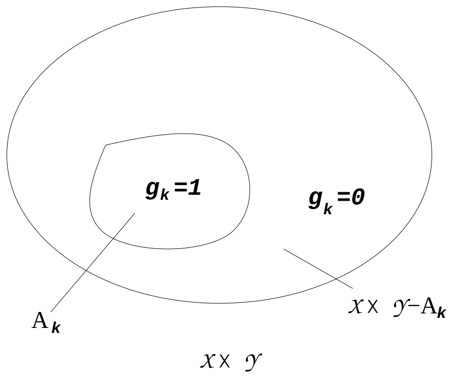
g_k=1
g_k=0
A$_k$
$\mathcal{X}\times\mathcal{Y}$−A$_k$
$\mathcal{X}\times\mathcal{Y}$

Figure 2.1: A binary feature g_k partitions $\mathcal{X} \times \mathcal{Y}$.

Each binary feature g_k for $k = 1, \cdots, K$, partitions the domain $\mathcal{X} \times \mathcal{Y}$ into two sets A_k and $(\mathcal{X} \times \mathcal{Y}) \setminus A_k$: those $\langle x, y\rangle \in \mathcal{X} \times \mathcal{Y}$ for which $g_k(x, y)$ is active and those

[5] And in many other empirical natural language processing applications.
[6] We will address how to choose these features for language models in subsequent chapters.

for which it is not (Figure 2.1), or more formally,

$$g_k(x,y) \doteq g_k(x,y;A_k) = \begin{cases} 1 & \langle x,y \rangle \in A_k, \\ 0 & \text{otherwise}, \end{cases} \tag{2.8}$$

where x and y are variables and A_k can be regarded as a coefficient.

For instance, for words $a, b, c \in V$, unigram, bigram and trigram feature functions are defined respectively as

$$g_1(x,y;c) = \begin{cases} 1 & y = c, \\ 0 & \text{otherwise}, \end{cases} \tag{2.9}$$

$$g_2(x,y;b,c) = \begin{cases} 1 & x = \cdots b \text{ and } y = c, \\ 0 & \text{otherwise}, \end{cases}$$

and

$$g_3(x,y;a,b,c) = \begin{cases} 1 & x = \cdots a, b \text{ and } y = c, \\ 0 & \text{otherwise}. \end{cases}$$

A vector $\mathbf{a} = (a_1, ..., a_K)$ is chosen to be the *target expectation* [7] for these features based on the observed samples $\langle x_1, y_1 \rangle$,..., $\langle x_L, y_L \rangle$. Typically, $a_k = \frac{\# < x_i, y_i > \in A_k}{L}$.

The set of features G and their target expectations $\mathbf{a}$ define a linear family of probability distributions

$$\mathcal{P} = \{p : p[g_k] = a_k, \ \forall \ k = 1, ..., K\}, \tag{2.10}$$

where $p[g_k]$ is the expectation of $g_k(x,y;A_k)$ with respect to the distribution $p(x,y)$, as

$$p[g_k] = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \cdot g_k(x,y;A_k). \tag{2.11}$$

Note that (2.10) has the same form as (2.1). The only difference is that g_k in (2.1) is a real-valued function. It becomes a binary one in (2.10).

[7] *E.g.,* empirical expectation.

Given such a class $\mathcal{P}$ of probability distributions, we want to find the distribution $p^*(x, y)$ in $\mathcal{P}$ that maximizes the entropy $H(p)$. Since we have shown in Section 2.1.3 that the maximum entropy solution has the exponential form, we consider the alternative class $\mathcal{R}$ of exponential models $m(x, y)$ defined over G as

$$\mathcal{R} = \{m : m(x, y) = \frac{1}{z} e^{\sum_{k=1}^{K} \lambda_k g_k(x,y;A_k)}\}, \tag{2.12}$$

or in the simpler notation of Ristad (1997)

$$\mathcal{R} = \left\{m : m(x, y) = \frac{r(x, y)}{z}\right\}, \tag{2.13}$$

where

$$r(x, y) = \prod_{k=1}^{K} \alpha_k^{g_k(x,y;A_k)}, \qquad z = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} r(x, y) \tag{2.14}$$

and the α's are nonnegative real numbers. Here $\lambda_k = \ln(\alpha_k)$.

In language modeling, we are more interested in the conditional probability $m(y|x)$ than in $m(x, y)$ where

$$m(y|x) = \frac{1}{z(x)} \prod_{k=1}^{K} e^{\lambda_k \cdot g_k(x,y;A_k)}, \qquad z(x) = \sum_{y \in \mathcal{Y}} \prod_{k=1}^{K} e^{\lambda_k \cdot g_k(x,y;A_k)}. \tag{2.15}$$

Given the set of feature functions G and their target expectations $\mathbf{a}$, parameters α (or λ) are computed using either GIS or IIS. The art of maximum entropy language modeling is in choosing the most "useful" feature functions and setting proper target expectations for them.

2.3 An Example: Building a Maximum Entropy Trigram Model

In this section, we show how to build an ME language model by a concrete example of a trigram model. For a triple (w_{i-2}, w_{i-1}, w_i), the model estimating $p(w_i|w_{i-2}, w_{i-1})$ should satisfy the following unigram, bigram and trigram constraints that are copied from (1.4)-(1.6):

$$\sum_{w_{i-2},w_{i-1}} p(w_i|w_{i-1},w_{i-2}) \cdot p(w_{i-2},w_{i-1}) = \frac{\#[w_i]}{\#[training\ data]}, \tag{2.16}$$

$$\sum_{w_{i-2}} p(w_i|w_{i-2},w_{i-i}) \cdot p(w_{i-2},w_{i-1}) = \frac{\#[w_{i-1},w_i]}{\#[training\ data]}, \tag{2.17}$$

$$p(w_i|w_{i-2},w_{i-1}) \cdot p(w_{i-2},w_{i-1}) = \frac{\#[w_{i-2},w_{i-1},w_i]}{\#[training\ data]}. \tag{2.18}$$

We define a binary function

$$g_1(w_i) \doteq g(x,y;w_i) = \begin{cases} 1 & y = w_i, \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

corresponding to the first (unigram) constraint. The expectation of this feature under a model p ,

$$p[g_1] = \sum_{\langle x,y\rangle : y=w_i} g(x,y;w_i)p(y|x)p(x) = \sum_{w_{i-2},w_{i-1}} p(w_i|w_{i-2},w_{i-1})p(w_{i-2},w_{i-1})$$

is identical to the left hand side of (2.16). Finding a model satisfying the unigram constraint (2.16) is equivalent to finding a model p under which the expectation of the feature function $g_1(w_i)$ satisfies

$$\sum_{\langle x,y\rangle : y=w_i} g(x,y;w_i)p(y|x)p(x) = \frac{\#[w_i]}{\#[training\ data]}. \tag{2.20}$$

Similarly we can define the bigram feature function

$$g_2(w_{i-1},w_i) \doteq g(x,y;w_{i-1},w_i) = \begin{cases} 1 & y = w_i \text{ and } x = *\ w_{i-1}, \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

corresponding to the bigram constraint (2.17), where $*$ means any word string, and the trigram feature function

$$g_3(w_{i-2},w_{i-1},w_i) \doteq g(x,y;w_{i-2},w_{i-1},w_i) = \begin{cases} 1 & y = w_i \text{ and } x = *\ w_{i-2},w_{i-1}, \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

corresponding to the trigram constraint (2.18). The model satisfying the N-gram constraints (2.16)-(2.18) is the model under which the expectations of the feature functions (2.19), (2.21) and (2.22) equal their empirical expectations. According to the conclusion in Section 2.2, the model has the exponential form of (2.15).

Equation (2.15) involves K factors for each conditional probability. However, given any 3 tuple w_{i-2}, w_{i-1}, w_i, the trigram model has at most three features $g_1(w_i)$, $g_2(w_{i-1}, w_i)$ and $g_3(w_{i-2}, w_{i-1}, w_i)$ active simultaneously for a given N tuple (w_{i-2}, w_{i-1}, w_i); therefore,

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{\alpha_{w_i}^{g_1(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g_2(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g_3(w_{i-2},w_{i-1},w_i)}}{z(w_{i-2}, w_{i-1})}, \tag{2.23}$$

where

$$z(w_{i-2}, w_{i-1}) = \sum_{w_i \in V} \alpha_{w_i}^{g_1(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g_2(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g_3(w_{i-2},w_{i-1},w_i)}.$$

Note that we use coefficients w_i, w_{i-1}, w_i and w_{i-2}, w_{i-1}, w_i as the subscripts for α's since the value of α's actually depends on these coefficients.

Substituting all $\lambda = \ln(\alpha)$ for $i = 1, 2$ and 3 in (2.23), we obtain (1.7). Now we have answered the remaining question in Section 1.4, namely, "why ME models have the exponential form, how to define feature functions and how to obtain feature parameters."

2.4 Features of Maximum Entropy Language Models

Features of maximum entropy models can come from any kind of information sources besides the N-grams shown in the previous section. In this section, we study several kinds of feature functions used in language models in this dissertation.

According to definition (2.8), the feature function $g_k(x, y; A_k)$ is identified by the subset A_k, and thus the subscript k may be omitted without causing any confusion. Even though A_k could be any subset of $\mathcal{X} \times \mathcal{Y}$ in principle, it may be assumed to contain a subset of $\mathcal{X}$ and only one element in $\mathcal{Y}$ in language modeling[8] for simplicity.

A meaningful constraint in a maximum entropy model should reflect the dependencies between histories and future words obtained from some sources of information. In order to organize the feature set properly, we categorize feature functions by the information source from which these features come and according to their order.

2.4.1 Feature Categorization By Information Source

Collocation Dependencies

We use the collocation dependencies, i.e., the dependencies of the current word and several preceding words, in all our ME models because they are reliable and have been shown to be effective in all kinds of language models. In this dissertation, we use w_i to represent the current word and $w_{i-N+1}^{i-1} \doteq w_{i-N+1}, \cdots, w_{i-1}$ for N-1 preceding words. The collocation N-gram features have the form of

$$\begin{aligned} g(w_{i-N+1}, \cdots, w_{i-1}, w_i) &\doteq g(x, y; w_{i-N+1}, \cdots, w_{i-1}, w_i) \\ &= \begin{cases} 1 & x \text{ end with the suffix } w_{i-N+1}^{w_{i-1}}, \text{ and } y = w_i, \\ 0 & otherwise. \end{cases} \end{aligned}$$

Topic Dependencies

The topic of the discussion t_i strongly influences the use of the current word w_i. An ME language model may use this topic dependency. A topic feature has the form of

$$g(t_i, w_i) \doteq g(x, y; t_i, w_i) = \begin{cases} 1 & t_i \text{ is the topic of } x \text{ and } y = w_i, \\ 0 & otherwise. \end{cases}$$

The short-hand notation $g(t_i, w_i)$ is used for topic unigram features in this dissertation.

Syntactic Dependencies

We will show in Chapter 5 the benefit of syntactic dependencies in improving the

[8]Because a language model is used to predict the probability of a word given the context

language model performance. Two kinds of syntactic dependencies are explored in this dissertation. The first kind involves in syntactic head words hw_{i-2}, hw_{i-1}[9] and the current word w_i. The feature function can depend on only one head word hw_{i-1} or on both head words, with the forms

$$\begin{aligned} g(hw_{i-1}, w_i) &\doteq g(x, y; hw_{i-1}, w_i) \\ &= \begin{cases} 1 & hw_{i-1} \text{ is the preceding head word in } x \text{ and} \\ & y = w_i, \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} g(hw_{i-2}, hw_{i-1}, w_i) &\doteq g(x, y; hw_{i-2}, hw_{i-1}, w_i) \\ &= \begin{cases} 1 & hw_{i-2}, hw_{i-1} \text{ are the two preceding head words in } x \text{ and} \\ & y = w_i, \\ 0 & \text{otherwise,} \end{cases} \end{aligned}$$

respectively.

Another kind of syntactic dependency (already mentioned in Chapter 1) is that between syntactic non-terminal (NT) labels nt_{i-2}, nt_{i-1} of the preceding heads and the current word w_i, with the corresponding feature functions

$$\begin{aligned} g(nt_{i-1}, w_i) &\doteq g(x, y; nt_{i-1}, w_i) \\ &= \begin{cases} 1 & nt_{i-1} \text{ is the NT of preceding head in } x \text{ and } y = w_i, \\ 0 & otherwise \end{cases} \end{aligned}$$

and

$$\begin{aligned} g(nt_{i-2}, nt_{i-1}, w_i) &\doteq g(x, y; nt_{i-2}, nt_{i-1}, w_i) \\ &= \begin{cases} 1 & nt_{i-2}, nt_{i-1} \text{ are the two preceding NT labels in } x \text{ and} \\ & y = w_i, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

[9]Briefly introduced in Chapter 1 and to be discussed in detail in Chapter 5

Word-Class Based Dependencies

When the vocabulary size is large and the training data is relatively sparse, the trigram dependence may not be reliable. One widely used approach to avoid this problem is to cluster words into word classes, and to accumulate statistics according to word classes instead of individual words. The most common word classification is based upon the part-of-speech (POS) tag of words. In this dissertation, we also explore the dependencies between the part-of-speech tags of the two preceding words, pos_{i-2}, pos_{i-1}, and the current word w_i, and we create feature functions

$$\begin{aligned} g(pos_{i-1}, w_i) &\doteq g(x, y; pos_{i-1}, w_i) \\ &= \begin{cases} 1 & pos_{i-1} \text{ is the preceding POS tag in } x \text{ and } y = w_i, \\ 0 & otherwise \end{cases} \end{aligned}$$

and

$$\begin{aligned} g(pos_{i-2}, pos_{i-1}, w_i) &\doteq g(x, y; pos_{i-2}, pos_{i-1}, w_i) \\ &= \begin{cases} 1 & pos_{i-2}, pos_{i-1} \text{ are the two preceding POS tags in } x \text{ and} \\ & y = w_i, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

2.4.2 Feature Categorization By Order

We call the unigram feature $g(w_i)$ an *order 1* feature because its value depends on one variable y. Similarly, we call the bigram feature $g(w_{i-1}, w_i)$ an *order 2* feature and the trigram feature $g(w_{i-2}, w_{i-1}, w_i)$ an *order 3* feature because they are determined by 2 and 3 variables, respectively. In general, we call the N-gram feature an *order N* feature.

Since the value of the unigram feature function $g(w_i)$ depends only on whether $y = w_i$ and is independent of x, the unigram feature is also called a *marginal feature.* All other features do depend on x and are thus called *conditional features.*

Even though features of ME models can come from other information sources than collocation dependence, they still look like N-gram features. For example, the head word features $g(hw_{i-2}, hw_{i-1}, w_i)$ look like a trigram model and thus can be regarded as head word trigram features. We can assign an order to all features as we have done for N-gram features.

If a history x is an m-vector $(x_1, x_2, \cdots, x_m)$ where x_1, x_2, $\cdots, x_m$ are symbols, a feature function is called an *order N* feature function if its value depends on $N-1$ symbol in $(x_1, x_2, \cdots, x_m)$ and on y. For example, $g(nt_{i-2}, nt_{i-1}, w_i)$ is an order 3 feature function.

In this dissertation, we only consider features of order 3 or lower because the estimate of higher order feature parameters is not very reliable due to the data sparseness problem.

2.4.3 Feature Patterns

Each kind of feature described in Section 2.4.1 may be called a *feature pattern* (or a *feature type*) in later chapters. In summary, we have defined some N-gram feature patterns denoted as $g(w_{i-N+1}, \cdots, w_i)$, one feature pattern for topic constraints, two patterns for head word constraints and two for non-terminal constraints in Section 2.4.1. Two features with the same order but from different information sources will not be confusing since they belong to different patterns, even though they are in a similar form. For instance, both $g(t_i, w_i)$ and $g(w_{i-1}, w_i)$ have two coefficients, but they are clearly corresponding to the topic feature and the regular bigram feature, respectively.

In the rest of this section, we will give a formal definition for a feature pattern, which is based on the concept of *subsequence of a history.*

Definition (Subsequence and Supersequence):
$s' = s_{k_1}, s_{k_2}, \cdots, s_{k_l}$ is a *subsequence* of a string (or a vector) $s = s_1, s_2, \cdots, s_d$ if $1 \leq k_1 < k_2 < \cdots < k_l \leq d$. s is then a *supersequence* of s'.

A history x can be treated as a vector $x_1, x_2, \cdots, x_d$ where x_i is a symbol at position i since the number of components of x is fixed. A subsequence $x' = x_{k_1}, x_{k_2}, \cdots, x_{k_l}$ of x can thus be treated as a class of histories with the same symbols in positions $k_1, k_2, \cdots, k_l$. For example, if $x = x_1, x_2, x_3, x_4 = hw_{i-2}, hw_{i-1}, nt_{i-2}, nt_{i-1}$

denotes the history of two preceding head words and two preceding non-terminal labels, $x' = x_3, x_4 = nt_{i-2}, nt_{i-1}$ denotes all histories with the same non-terminal labels nt_{i-2}, nt_{i-1} no matter what the head words are.

Definition (Feature Pattern):

Let $x = x_1, x_2, \cdots, x_d$ be a history and $x_{k_1}, x_{k_2}, \cdots, x_{k_l}$ be a subsequence of history x where $k_l \leq d$. All features that depend on symbols at position $k_1, k_2, \cdots, k_l$ are regarded as belonging to one feature pattern denoted as $g(x_{k_1}, x_{k_2}, \cdots, x_{k_l}, y)$.

The order and the information source of a feature are clearly indicated by the feature pattern.

2.5 Training Maximum Entropy Language Models

We have introduced the basic ideas of iterative scaling for training maximum entropy models in general in Section 2.1.3. In this section we show the state-of-the-art training methods for language models in particular.

In the language model of Equation(2.15),

$$m(y|x) = \frac{1}{z_\lambda(x)} \prod_{k=1}^{K} e^{\lambda_k \cdot g_k(x,y;A_k)}, \qquad z_\lambda(x) = \sum_y \prod_{k=1}^{K} e^{\lambda_k \cdot g_k(x,y;A_k)}.$$

To train λ's by either GIS or IIS, we need to compute the expectations of all features g_k under a model m as

$$m[g_k] = \sum_{x,y} m(x,y) \cdot g_k(x,y) = \sum_{\langle x,y\rangle : g_k(x,y)=1} m(x) \cdot m(y|x), \text{ for } k = 1, \cdots, K, \tag{2.24}$$

or the partial counts of the expectations as:

$$m_j[g_k] = \sum_{\langle x,y\rangle : g_{k\#}(x,y)=j} m(x) \cdot m(y|x) \cdot g_k(x,y), \text{ for } k = 1, \cdots, K \tag{2.25}$$

where $j = 1, \cdots, \max g_{k\#}(x,y)$ and when we use the abbreviated notation $g_k(x,y) = g_k(x,y;A_k)$ for simplicity.

For each feature g_k, we must find all $\langle x,y\rangle$ for which g_k is active and $m(x)m(y|x)$ is non-zero. If that feature is a marginal one whose value is independent of x, it has $|\mathcal{X}|$ nonzero terms in the summation. Since there are $|\mathcal{Y}|$ marginal features, the training time per iteration is at least $O(|\mathcal{X}| \cdot |\mathcal{Y}|)$. For a simple trigram model, this complexity is $O(|\mathcal{Y}|^3)$ as $\mathcal{X} = \mathcal{Y} \times \mathcal{Y}$, which is intractable even for a small task such as Switchboard with $|\mathcal{Y}| \approx 20,000$.

Berger, Della Pietra & Della Pietra (1996) made the observation that the empirical distribution $\hat{p}(x)$ may be used as our marginal $m(x)$ on $\mathcal{X}$ to simplify the computation if we are only interested in the conditional probability $m(y|x)$ rather than the joint one $m(x, y)$. Equations (2.24) and (2.25) can thus be approximated as

$$m[g_k] = \sum_{\langle x,y \rangle} m(x,y) \cdot g_k(x,y) \approx \sum_{\langle x,y \rangle : g_k(x,y)=1} \hat{p}(x) \cdot m(y|x), \text{ for } k = 1, \cdots, K$$

and

$$m_j[g_k] = \sum_{\langle x,y \rangle : g_{k\#}(x,y)=j} m(x,y) \cdot g_k(x,y) \approx \sum_{\langle x,y \rangle : g_{k\#}(x,y)=j} \hat{p}(x) \cdot m(y|x), \text{ for } k = 1, \cdots, K,$$

respectively. This implementation takes only $O(|\hat{X}| \cdot |\mathcal{Y}|)$ time where the set of "seen" histories $\hat{X}$ is much smaller than $\mathcal{X}$, and its size $|\hat{X}|$ is strictly bounded by the amount of training data.

In equations (2.13) and (2.14), $m(y|x)$ is uniquely decided by α's that are the only unknown parameters of the model to be trained. We take advantage of the fact that $g_k(x, y)$ is either 0 or 1, and we obtain the GIS equation for undating $\alpha_k(x, y)$ as

$$\alpha_k^{(n+1)} = \alpha_k^{(n)} \cdot \frac{a_k}{m[g_k]} \tag{2.26}$$

where $m[g_k]$ is the expectation of g_k.

Applying the IIS algorithm, the update equation (2.26) for individual α's is replaced by

$$\alpha_k^{(n+1)} = \alpha_k^{(n)} \cdot u_i \tag{2.27}$$

where u_i is the solution to the simultaneous equations

$$\sum_{j=0}^{\max g_{k\#}} m_j[g_k] \cdot u_i^j = a_k, \tag{2.28}$$

$$m_j[g_k] = \sum_{\langle x,y \rangle : g_{k\#}(x,y)=j} \hat{p}(x) \cdot m(y|x) \cdot g_k(x,y). \tag{2.29}$$

Since IIS converges faster than GIS, we use IIS to update α's in our work.

2.5.1 Computing the Normalization Factors

We need first to calculate the normalization factor $z(x)$ for each history according to (2.15) in order to compute $m(y|x)$, and then compute the estimation of all feature expectations $m[g_k]$. We introduce the state-of-the-art approach (as used before this dissertation) for computing z in this section and for computing $m[g_k]$ in the next section.

In the straight-forward implementation (2.15), it takes $O(|\mathcal{Y}|)$ time to compute the normalization factor $z(x)$ for each $x \in \hat{X}$. IBM researchers treat the marginal features and other features in different ways, and take advantage of marginal features (Jelinek, 1997). Feature functions in G are partitioned into two subsets: the marginal-feature subset G_m that contains all features independent of history, and the conditional-feature subset G_c that contains the rest of features. We also write marginal features $g_k(x, y)$ as $g_k(y)$ for the sake of simplification. For each "history" x, the vocabulary V is split into subsets Y_x and $V - Y_x$ where

$$Y_x = \{y : g_k(x, y) = 1 \text{ for some } g_k \in G_c\},$$

where Y_x contains all y's that have a conditional constraint in the given context x. Let

$$I(y) = \{k : g_k(y) = 1\}$$

be the indices corresponding to marginal (unigram) constraints[10] on y. Setting

$$r(y) = \prod_{k \in I(y)} \alpha_k$$

to be the factor corresponding to unigram constraints on the word y, the calculation of $z(x)$ is simplified as follows

$$z(x) = \sum_{y \in Y} r(y) + \sum_{y \in Y_x} (r(x, y) - r(y)). \tag{2.30}$$

[10]In the case of the basic N-gram model, there is only one such k for each y. However, in some language models, such as the topic-sensitive model described later, there can be more than one such k.

It should be noted that the first term in (2.30) can be pre-calculated once in $O(|\mathcal{Y}|) = O(|V|)$ time for all seen x's, and the second term requires $O(|\hat{X}| \cdot \bar{C})$ time for all x, where $\bar{C}$ is the average size of Y_x, i.e.,

$$\bar{C} = \frac{1}{|\hat{X}|} \sum_{x \in \hat{X}} |Y_x|.$$

The total running time for all $z(x)$'s is $O(|\mathcal{Y}| + |\hat{X}| \cdot \bar{C})$. Typically, $\bar{C}$ is two orders of magnitude smaller than the vocabulary size $|\mathcal{Y}|$, resulting in a considerable computational speed-up relative to $O(|\hat{X}| \cdot |\mathcal{Y}|)$. This implementation has been called *unigram-caching* in the literature.

Unigram-caching, however, is still impractical for a large corpus, especially when constraints other than N-grams are considered, where the number of distinct histories increases by several orders of magnitude and $\bar{C}$, which is actually a function of the vocabulary size $|V|$ and the size L of the training corpus, also becomes extremely large. Obviously, the most computationally expensive part in (2.30) is in the second term.

2.5.2 Computing Feature Expectations

The other half of computation required for maximum entropy modeling is to calculate the expectations of our features with respect to the joint model

$$m[g_k] = \sum_{\langle x,y \rangle : g_k(x,y)=1} \hat{p}(x) \cdot m(y|x). \tag{2.31}$$

For each feature g_k we need to list all $< x, y >$ for which g_k is active and $m(y|x)$ is nonzero. If the feature g_j is a marginal one, it has $|\hat{X}|$ nonzero $m(y|x)$'s. The total calculation for all $m[g_k(y)]$'s is $O(|\hat{X}| \cdot |\mathcal{Y}|)$ since the number of marginal features is $|\mathcal{Y}|$.

It should be noted that for each marginal g_j, there exists a y such that $g_j(x, y) = 1$ for all $x \in \hat{X}$. But for the same y, only a few x's have nonzero conditional features $g_k(x, y)$'s. We can take advantage of this to reduce the computation as described next.

$$\begin{aligned} m[g_j] &= \sum_{x\in\hat{X}} \hat{p}(x)\cdot m(y|x) \\ &= \alpha(y)\sum_{x\in\hat{X}} \alpha(x,y)\cdot\frac{\hat{p}(x)}{z(x)} \\ &= \alpha(y)\sum_{x\in\hat{X}}(\alpha(x,y)-1)\cdot\frac{\hat{p}(x)}{z(x)} + \alpha(y)\sum_{x\in\hat{X}}\frac{\hat{p}(x)}{z(x)} \end{aligned} \tag{2.32}$$

where y can be regarded as fixed in the above equations. Now the running time is $O(|\hat{X}|\cdot\bar{C}+|\mathcal{Y}|)$ [11] instead of $O(|\hat{X}|\cdot|\mathcal{Y}|)$. *It should be noted that amount of the computation for the normalization factors and that for the estimate of parameters is the same. The former is a summation for each seen history over all words following that history in the training data, while the latter one is a summation for each word over all seen histories preceding it.*

More efficient training methods for computing both will be discussed in the next chapter.

2.5.3 Updating α by Newton's Method

The last step in training is to find the solution u_k to equations (2.28) and then to update the corresponding α' by (2.27). We use Newton's method to find the root of (2.28). We rewrite Equation (2.28) as

$$f_k(u) = \sum_{j=0}^{\max g_{k\#}} m_j[g_k]\cdot u^j - a_k, \text{ for } k=1,\cdots,K.$$

The following algorithm gives a numerical solution for $f(u)=0$.

Algorithm (Newton's Method)

Initial Step: Set initial value u_0 for u and a small number $\epsilon > 0$.
Iteration Steps: While $|f(u)|\geq\epsilon$ do

[11] $\sum_{x\in\hat{X}}\frac{p(x)}{z(x)}$ can be pre-calculated in time of $O(|\hat{X}|)$ so the total running time for $\alpha(y)\sum_{x\in\hat{X}}\frac{\hat{p}(x)}{z(x)}$ is $O(|\hat{X}|+|\mathcal{Y}|)$. Furthermore $g_k(x,y)=1$ only for $<x,y>$ active, so the computation for all $\alpha(y)\sum_{x\in\hat{X}}(\alpha_k^{g_k(x,y)}-1)\cdot\frac{\hat{p}(x)}{z(x)}$ is $O(|\hat{X}|\cdot\bar{C})$.

$$\begin{aligned} \delta &= -\frac{f'(u)}{f(u)}, \\ u &= u - \delta. \end{aligned}$$

∎

Readers may refer to Numerical Recipes (2002) (or any textbook for numerical analysis) for details about Newton's method.

Chapter 3

Hierarchical Training Methods

Training a maximum entropy model is usually a computationally intensive task. This drawback of the maximum entropy approach has prevented many people from using this method. In this chapter, we will describe some efficient training methods for maximum entropy models. Preliminary results have been introduced in Wu & Khudanpur (2000b) and Wu & Khudanpur (2002). We will present more detailed results in this chapter.

3.1 Introduction

Training an ME model using either the generalized iterative scaling or its improved version described by Della Pietra, Della Pietra & Lafferty (1997) needs several iterations for converging. The running time is proportional to both the number of iterations and the time for each iteration. The training procedure inside each iteration can be separated into three phases: computing normalization factors z for all the histories seen in training, calculating feature expectations $p[g]$ for each constraint g and updating model parameters α. We study the first two phases of training in this chapter, and the last phase has been discussed in Section 2.5.3. The first two phases are the same for both GIS and IIS. The only difference lies in the updating of model parameters, which takes only a small amount of computation in each iteration. Therefore, the total training time is predominantly the time for computing z and $p[g]$.

The crucial idea of training maximum entropy models efficiently is to take advantage of hierarchical structures among features, i.e., to share the computation corresponding to lower order features among high order features nesting them. The computation of z's and that of $p[g]$'s take the same amount of time, as we have shown in the previous chapter, and any improvement made for the former applies to the latter (and vice versa). We therefore focus on improving the computation of normalization factors first, and then transfer the resulting efficient methods for computing z to the computation of feature expectations.

In order to address the training methods for maximum entropy models, we need to introduce the relations between features and the feature hierarchy of maximum entropy models. It will later be shown that the efficiency of training an ME model mainly depends on the extent of computation sharing permitted by the hierarchical structure of this model.

By taking advantage of the hierarchical structure of N-gram features, training N-gram models (and models that look like N-gram models) can be as fast per iteration as training the corresponding back-off models, which is the time for scanning the training data once. However, the major motivation of using the ME method is not to build N-gram models, but rather models with some kinds of non-local dependencies. Unfortunately, training these models cannot take advantage of nested features directly and thus it is computationally intensive. However, the ideas of sharing computation among features can be ported from the training of N-gram models to these kinds of models. There are some nested features (although not all features are nested) in these models, and the computation related to these nested features can still be simplified. As a result, the training time can be made much shorter than that of the state-of-the-art unigram-caching method introduced in the previous chapter, even for more complex models.

For specific kinds of non-N-gram models such as the topic-dependent model, a divide-and-conquer strategy is designed to train them almost as efficiently as training back-off models.

3.1.1 Lower Bound and Upper Bound on Complexity

Training a maximum entropy model per iteration needs at least the time $O(L)$ of scanning the training data once. Even the empirical estimation needs $O(L)$ time. If a training algorithm reaches this lower bound, then it is already optimal.

On the other hand, if the probability of histories is approximated by their relative frequency in the training data, a maximum entropy model needs no more than $O(L \cdot |V|)$ time where $|V|$ is the vocabulary size. Therefore, the upper bound of training any ME model is $O(L \cdot |V|)$. It should be noted that $O(L \cdot |V|) \neq O(L)$ because $|V|$ is actually a function of L and cannot be simply treated as a constant[1]. Using unigram-caching will reduce the training time to $O(|\hat{X}| \cdot \bar{C})$ where $|\hat{X}|$ is the number of different history classes seen in the training data and $\bar{C}$, the average number of words with some conditional features activated for each history class, is also a function of $|V|$ and L. Usually $|\hat{X}|$ is several times smaller than L and $\bar{C}$ is 10-100 times smaller than $|V|$ in language modeling.

In some applications such as part-of-speech tagging and syntactic parsing, $O(|V|)$ is only of the order of 100 and independent of L. A complexity of $O(|\hat{X}| \cdot \bar{C})$ or $O(L \cdot |V|)$ is acceptable in such applications. In language modeling, however, the vocabulary size is more than tens of thousands. Consequently, a time of $O(|\hat{X}| \cdot \bar{C})$ becomes intractable and we are seeking training methods with $O(|\hat{X}| \cdot c)$ complexity where c is significantly smaller than $\bar{C}$.

The remainder of this chapter is organized as follows. We start with an example of training a trigram model and then derive the hierarchical training method for N-gram models in Section 3.2. In Section 3.3, we define three relations between features: nested, overlapping and non-overlapping and describe the feature hierarchy of an ME model by a digraph, in which nodes are feature patterns and arcs are feature relations. The extension of hierarchical training method for the model with non-nested features is described in Section 3.4. The generalized version of the hierarchical method for ME models and the divide-and-conquer strategy for topic models in particular follow

[1]Even if $|V|$ were to be treated as a constant, it is typically $10^3 - 10^4$ and hence not negligible in practice even if it were to be negligible in the $O(\cdot)$ sense.

in Section 3.5 and Section 3.6, respectively. We only describe the computation of the normalization factor z in these sections. The computation of the expectation of feature functions is briefly discussed in Section 3.7. In Section 3.8, we provide some details of training specific language models studied in this dissertation. The efficiency of training methods is evaluated in Section 3.9 based on two tasks, the Switchboard and the Broadcast News.

3.2 Hierarchical Training Method for Maximum Entropy Models with only Nested Features

In an N-gram model

$$p(w_i|w_{i-N+1}, \cdots, w_{i-1}) = \frac{1}{z} \prod_{n=1}^{N} \alpha_{w_{i-n+1}, \cdots, w_i}^{g(w_{i-n+1}, \cdots, w_i)}$$

where

$$g(w_{i-n+1}, \cdots, w_i) = \begin{cases} 1 & w_{i-n+1}, \cdots, w_i \text{ has an n-gram constraint,} \\ 0 & \text{otherwise} \end{cases}$$

are feature functions and $\alpha_{w_{i-n+1}, \cdots, w_i}$ are corresponding parameters. If a word w_i has an order n feature $g(w_{i-n+1}, \cdots, w_i)$ active in the given history $w_{i-N+1}, \cdots, w_{i-1}$, it also usually has an order $n-1$ feature $g(w_{i-n+2}, \cdots, w_i)$ activated (for $2 \leq n < N$). Therefore, $g(w_{i-n+1}, \cdots, w_i)$ is nested in a feature $g(w_{i-n+2}, \cdots, w_i)$ for $2 \leq n \leq N$. Figure 3.1 shows the sets of words Y_1, Y_2, $\cdots$, Y_N with unigram, bigram,$\cdots$, and N-gram features activated respectively for a given history $w_{i-N+1}, \cdots, w_i$.

We will show in this section that this kind of model can be trained hierarchically, and training such a model for one iteration takes the same amount of time as the calculation of empirical expectations. We demonstrate the hierarchical training method by a trigram model and then extend this method to any model with only nested features.

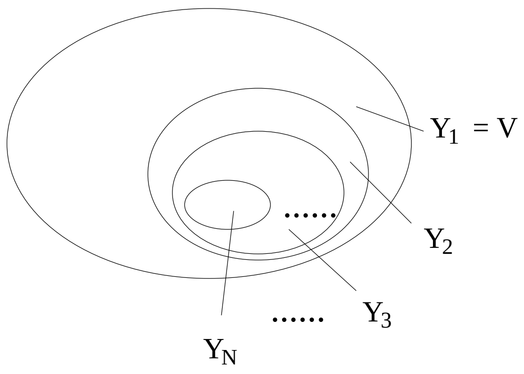

Figure 3.1: Word sets with unigram, bigram, $\cdots$, N-gram features activated.

3.2.1 Computing Normalization Factors in a Trigram Model

Recall the trigram model

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{\alpha_{w_i}^{g(w_i)} \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)}}{z(w_{i-2}, w_{i-1})} \tag{3.1}$$

where

$$z(w_{i-2}, w_{i-1}) = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)}.$$

Subscripts of α's here indicate the indices of corresponding features in the feature set. The order of features g is indicated by the number of coefficients of g and may be omitted without causing any confusion. The normalization factor z can be computed as

$$z(w_{i-2}, w_{i-1}) = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} + \sum_{w_i \in Y_x} [\alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} - 1] \cdot \alpha_{w_i}^{g(w_i)} \tag{3.2}$$

as described in Chapter 13, page 226, in Jelinek (1997). According to the analysis in Section 2.5.1, the second term in Equation (3.2)

$$\sum_{w_i \in Y_x} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1) \cdot \alpha_{w_i}^{g(w_i)} \tag{3.3}$$

with complexity of $O(\sum_{x \in \hat{X}} |Y_x|)$ dominates the computation. Therefore, in the future, the short-hand notation $O(|\hat{X}| \cdot \bar{C})$ will be used for $O(\sum_{x \in \hat{X}} |Y_x|)$ for simplicity, where $\bar{C} = \frac{1}{|\hat{X}|} \sum_{x \in \hat{X}} |Y_x|$.

Most of the words in Y_x have only bigram features active. For example, in any given history x, the set of words with bigram features contains on average roughly 300 words and 6000 words respectively in Switchboard and Broadcast News, among which only 2 or 3 (on average) have trigram or higher order constraints. We take advantage of this fact and split the word set Y_x into Y_2 and $Y_x - Y_2$ where

$$Y_2 = \{w_i : g(w_{i-1}, w_i) = 1 \text{ for some bigram feature}\}$$

and also into Y_3 and $Y_x - Y_3$ where

$$Y_3 = \{w_i : g(w_{i-2}, w_{i-1}, w_i) = 1 \text{ for some trigram feature}\}.$$

Note that Y_2 and Y_3 are history dependent even though we omit the subscripts x for the sake of simplicity. Now, since $Y_x = Y_2 \cup Y_3$ and $\overline{Y}_2 \cap Y_3 = \phi$, Y_x is split into three disjoint subsets as shown in Figure 3.2

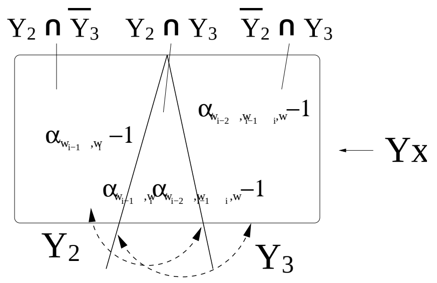
$Y_2 \cap \overline{Y_3}$
$Y_2 \cap Y_3$
$\overline{Y_2} \cap Y_3$
$\alpha_{w_{i-2},w_{i-1},w_i} - 1$
$\alpha_{w_{i-1},w_i} - 1$
$\alpha_{w_{i-1},w_i}\alpha_{w_{i-2},w_{i-1},w_i} - 1$
Yx
Y_2
Y_3

Figure 3.2: Splitting Y_x according to Y_2 and Y_3. The values of $(\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1)$ in different cases are shown.

Obviously, for any sum over all words in Y_x,

$$\sum_{Y_x} * = \sum_{Y_2 \cap \overline{Y_3}} * + \sum_{\overline{Y_2} \cap Y_3} * + \sum_{Y_2 \cap Y_3} *.$$

Table 3.1 gives the value of $g(w_{i-1}, w_i)$ and $g(w_{i-2}, w_{i-1}, w_i)$ for words w_i in the three subsets above. Applying the values in Table 3.1 to the sum in (3.3), we have

$$\begin{aligned} &(\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1)\alpha_{w_i}^{g(w_i)} \\ = &\begin{cases} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & w_i \in Y_2 \cap \overline{Y_3} \\ (\alpha_{w_{i-2},w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & w_i \in \overline{Y_2} \cap Y_3 \ , \\ (\alpha_{w_{i-1},w_i} \cdot \alpha_{w_{i-2},w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & w_i \in Y_2 \cap Y_3 \end{cases} \end{aligned}$$

word set	$g(w_{i-1}, w_i)$	$g(w_{i-2}, w_{i-1}, w_i)$
$Y_2 \cap \overline{Y_3}$	1	0
$\overline{Y}_2 \cap Y_3$	0	1
$Y_2 \cap Y_3$	1	1

Table 3.1: Value of $g(w_{i-1}, w_i)$ and $g(w_{i-2}, w_{i-1}, w_i)$

and therefore

$$\begin{aligned} & \sum_{w_i \in Y_x} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1)\alpha_{w_i}^{g(w_i)} \\ = & \sum_{w_i \in Y_2} (\alpha_{w_{i-1},w_i} - 1)\alpha_{w_i} + \sum_{w_i \in Y_3} (\alpha_{w_{i-2},w_{i-1},w_i} - 1)\alpha_{w_{i-1},w_i}\alpha_{w_i}. \end{aligned} \tag{3.4}$$

We merge (3.2) and (3.4) into

$$\begin{aligned} z(w_{i-2}, w_{i-1}) &= \sum_{w_i \in V} \alpha_{w_i} + \sum_{w_i \in Y_2} (\alpha_{w_{i-1},w_i} - 1)\alpha_{w_i} \\ &+ \sum_{w_i \in Y_3} (\alpha_{w_{i-2},w_{i-1},w_i} - 1)\alpha_{w_{i-1},w_i}\alpha_{w_i}. \end{aligned} \tag{3.5}$$

Analysis of Complexity

Now we analyze the computational complexity of implementation of the sum in (3.5).

- The first term on the right-hand size of (3.5) takes $O(U)$ time to calculate, where U is the number of unigrams in the training data.

- The sum over Y_2 in the right-hand side of the equation is fixed for all histories sharing a suffix w_{i-1} and need be computed only once for each word w_{i-1} in the vocabulary. If for a given w_{i-1}, Y_2 contains only those words following w_{i-1} in the training data, the size of Y_2 is exact the number of bigrams beginning with w_{i-1}. The number of combinations of w_{i-1} and Y_2 is exactly the number of bigrams in the training data denoted as B. The running time for the second summation over Y_2 is thus $O(B)$ for all histories.

- The sum over Y_3 depends on both w_{i-2} and w_{i-1}. Each Y_3 for a given history class w_{i-2} and w_{i-1} contains only a few words that have trigram constraints, i.e., those words following w_{i-2}, w_{i-1} in the training data. The cumulative size of all Y_3's for different w_{i-2}, w_{i-1} is thus no more than the number of seen trigrams started by w_{i-2}, w_{i-1}, denoted as T. The running time of the third summation for all histories is thus $O(T)$.

Overall, the complexity of the implementation (3.5) is therefore $O(U + B + T)$ for one iteration.

Discussion

In language modeling, training an interpolated trigram model (1.1) or a back-off trigram model (1.3) takes exactly $O(U+B+T)$ time because the empirical frequency for all unigrams, bigrams and trigrams must be collected.

Even though the value of feature functions g is binary in language modeling, the implementation (3.5) does not require g to be binary. The equation still holds when g is a real function and can be rewritten as

$$\begin{aligned} & z(w_{i-2}, w_{i-1}) \\ = & \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} + \sum_{w_i \in Y_2} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} - 1) \cdot \alpha_{w_i}^{g(w_i)} \\ + & \sum_{w_i \in Y_3} (\alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1) \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i))} \cdot \alpha_{w_i}^{g(w_i)}. \end{aligned}$$

3.2.2 Extension for N-gram Models

The training method discussed in the previous section can be extended to N-gram models with any arbitrary value of N. Without the loss of generality, we can assume each N-gram seen in the training data corresponds to a feature function[2].

In an N-gram model,

$$p(w_i | w_{i-N+1}, \cdots, w_{i-1}) = \frac{1}{z} \prod_{j=1}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}, \tag{3.6}$$

[2]If some N-grams are not included in the feature set, we can simply assume that they correspond to features g_k whose parameters equal $\alpha_k = 1$.

where

$$\begin{aligned} z &= z(w_{i-N+1}, \cdots, w_{i-1}) \\ &= \sum_{w \in V} \prod_{j=1}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}. \end{aligned} \tag{3.7}$$

We first apply unigram-caching to (3.7) and rewrite z as

$$z = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} + \sum_{w_i \in Y_x} \left[\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_{i-1},w_i}^{g(w_{i-j+1},\cdots,w_{i-1},w_i)} - 1 \right] \alpha_{w_i}^{g(w_i)}. \tag{3.8}$$

Now we use the computational trick for the trigram model recursively to the N-gram model. We split the word set Y_x into Y_2 and $\overline{Y_2} = Y_x - Y_2$ where

$$Y_2 = \{w_i : g(w_{i-1}, w_i) = 1 \text{ for some bigram feature}\}$$

as we have done in the previous section. However, instead of defining a subset Y_3 as we have done for trigram models, we define

$$Y_3' = \{w_i : g(w_{i-n+1}, \cdots, w_i) = 1 \text{ for some n-gram features}, n \geq 3\}$$

instead. Y_x can be split into Y_3' and $\overline{Y_3'} = Y_x - Y_3'$.

Obviously, $Y_x = Y_2 \cup Y_3'$ and $\overline{Y_2} \cap \overline{Y_3'} = \phi$. Y_x is split into three disjoint subsets: $Y_2 \cap \overline{Y_3'}$, $\overline{Y_2} \cap Y_3'$ and $Y_2 \cap Y_3'$. This splitting shown in Figure 3.3 is similar to the one for training trigram models.

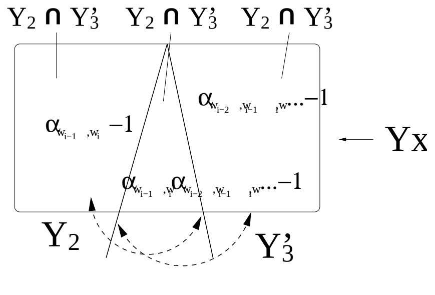
$Y_2 \cap \overline{Y_3'}$
$Y_2 \cap Y_3'$
$\overline{Y_2} \cap Y_3'$
$\alpha_{w_{i-1},w_i} - 1$
$\alpha_{w_{i-2},w_{i-1},w_i} \cdots - 1$
$\alpha_{w_{i-1},w_i} \alpha_{w_{i-2},w_{i-1},w_i} \cdots - 1$
Yx
Y_2
Y_3'

Figure 3.3: Splitting Y_x according to Y_2 and Y_3'. The value of $\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_{i-1},w_i}^{g(w_{i-j+1},\cdots,w_{i-1},w_i)} - 1$ is shown.

Words in $Y_2 \cap \overline{Y_3'}$ have only bigram features $g(w_{i-1}, w_i)$ activated, while words in $\overline{Y}_2 \cap Y_3'$ have only order 3 or higher features $g(w_{i-2}, w_{i-1}, w_i)$, $g(w_{i-3}, w_{i-2}, w_{i-1}, w_i)$,

$\cdots$, or $g(w_{i-N+1}, \cdots, w_i)$ activated. Only words in $Y_2 \cap Y_3'$ have both bigram features and some higher order features are activated.

By taking advantage of the fact that all feature functions g's are binary, the value of the term in the second sum in (3.8) can be rewritten as

$$\left[\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1\right] \cdot \alpha_{w_i}^{g(w_i)}$$
$$= \begin{cases} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & w_i \in Y_2 \cap \overline{Y_3'} \\ (\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1) \cdot \alpha_{w_i} & w_i \in \overline{Y_2} \cap Y_3' \\ (\alpha_{w_{i-1},w_i} \prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1) \cdot \alpha_{w_i} & w_i \in Y_2 \cap Y_3' \end{cases}, \quad (3.9)$$

in different cases. Note that $g(w_{i-j+1}, \cdots, w_i) = 0$ is equivalent to setting $\alpha_{w_{i-j+1},\cdots,w_i} = 1$ in the equations above.

The second summation in (3.8) is thus calculated as

$$\sum_{w_i \in Y_x} \left[\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1\right] \alpha_{w_i}^{g(w_i)} \quad (3.10)$$
$$= \sum_{w_i \in Y_2 \cap \overline{Y_3'}} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i}$$
$$+ \sum_{w_i \in \overline{Y_2} \cap Y_3'} \left[\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1\right] \alpha_{w_i}$$
$$+ \sum_{w_i \in Y_2 \cap Y_3'} \left[\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1\right] \alpha_{w_i}$$
$$= \sum_{w_i \in Y_2 \cap \overline{Y_3'}} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i}$$
$$+ \sum_{w_i \in \overline{Y_2} \cap Y_3'} \left[\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1\right] \cdot \alpha_{w_{i-1},w_i} \cdot \alpha_{w_i}{}^{3}$$
$$+ \sum_{w_i \in Y_2 \cap Y_3'} \left[\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - \alpha_{w_{i-1},w_i} + \alpha_{w_{i-1},w_i} - 1\right] \cdot \alpha_{w_i}$$

[3]Note: $\alpha_{w_{i-1},w_i} = 1$ here.

$$\begin{aligned} &= \sum_{w_i \in Y_2 \cap \overline{Y_3'}} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & (3.11) \\ &\quad + \sum_{w_i \in \overline{Y_2} \cap Y_3'} \left[\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1 \right] \cdot \alpha_{w_{i-1},w_i} \cdot \alpha_{w_i} \\ &\quad + \sum_{w_i \in Y_2 \cap Y_3'} \left[\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1 \right] \cdot \alpha_{w_{i-1},w_i} \cdot \alpha_{w_i} \\ &\quad + \sum_{w_i \in Y_2 \cap Y_3'} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} \\ &= \sum_{w_i \in Y_2} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} & (3.12) \\ &\quad + \sum_{w_i \in Y_3'} \left[\prod_{j=3}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1 \right] \cdot \alpha_{w_{i-1},w_i} \cdot \alpha_{w_i}. & (3.13) \end{aligned}$$

The summation (3.12) for all histories takes $O(B)$ time as we have shown in the previous section. Even if we stop here, the above implementation with the complexity of $O(B + |\hat{X}| \cdot |Y_3'|)$ is already much more efficient than that of unigram-caching $(O(|\hat{X}| \cdot |Y_x|))$ because $|Y_3'| << |Y_x|$ on average. However, we can further optimize the training algorithm. The second summation (3.13) has the same form as (3.10), and can be computed recursively by applying the same strategy.

In the remainder of this section, we formally state the hierarchical training method and prove its correctness. We begin with the following lemma that gives the recursive equation for the second summation above.

Lemma 1

In the N-gram model (3.6), $g(w_{i-n+1}, \cdots, w_i)$ for $n = 1, \cdots, N$, denotes an n-gram feature function and $\alpha_{w_{i-n+1},\cdots,w_i}$ is its corresponding parameter. Let

$$\begin{aligned} Y_n = Y_n(x) &= \{w_i : g(w_{i-n+1}, \cdots, w_i) = 1\}, \text{ and} & (3.14) \\ Y_n' = Y_n'(x) &= \{w_i : g(w_{i-k+1}, \cdots, w_i) = 1, \text{ for some } k \geq n\}, & (3.15) \end{aligned}$$

for $n = 1, 2, \cdots, N$. Then, for any given history class $w_{i-N+1}, \cdots, w_i$,

$$\sum_{w \in Y'_n} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} \tag{3.16}$$

$$= \sum_{w \in Y_n} (\alpha_{w_{i-n+1},\cdots,w_i}^{g(w_{i-n+1},\cdots,w_i)} - 1) \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$+ \sum_{w_i \in Y'_{n+1}} [\prod_{j=n+1}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1] \prod_{j=1}^{n} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}.$$

Proof: $Y'_n = Y_n \cup Y'_{n+1}$ and $\overline{Y_n} \cap \overline{Y'_{n+1}} = \phi$ according to the definition of Y_n and Y'_n. Y'_n can be decomposed to three disjoint subsets: $Y_n \cap Y'_{n+1}$, $\overline{Y_n} \cap Y'_{n+1}$, and $Y_n \cap \overline{Y'_{n+1}}$. Note that words in $Y_n \cap \overline{Y'_{n+1}}$ have no order $n+1$ or higher features activated, those in $\overline{Y_n} \cap Y'_{n+1}$ do not have n^{th} order features activated and only words in $Y_n \cap Y'_{n+1}$ have both activated. Therefore,

$$\sum_{w_i \in Y'_n} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} \tag{3.17}$$

$$= \sum_{w_i \in Y_n \cap \overline{Y'_{n+1}}} (\alpha_{w_{i-n+1},\cdots,w_i}^{g(w_{i-n+1},\cdots,w_i)} - 1) \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$+ \sum_{w_i \in \overline{Y_n} \cap Y'_{n+1}} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$+ \sum_{w_i \in Y_n \cap Y'_{n+1}} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$= \sum_{w_i \in Y_n \cap \overline{Y'_{n+1}}} (\alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1) \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$+ \sum_{w_i \in \overline{Y_n} \cap Y'_{n+1}} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \cdot \prod_{n=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$+ \sum_{w_i \in Y_n \cap Y'_{n+1}} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - \alpha_{w_{i-n+1},\cdots,w_i}^{g(w_{i-n+1},\cdots,w_i)} + \alpha_{w_{i-n+1},\cdots,w_i}^{g(w_{i-n+1},\cdots,w_i)} - 1 \right] \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}$$

$$= \sum_{w_i \in Y_n} (\alpha_{w_{i-n+1},\cdots,w_i}^{g(w_{i-n+1},\cdots,w_i)} - 1) \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} \tag{3.18}$$

$$+ \sum_{w_i \in Y'_{n+1}} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1 \right] \cdot \prod_{j=1}^{n} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}. \tag{3.19}$$

∎

This proves the assertion of the lemma.

Note the first summation (3.18) takes $O(\#[\text{n-gram}])$ time, and the second one (3.19) has the same form as (3.17) does and can be computed recursively.

We summarize the above ideas in the following theorem.

Theorem

In the N-gram model of (3.6), the normalization factor $z(x)$ may be computed as

$$z(x) = \sum_{w_i \in V} \alpha_{w_i} + \sum_{n=2}^{N} \sum_{w_i \in Y_n} \left[\prod_{j=n}^{N} \alpha_{w_{i-j+1},\cdots,w_i} - 1 \right] \cdot \prod_{j=1}^{n-1} \alpha_{w_{i-j+1},\cdots,w_i} \tag{3.20}$$

for all x in $O(\sum_{n=1}^{N} \#[\text{n-gram}])$ time, where Y_n, $n = 2, \cdots, N$, are defined in (3.14) and α's are set to one if the corresponding g's are zero.

Proof:

$$z(x) = \sum_{w_i} \prod_{j=1}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)}.$$

Define Y_n and Y'_n as in (3.14) and (3.15). Start from

$$z(x) = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} + \sum_{w_i \in Y_x} [\prod_{j=2}^{N} \alpha_{w_{i-j+1},\cdots,w_i}^{g(w_{i-j+1},\cdots,w_i)} - 1] \cdot \alpha_{w_i}^{g(w_i)},$$

apply Lemma 1 recursively $N-1$ times with $n = 2, \cdots, N-1$ to the second term in the right hand side of the equation above and take advantage of g being binary to derive (3.20).

The n^{th} inner summation $\sum_{w \in Y_n}$ in (3.20) has exact #[n-gram] terms for all histories. The overall complexity for computing z for all histories in $|\hat{X}|$ is strictly bounded by $O(\sum_{n=1}^{N} \#[\text{n-gram}])$.

∎

In the algorithm above, the first summation involves unigram features only; the second one involves unigram and bigram features, etc. The normalization factors are thus computed hierarchically. We call this method, the *hierarchical training* method.

Time Complexity

In language modeling, the computation time for all denominators z per iteration is $O(\sum_{n=1}^{N} \#[\text{n-gram}])$ for all $n = 1, 2, \cdots, N$, which is the same as that for training a corresponding interpolated or back-off N-gram model. However, a maximum entropy model needs several iterations, whereas interpolation and back-off models need only one.

Space complexity

The theorem above gives a constructive algorithm for computing the normalization factors z hierarchically. First, the summation over V is computed, then that over Y_2, etc. This algorithm needs some extra space to save the intermediate summations over V, Y_2, Y_3, etc. If N-grams are sorted lexicographically according to $w_i, w_{i-1}, w_{i-2}, \cdots, w_{i-N+1}$, the algorithm requires only $O(N)$ extra space more than that needed for unigram-caching.

The theorem above also holds when g's are real-valued functions. The following corollary extends the theorem for the case when the g's are not binary.

Corollary

In a maximum entropy model

$$p(y|x) = \frac{1}{z(x)} \prod_{j=0}^{N} \alpha_{x_1,\cdots,x_j,y}^{g(x_1,\cdots,x_j,y)} \tag{3.21}$$

where the feature functions are defined as

$$g(x_1,\cdots,x_j,y) \begin{cases} \neq 0 & \langle x_1,\cdots,x_j,y\rangle \text{in the training data} \\ = 0 & otherwise \end{cases} \quad \text{for } j = 1,\cdots,N, \tag{3.22}$$

the normalization factor $z(x)$ can be computed as

$$z(x) = \sum_{y\in\mathcal{Y}} \alpha_y^{g(y)} + \sum_{n=1}^{N} \left[\sum_{y\in Y_n} [\prod_{j=n}^{N} \alpha_{x_1,\cdots,x_j,y}^{g(x_1,\cdots,x_j,y)} - 1] \cdot \prod_{j=0}^{n-1} \alpha_{x_1,\cdots,x_j,y}^{g(x_1,\cdots,x_j,y)} \right]$$

where

$$Y_n \;=\; \{w : g(x_1,\cdots,x_{n-1},w) \neq 0 \text{ for some n-gram feature } g\}.$$

Time to compute the normalization factors z for all histories is $O(\sum_{n=0}^{N} S_n)$ for all x where $S_0 = V$ and S_n is the number of $n+1$-tuples $\langle x_1,\cdots,x_n,y\rangle$ seen in the training data. This time is exactly the same as that to gather empirical counts from the training data.

∎

3.3 Exploiting the Feature Hierarchy for Computing Maximum Entropy Models

N-gram models are a particular kind of ME model. Obviously if a word w_i has the n-gram feature activated for a given history $w_{i-n+1},\cdots,w_{i-1}$ then it must have $(n-1)$-gram, $(n-2)$-gram, ..., unigram features activated too. Therefore, for a given history, $Y_N \subset Y_{N-1}\cdots Y_2 \subset Y_1 = V$, as we have shown in the previous section. The efficiency of the hierarchical training method introduced in this previous section comes from the nested relation among N-gram features.

In the real world, however, many models have non-nested features in addition to nested ones. The training of these models is more complicated than that of N-gram models. The complexity of computing model parameters depends on the hierarchical structure of feature types in the model. Before we address the training methods for maximum entropy models in general, we need to describe the feature hierarchy of ME models in this section.

3.3.1 Relations between Features

We have shown that N-gram features are nested. Here we give a more formal definition for the nested relation between features.

Definition 1 (Nested Features):

Let $x' = x_{k_1}, \cdots, x_{k_l}$ be a history class and $x'' = x_{j_1}, \cdots, x_{j_m}, m < l$ be a subsequence of x'. $g_1(x', y)$ is identified by x' and y and $g_2(x'', y)$ is identified by x'' and y. If it holds that $g_1 = 1 \Rightarrow g_2 = 1$, then we say that g_1 is nested in g_2.

Example 1

The trigram feature function $g(w_{i-2}, w_{i-1}, w_i)$ is nested in the bigram feature function $g(w_{i-1}, w_i)$, and both the trigram and the bigram features are nested in the unigram feature $g(w_i)$. However, the topic-dependent feature $g(t_i, w_i)$ and the trigram feature function $g(w_{i-2}, w_{i-1}, w_i)$ are not nested in either direction since neither t_i is a subsequence of w_{i-2}, w_{i-1} nor vice verse.

Definition 2 (Nested Feature Types):

We say feature type-I is nested in feature type-II if any feature in the feature type-I is nested to a feature in type-II, i.e.,

$$\forall g_1 \in \text{type-I}, \exists g_2 \in \text{type-II such that } g_1 \text{ is nested in } g_2.$$

Example 2

The trigram feature type (pattern) is nested in the bigram feature pattern because for each trigram feature $g(w_{i-2}, w_{i-1}, w_i)$, there is a bigram feature $g(w_{i-1}, w_i)$ that nests it.

Obviously, transitivity holds for the nested relation, i.e., if $g(w_{i-1}, w_i)$ is nested in $g(w_i)$ and $g(w_{i-2}, w_{i-1}, w_i)$ is nested in $g(w_{i-1}, w_i)$, then $g(w_{i-2}, w_{i-1}, w_i)$ is nested in $g(w_i)$.

We further classify features that are not nested as either non-overlapping features or overlapping features.

Definition 3 (Non-overlapping Features and Overlapping Features):

Two features g_j and g_k are *non-overlapping* if $g_j(x, y)$ and $g_k(x, y)$ cannot be active simultaneously for any $y \in V$ a given history $x \in \mathcal{X}$; otherwise, they are special *overlapping* features.

Example 3

For example, if w_i and w_j only occur in topic t_i and t_j, respectively, the features $g(t_i, w_i)$ and $g(t_j, w_j)$ are non-overlapping features. However, the bigram feature $g(w_{i-1}, w_i)$ and the topic feature $g(t_i, w_i)$ may be overlapping features, because for a given history (t_i, w_{i-1}), some words w_i may have both the topic feature and the bigram feature activated.

We can also define the *non-overlapping* and *overlapping* relations between feature types (patterns). If none of the features in type-I overlap with any features in type-II, then we say type-I and type-II are non-overlapping feature patterns.

Pure non-overlapping feature patterns are rare in an ME model, but some features may be treated as non-overlapping features in practice if their overlapping parts are very small. Readers will see that N-gram features and topic-dependent features are treated as non-overlapping features in Chapter 7.

Figure 3.4 shows a Venn diagram of sets of words Y_1 and Y_2 (for a given history x) activating two nested, non-overlapping and overlapping features, respectively.

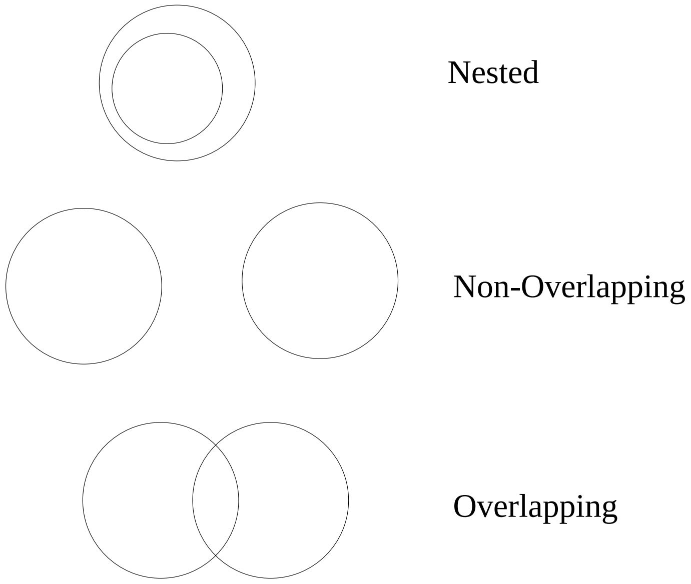

Figure 3.4: Nested, non-overlapping and overlapping features

3.3.2 Representing the Feature Hierarchy of ME Models in a Digraph

After defining the relations between features, we can now draw the feature hierarchy of a maximum model in a digraph. In this digraph, nodes are feature patterns. Two feature patterns $g(x'', y)$ and $g(x', y)$ are connected by a directed edge (or arc) $(g(x'', y), g(x', y))$ if $g(x'', y)$ is nested in $g(x', y)$. Figure 3.5 shows the structure of the topic model (1.9).

If all feature patterns of an ME model are ordered by the nested relation as are those in N-gram models, then the structure of this model is a unary tree as shown in Figure 3.6.

3.4 Expansion for Models with Non-Nested Features

Many maximum entropy models have some non-nested (overlapping) features. Therefore, they cannot be trained by the hierarchical method described in the previous

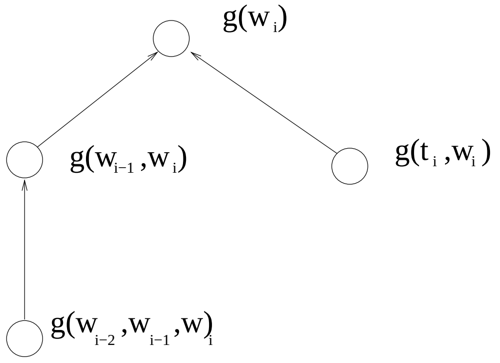

Figure 3.5: Structure of a topic-dependent model

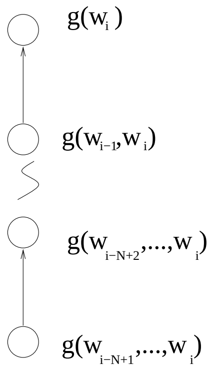

Figure 3.6: Structure of N-gram models

section. However the idea of sharing computation is still applicable. In this section we extend the ideas in Section 3.2 to ME models with non-nested features. We focus on a model whose conditional features are in the same order, or, in other words, the same layer in the feature hierarchy, as shown in Figure 3.7. None of conditional constraints in this kind of model are nested.

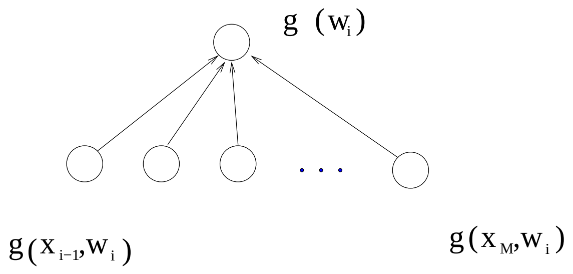

Figure 3.7: The model with a flat feature hierarchy

3.4.1 Training Models with Non-Nested and Overlapping Features

We again start from a simple model: a model with unigram features $g(w_i)$, bigram features $g(w_{i-1}, w_i)$ and the topic features $g(t_i, w_i)$. For this model

$$p(w_i|t_i, w_{i-1}) = \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)}}{z(t_i, w_{i-1})}, \tag{3.23}$$

where

$$z(t_i, w_{i-1}) = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)}.$$

The feature hierarchy of the topic-dependent bigram model is shown in Figure 3.8.

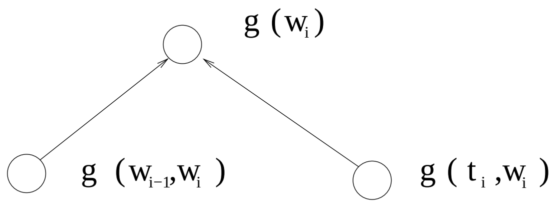

Figure 3.8: Feature hierarchical of the topic-dependent bigram model

The normalization constant z can be computed as

$$z(t_i, w_{i-1}) = \sum_{w_i \in V} \alpha_{w_i}^{g(w_i)} + \sum_{w_i \in Y_x} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} - 1) \cdot \alpha_{w_i}^{g(w_i)} \tag{3.24}$$

by unigram-caching in roughly $O(\sum_{x\in\hat{X}} |Y_x|)$ time for all seen (t_i, w_{i-1}) pairs. We notice that for $w \in \hat{Y}_x$, many of them have either regular bigram constraints or topic constraints but not both. We take advantage of this fact and simplify the computation of calculating z.

Similar to splitting Y_x in (3.24) according to bigram and trigram constraints, when training trigram models, we split Y_x into subsets Y_w and $\overline{Y_w} = Y_x - Y_w$, where

$$Y_w = \{w_i : g(w_{i-1}, w_i) = 1 \text{ for some bigram features }\},$$

and also into Y_t and $\overline{Y_t} = Y_x - Y_t$, where

$$Y_t = \{w_i : g(t_i, w_i) = 1 \text{ for some topic features }\}$$

Table 3.2 shows the values of $g(w_{i-1}, w_i)$ and $g(t_i, w_i)$ for words w_i in $Y_w \cap \overline{Y_w}$, $\overline{Y}_w \cap Y_w$ and $Y_w \cap Y_t$ respectively.

word set	$g(w_{i-1}, w_i)$	$g(t_i, w_i)$
$Y_w \cap \overline{Y_w}$	1	0
$\overline{Y}_w \cap Y_w$	0	1
$Y_w \cap Y_t$	1	1

Table 3.2: Value of $g(w_{i-1}, w_i)$ and $g(t, w_i)$

Figure 3.9 shows the value of $\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t,w_i}^{g(t_i,w_i)} - 1$ in the three subsets of Y_x.

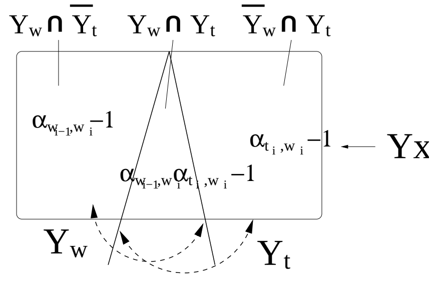
$Y_w \cap \overline{Y_t}$
$Y_w \cap Y_t$
$\overline{Y_w} \cap Y_t$
$\alpha_{w_{i-1},w_i} - 1$
$\alpha_{t_i,w_i} - 1$
$\alpha_{w_{i-1},w_i}\alpha_{t_i,w_i} - 1$
Yx
Y_w
Y_t

Figure 3.9: Division of Y_x according to bigram and topic constraints

Therefore,

$$(\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} - 1)\alpha_{w_i}^{g(w_i)} = \begin{cases} (\alpha_{w_{i-1},w_i} - 1)\alpha_{w_i} & w_i \in Y_w \cap \overline{Y_t} \\ (\alpha_{t_i,w_i} - 1)\alpha_{w_i} & w_i \in \overline{Y_w} \cap Y_t \\ (\alpha_{w_{i-1},w_i} \cdot \alpha_{t_i,w_i} - 1)\alpha_{w_i} & w_i \in Y_w \cap Y_t. \end{cases} \quad (3.25)$$

The second sum of $z(x)$ in (3.24) can thus be re-written as

$$\begin{aligned} & \sum_{w \in Y_x} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} - 1) \cdot \alpha_{w_i}^{g(w_i)} & (3.26) \\ = & \sum_{w_i \in Y_w \cap \overline{Y_t}} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} + \sum_{w_i \in \overline{Y_w} \cap Y_t} (\alpha_{t_i,w_i} - 1) \cdot \alpha_{w_i} \\ + & \sum_{w_i \in Y_w \cap Y_t} [(\alpha_{w_{i-1},w_i} - 1)(\alpha_{t_i,w_i} - 1) + (\alpha_{w_{i-1},w_i} - 1) + (\alpha_{t_i,w_i} - 1)]\alpha_{w_i} \\ = & \sum_{w_i \in Y_w} (\alpha_{w_{i-1},w_i} - 1) \cdot \alpha_{w_i} + \sum_{w_i \in Y_t} (\alpha_{t_i,w_i} - 1) \cdot \alpha_{w_i} \\ + & \sum_{w_i \in Y_w \cap Y_t} (\alpha_{w_{i-1},w_i} - 1) \cdot (\alpha_{t_i,w_i} - 1) \cdot \alpha_{w_i}. & (3.27) \end{aligned}$$

Analysis of Complexity

- The first summation in (3.27) depends only on w_{i-1}. It can be computed in $O(B)$ time for all histories and cached for re-use.

- The second one depends only on t_i, and can be computed in $O(B_t)$, where B_t is the number of topic unigram constraints, and then shared by the histories with the same t_i. Note that these two numbers are strictly smaller than the size of training data L.

- The complexity for the last summation is $O(|Y_w \cap Y_t|)$ time for each history and $O(\sum_{(w_{i-1},t_i) \in \hat{X}} |Y_w \cap Y_t|)$ for all histories.

In summary, computing z for all "seen" histories by (3.27) will take $O(U + B + B_t + \sum_{(w_{i-1},t_i) \in \hat{X}} |Y_w \cap Y_t|)$ time. U, B and B_t are very small compared to $\sum_{(w_{i-1},t_i) \in \hat{X}} |Y_w \cup Y_t|$ and thus can be omitted. Since the baseline unigram-caching method takes

$O(\sum_{(w_{i-1},t_i)\in\hat{X}} |Y_w \cup Y_t|)$ time, on average, the speed-up of using (3.27) is roughly the ratio of $|Y_w \cup Y_t|$ and $|Y_w \cap Y_t|$.

$|Y_w \cap Y_t|$ is very small if $g(w_{i-1}, w_i)$ and $g(t_i, w_i)$ are almost non-overlapping features. On the other hand, the size of this intersection can be as large as $\min(|Y_w|, |Y_t|)$ if $g(w_{i-1}, w_i)$ is almost nested in $g(t_i, w_i)$ or vice versa. This implementation is practical[4] if $|Y_w \cap Y_t| << |Y_w \cup Y_t|$. For example, in Switchboard, $|\hat{X}| \approx 700,000$, $Y_w \approx 200$, $|Y_t| \approx 200$ and $|Y_w \cup Y_t| \approx 330$, so $|\hat{X}| \cdot |Y_w \cup Y_x| \approx 2 \cdot 10^9$. However, $|Y_w \cap Y_t| \approx 50$, $B \approx 300,000$ and $B_t \approx 15,000$, so $|\hat{X}| \cdot |Y_w \cap Y_x| + B + B_t \approx 4 \cdot 10^8$. Using (3.27) to train the model will result in a five-sixfold speed-up compared to using (2.30) directly. In Broadcast News, the speed-up from (3.27) is roughly fourfold. Since the gain of this implementation comes from sharing the computation among the "non-overlapping" part of Y_w and Y_t, we will refer to (3.27) as *non-overlapping sharing* in the remainder of this dissertation.

Unlike N-gram models whose training complexity is strictly bounded by L, i.e., the size of training data, non-nested models usually cannot be trained in $O(L)$ time because $\sum_{(w_{i-1},t_i)\in\hat{X}} |Y_w \cap Y_t|$ may exceed L. For a given history class (w_{i-1}, t_i) in the training data, $Y_w \cap Y_t$ contains words w_i where (w_{i-1}, w_i) occur and (t_i, w_i) occur in the training data no matter whether the 3-tuple $\langle w_{i-1}, t_i, w_i \rangle$ occurs or not. For example, the phrase "a big dog and a small cat" appears in the topic *pets*, but "big cat" does not occur in the topic *pets* even though it does appear in other topics. However, "cat" still needs to be considered in $|Y_w \cap Y_t|$ when the history is "w_{i-1} =big, t_i=pets." As a result, many triple w_{i-1}, w_i, t_i not seen in the training data have to be considered in computing z.

Extra Space Requirement

One minor drawback of non-overlapping sharing is the added space complexity. Summations over Y_t and Y_w must be saved for all topics and words respectively in order to compute the summation over $Y_t \cap Y_w$. Therefore, extra space of $O(|V|)$ plus $O(|T|)$ is required where T is the number of topics. This drawback is minor for only

[4]Although $|Y_w \cap Y_t| < |Y_w \cup Y_t|$ always holds since *intersection is smaller than union*, this implementation is not necessary if the speed-up is not substantial.

two kinds of overlapping features. However, it may becomes a severe problem when a large number of overlapping features are active simultaneously.

3.4.2 Model with M Kinds of Bigram Constraints

The method above can be extended to a model with a unigram feature $g(w)$ and M bigram-like features $g(u_1, w), g(u_2, w), \cdots, g(u_M, w)$:

$$p(w|u_1, \cdots, u_M) = \frac{\alpha_w^{g(w)} \cdot \alpha_{u_1,w}^{g(u_1,w)} \cdots \alpha_{u_M,w}^{g(u_M,w)}}{z(u_1, \cdots, u_M)}, \tag{3.28}$$

where

$$\begin{aligned} z(x) &= z(u_1, \cdots, u_M) \\ &= \sum_w \alpha_w^{g(w)} + \sum_{w \in Y_x} \alpha_w^{g(w)} [\alpha_{u_1,w}^{g(u_1,w)} \cdots \alpha_{u_M,w}^{g(u_M,w)} - 1]. \end{aligned}$$

We again take advantage of the fact that *the intersection of sets is smaller than their union* and split the summation over Y_x into several summations over smaller subsets. We first define "base" word subsets as

$$Y_n = \{w : g(u_n, w) = 1 \text{ for } n = 1, \cdots, M\}.$$

Y_x can be thus split into Y_n and $Y_x - Y_n$ in M ways, one for each n. There are 2^M different intersections I of sets $Y_1, Y_2, \cdots, Y_M$ as listed in Table 3.3.

$$\begin{array}{rclllll} I_0 & = & Y_x & & & & \\ I_1 & = & Y_1 & & & & \\ I_2 & = & & Y_2 & & & \\ I_3 & = & Y_1 & \cap Y_2 & & & \\ & & & \vdots & & & \\ I_{2^M-3} & = & Y_1 & & \cap Y_3 & \cdots & Y_M, \\ I_{2^M-2} & = & & Y_2 & \cap Y_3 & \cdots & Y_M, \\ I_{2^M-1} & = & Y_1 & \cap Y_2 & & \cdots & Y_M, \end{array}$$

Table 3.3: All 2^M intersections of M subsets.

Let $Y_{m_1} \cap Y_{m_2} \cap \cdots \cap Y_{m_P}$ be the intersection of P base subsets $Y_{m_1}, Y_{m_2}, \cdots, Y_{m_P}$ of Y_x where $m_1, \cdots, m_P \in \{1, 2, \cdots, n\}$ are their indices respectively. We choose subscript k for this intersection, such that

$$k = \sum_{q=1}^{P} 2^{m_q - 1},$$

and denote $Y_{m_1} \cap Y_{m_2} \cap \cdots \cap Y_{m_P}$ by I_k for simplicity. From the subscript k we can find which subsets construct the intersection I.

Table 3.4 illustrates the bijection mapping between the intersection I_k and the binary number k.

base sets	Y_1	Y_2	Y_3	$\cdots$	Y_{M-2}	Y_{M-1}	Y_M
I_k	1	0	1	$\cdots$	0	1	1

Table 3.4: The mapping between intersections and binary numbers. I_k is the intersection of subsets with value 1 under them. k is a binary with value 1 in corresponding bits.

For instance, when $M = 3$,

$$\begin{aligned}
I_0 &= Y_x \\
I_1 &= Y_1 \\
I_2 &= Y_2 \\
I_3 &= Y_1 \cap Y_2 \\
I_4 &= Y_3 \\
I_5 &= Y_1 \cap Y_3 \\
I_6 &= Y_2 \cap Y_3 \\
I_7 &= Y_1 \cap Y_2 \cap Y_3
\end{aligned}$$

Now we exam the value of $\alpha_{u_j,w}^{g(u_j,w)} - 1$ (for $j = 1, \cdots, M$) for words $w \in Y_1$, $\cdots$, Y_M and find it is non-zero only for $w \in Y_j$. In more general cases, the product $\prod_{q=1}^{P}(\alpha_{u_{m_q},w}^{g(u_{m_q},w)} - 1)$ is non-zero only for word $w \in I_k$ where $k = \sum_{q=1}^{P} 2^{m_q-1}$.

Therefore,

$$\sum_{w\in V}\prod_{j=1}^{P}(\alpha_{u_{m_j},w}^{g(u_{m_j},w)}-1)=\sum_{w\in I_k}\prod_{j=1}^{P}(\alpha_{u_{m_j},w}^{g(u_{m_j},w)}-1).$$

This property is very useful in deriving the algorithms for computing the normalization factors z in the model with M bigram constraints.

We need define another shorthand notation

$$A_k \doteq (\alpha_{u_{m_1},w}^{g(u_{m_1},w)}-1)\cdots(\alpha_{u_{m_P},w}^{g(u_{m_P},w)}-1)\alpha_w^{g(w)} \tag{3.29}$$

where k is the index for the intersection of P subsets Y_{m_1}, Y_{m_1}, $\cdots$, Y_{m_P}; for instance,

$$A_0=\alpha_w,\text{ the unigram parameters and}$$

$$A_{2^M-1}=\alpha_w\prod_{k=1}^{M}(\alpha_{u_k,w}-1).$$

We can write the property about I_k using the notation A_k as

$$\sum_{w\in V}A_k=\sum_{w\in I_k}A_k. \tag{3.30}$$

There is also an important property for A_k's. We can rewrite $\alpha_w^{g(w)}\prod_{i=1}^{M}\alpha_{u_i,w}^{g(u_i,w)}$ by

$$\begin{aligned}&\alpha_w^{g(w)}\prod_{i=1}^{M}\alpha_{u_i,w}^{g(u_i,w)} \\ =\ &\alpha_w^{g(w)}\prod_{i=1}^{M}[(\alpha_{u_i,w}^{g(u_i,w)}-1)+1] \\ =\ &\sum_{k=0}^{2^M-1}A_k.\end{aligned} \tag{3.31}$$

We now can derive the following lemma, which is an extension of Equations (3.26)-(3.27), from the two properties above.

Lemma 2: In the model (3.28), the normalization factor can be computed as

$$z(x)=\sum_{k=0}^{2^M-1}\sum_{w\in I_k}A_k. \tag{3.32}$$

proof:

Using the property (3.31), $z(x)$ can be rewritten as

$$\begin{aligned} z(x) &= \sum_{w \in V} \alpha_w^{g(w)} \cdot \prod_{i=1}^{M} \alpha_{u_i,w}^{g(u_i,w)} \\ &= \sum_{w \in V} [\sum_{k=0}^{2^M-1} A_k]. \end{aligned}$$

Using the property (3.30), it can be rewritten as

$$z(x) = \sum_{k=0}^{2^M-1} \sum_{w \in I_k} A_k.$$

∎

3.4.3 Analysis of Complexity

First we analyze the complexity when $M = 3$:

$$\begin{aligned} z(u_1, u_2, u_3) &= \sum_{w \in Y_x} A_0 + \sum_{w \in Y_1} A_1 + \sum_{w \in Y_2} A_2 + \sum_{w \in Y_3} A_4 \\ &+ \sum_{w \in Y_1 \cap Y_2} A_3 + \sum_{w \in Y_1 \cap Y_3} A_5 + \sum_{w \in Y_2 \cap Y_3} A_6 \\ &+ \sum_{w \in Y_1 \cap Y_2 \cap Y_3} A_7. \end{aligned}$$

The first four summations can be done in $O(U)$, $O(B_1)$, $O(B_2)$ and $O(B_3)$ time, respectively, where B_1, B_2 and B_3 are numbers of bigram constraints of the kind $\langle u_1, w \rangle$, $\langle u_2, w \rangle$ and $\langle u_3, w \rangle$, respectively. The three following sums require the time of $O(\sum_{u_1,u_2 \in \hat{X}} |Y_1 \cap Y_2|)$, $O(\sum_{u_1,u_3 \in \hat{X}} |Y_1 \cap Y_3|)$ and $O(\sum_{u_2,u_3 \in \hat{X}} |Y_2 \cap Y_3|)$, respectively as analyzed in Section 3.3.1. The last one needs $O(\sum_{u_1,u_2,u_3 \in \hat{X}} |Y_1 \cap Y_2 \cap Y_3|)$ time. $O(U + B_1 + B_2 + B_3)$ is small comparative to the rest and thus may be ignored. This implementation, therefore, is more efficient than unigram-caching when

$$\begin{aligned} \sum_{u_1,u_2 \in \hat{X}} |Y_1 \cap Y_2| + \sum_{u_1,u_3 \in \hat{X}} |Y_1 \cap Y_3| + \sum_{u_2,u_3 \in \hat{X}} |Y_2 \cap Y_3| + \sum_{u_1,u_2,u_3 \in \hat{X}} |Y_1 \cap Y_2 \cap Y_3| \\ << \sum_{u_1,u_2,u_3 \in \hat{X}} |Y_1 \cup Y_2 \cup Y_3|. \end{aligned}$$

Obviously, each sum on the left-hand side is strictly less than what on the right-hand side. However, it is not guaranteed that the four sums together on the left-hand side are still significantly less than what on the right-hand side if the intersections $Y_1 \cap Y_2$, $Y_1 \cap Y_3$, $Y_2 \cap Y_3$ and $Y_1 \cap Y_2 \cap Y_3$ are large. Therefore, this implementation is useful only when the extent of overlap between features is small. In practice, it will reduce the computation by about 50% in training syntactic models.

Now we analyze the training time for the model with M arbitrary kinds of bigrams. $\sum_{w \in I_{2^k-1}} A_{2^k-1}$ for $k = 1, \cdots, M$ is fixed for all histories sharing the same symbols u_k at positions k, and it needs to be computed only once and may then be reused. One can easily check that this sum needs only $O(B_k)$ time, where B_k is the number of "bigrams" $\langle u_k, w_i \rangle$ seen in the training data.

It is difficult to precisely enumerate the complexity of other terms, which complexity depends on two factors, the number of different history subsequence $u_{m_1}, \cdots, u_{m_P}$ denoted as H_k, $k = \sum_{j=1}^{P} 2^{m_j}$, and the average size of I_k. Usually, the larger the size of I_k, the smaller is H_k, and vice versa. If, fortunately, all features are almost non-overlapping features, i.e., if $\cap_k I_k \approx \phi$, then the computation load is quite low. On the other hand, if some features are heavily overlapping, this implementation may not save computational time. In the worst case, this implementation could be even more inefficient than the direct implementation of (3.29).

Next we provide an upper bound of the running time of this non-overlapping sharing method and give a sufficient condition for using this method instead of unigram-caching in training maximum entropy models. Let

$$T_k = \# \text{ of terms in } \sum_{w \in I_k} A_k \text{ for all seen histories.}$$

E.g., $T_0 = U$, $T_1 = B_1$, etc. Let

$$T_{max} = \max_k T_k. \tag{3.33}$$

The overall complexity is then less than $O(2^M \cdot T_{max})$.

If this number $2^M \cdot T_{max}$ is less than the number of terms in unigram-caching, then non-overlapping sharing is worth applying. T_{max} is usually close to T_{2^M-1}. Care must be taken regarding two factors when using this method. First, the size of M should be small because 2^M increase exponentially in M. In practice, M should be less than four in language modeling. Second, redundant features should be avoided so that the intersections of two or more Y_k subsets are kept small in size. We will show in Section 5.5 that dropping some redundant syntactic features will not degrade the precision of the syntactic model.

Concerning Extra Space Requirement

Summations over I_k for $k = 1, \cdots, 2^M - 1$ must be pre-computed and stored in order to compute the summation over Y_x. Therefore, the extra space of $O(\sum_{k=1}^{2^M-1} I_k)$ is required.

More Discussions

This approach can be extended easily to train a model with only M *"order n"* conditional features. As we will show in subsequent sections, it can also be used with other simplifications in training methods.

In general, the training of the models with overlapping features is less efficient than that of the N-gram models because, as stated before, of the need to consider many tuples that do not appear in the training data; specifically,

1. the right hand side of the Equation (3.32) has 2^M summations instead of N as in (3.20), and
2. the computational load for each summation $\sum_{w \in I_k} A_k$ over the overlapping parts may be enormous if feature patterns are heavily overlapping.

It is worth mentioning here that cluster expansion in Lafferty & Suhm (1996) works similarly and has the same complexity as this method for models with one kind of unigram features and M kinds of bigram features. The difference is that we only use non-overlapping sharing for the features in the same hierarchy, while they applied the idea to all features. Therefore, non-overlapping sharing can be regarded as a special case of cluster expansion.

3.5 Generalized Hierarchical Training Methods

Section 3.2 shows the best cases of maximum entropy models whose features are all nested. Their training time of $O(L)$ has already reached the lower bound of the empirical estimation. Section 3.4 shows the worst cases in which none of conditional features are nested. Most maximum entropy models lie in the middle between these two extreme cases: they have both hierarchically nested features and non-nested features. In this section, we will combine the ideas earlier shown for these two cases, and achieve a more general approach for ME models that we call generalized hierarchical training (GHT).

Most of maximum entropy models are designed to combine several sources of information together. It is natural to assume that these information sources are mutually non-redundant, even though the maximum entropy method does not require this assumption.

Let M be the number of information sources from which the constraints of the model are selected, and let N be the highest order of constraints. We first derive the generalized hierarchical training methods based upon the following two assumptions:

i. *independence assumption:* all information sources are non-overlapping. (The subsets of the "history" x representing each information source from a disjoint partition of x.), and
ii. *hierarchy assumption:* all feature patterns within an information source are nested.

Later, we extend GMH to models where these two assumptions need not hold.

3.5.1 Training MN-gram Models Hierarchically

MN-gram Model

Let M be the number of information sources, and N be the highest order of constraints. We call the maximum entropy model an MN-gram model if it satisfies the independence and hierarchy assumptions. For example, a regular N-gram model can be regarded as a 1N-gram model, and the model in (3.28) is an M2-gram model.

We need to introduce some notation before we discuss the training of MN-gram models. Let

$$u_{1,1}^{M,N-1} = x = \langle u_{1,1}, u_{1,2}, \cdots, u_{1,N-1}, u_{2,1}, \cdots, u_{2,N-1}, \cdots, u_{M,1}, \cdots, u_{M,N-1} \rangle$$

be a history equivalence class. If the order of constraints from some information source i is less than $N-1$, we simply set the corresponding high order $u_{i,j+1}, \cdots, u_{i,N-1}$ to *null*.

Let $g(w)$ be the unigram feature function, $g(u_{i,1}, w)$, $\cdots$, $g(u_{i,1}, \cdots, u_{i,N-1}, w)$ for $i = 1, 2, \cdots, M$ be bigram, $\cdots$, N-gram features from information source i.

The maximum entropy MN-gram model can be written as

$$p(w|u_{1,1}^{M,N-1}) = \frac{\alpha_w^{g(w)} \prod_{i=1}^{M} \prod_{j=2}^{N} \alpha_{u_{i,1}^{i,j-1},w}^{g(u_{i,1}^{i,j-1},w)}}{z(u_{1,1}^{M,N-1})}, \tag{3.34}$$

where

$$\begin{aligned} u_{i,1}^{i,j-1} &= u_{i,1}, \cdots, u_{i,j-1}, \text{ and} \\ z(u_{1,1}^{M,N-1}) &= \sum_{w \in V} \alpha_w^{g(w)} \prod_{i=1}^{M} \prod_{j=2}^{N} \alpha_{u_{i,1}^{i,j-1},w}^{g(u_{i,1}^{i,j-1},w)}. \end{aligned} \tag{3.35}$$

Note that we use subscript i for the information source, and j for the order. The left digraph in Figure 3.5.1 shows the feature hierarchy of MN-models. Now we hierarchize this MN-gram model to a model that looks like N-gram models.

Hierarchizing

Let $U_j = (u_{1,j}, \cdots u_{M,j})$ for $j = 1, 2, \cdots, N-1$[5]. We define a *super feature* f as

$$f(U_1, \cdots, U_{j-1}, w) = \begin{cases} 1 & \text{if } g(u_{i,1}, \cdots, u_{i,j-1}, w) = 1 \text{ for some } i, \\ 0 & \text{otherwise} \end{cases}$$

[5]If the highest order of constraints from some information source i is less than j, we simply let the feature corresponding $u_{i,j}$ be empty.

and its parameter:

$$\beta_{U_1,\cdots,U_{j-1},w} = \prod_{i=1}^{M} \alpha_{u_{i,1},\cdots,u_{i,j-1},w}^{g(u_{i,1},\cdots,u_{i,j-1},w)}.$$

We call this procedure *hierarchizing.* For example, in the model (3.23), the bigram feature $g(w_{i-1}, w_i)$ and the topic feature $g(t_i, w_i)$ are compounded to generate the super bigram features

$$f(w_{i-1}, t_i, w_i) = \begin{cases} 0 & \text{if } g(w_{i-1}, w_i) = 1 \text{ or } g(t_i, w_i) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

After hierarchizing, *all super-features f_i are now nested features.* Model (3.34) becomes

$$p(w|U_1, \cdots, U_{N-1}) = \frac{\beta_w^{f(w)} \prod_{j=2}^{N} \beta_{U_1^{j-1},w}^{f(U_1^{j-1},w)}}{z(U_1, \cdots, U_{N-1})} \tag{3.36}$$

where

$$\begin{aligned} U_1^{j-1} &= U_1, U_2, \cdots, U_{j-1}, \\ z(U_1, \cdots, U_{N-1}) &= \sum_{w \in V} \beta_w^{f(w)} \prod_{j=2}^{N} \beta_{U_1^{j-1},w}^{f(U_1^{j-1},w)}. \end{aligned}$$

Figure 3.5.1 shows the feature hierarchies of MN-gram model before and after this transform. Now we show the hierarchical training method for model (3.36). We define base word subsets

$$Y_{i,j} = \{w : g(u_{i,1}, \cdots, u_{i,j-1}, w) = 1\},$$

and

$$Y_j = \cup_{i=1}^{M} Y_{i,j}.$$

We also define

$$Y_j' = \cup_{k=j}^{N} Y_k.$$

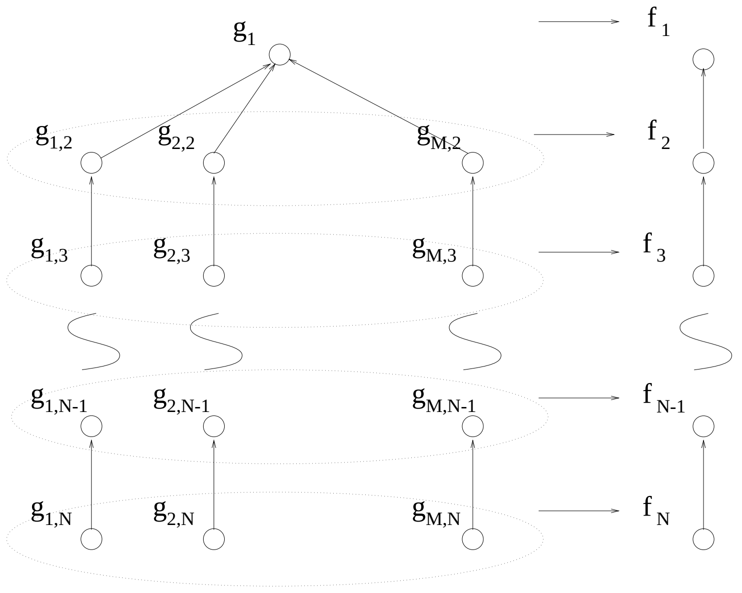

Figure 3.10: Converting the MN-gram model to the n-gram model with super features. $g_{[1]} = g(w)$ and $g_{[j,i]} = g(u_{i,1}, \cdots, u_{i,j-1})$ for simplicity.

We extend the recursive equation (3.16) in Lemma 1 as

$$\begin{aligned} &\sum_{w \in Y'_j} [\prod_{n=j+1}^{N} \beta_{U_1^{n-1},w}^{f(U_1^{n-1},w)} - 1] \cdot \prod_{n=1}^{j} \beta_{U_1^{n-1},w}^{f(U_1^{n-1},w)} \qquad (3.37) \\ = &\sum_{w \in Y_j} (\beta_{U_1^j,w}^{f(U_1^j,w)} - 1) \cdot \prod_{n=1}^{j} \beta_{U_1^{n-1},w}^{f(U_1^{n-1},w)} + \sum_{w \in Y'_{j+1}} \left[\prod_{n=j+2}^{N-1} \beta_{U_1^n}^{f(U_1^n,w)} - 1 \right] \cdot \prod_{n=1}^{j+1} \beta_{U_1^{n-1}}^{f(U_1^{n-1},w)}. \end{aligned}$$

Only the first summation on this right hand side in (3.37) need be calculated directly; the second one will be evolved recursively. If M is small, We can use the trick presented in Section 3.4 to compute the first summation.

We call this method *generalized hierarchical training.*

Analysis of Complexity

The computation of z is split into N summations $\sum_{w \in Y_j}$ for $j = 1, \cdots, N$. We need to mention here that the time needed for each summation $\sum_{w \in Y'_j} [\prod_{n=j+1}^{N} \beta_{U_1^{n-1},w}^{f_n(U_1^{n-1},w)} - 1] \cdot \prod_{n=1}^{j} \beta_{U_1^n,w}^{f_n(U_1^n,w)}$ is no longer bounded by the size of the training data, and it is roughly $O(2^M \cdot T_{max})$ where T_{max} is defined in (3.33) in Section 3.4.3. However, in practice, this implementation of GHT is still much more efficient than the baseline. In the baseline method, the time complexity can be estimated as $O(|\hat{X}| \cdot |\bar{Y}_x|)$ where $|\hat{X}|$ is the number of $u_{1,1}, \cdots, u_{M,N-1}$ seen in the training data and $|\bar{Y}_x| > |\bar{Y}_2|$. In the generalized hierarchical method, the complexity can be roughly estimated as $O(\sum_{j=2}^{N} |\hat{X}_j| \cdot |\bar{Y}_j|)$ where $|\hat{X}_j|$ is the number of history class $u_{1,1}, \cdots, u_{M,j-1}$ seen in the training data. Since $|\bar{Y}_N| < \cdots |\bar{Y}_j| \cdots < |\bar{Y}_3| << |\bar{Y}_2|$ and $|\hat{X}_2| << |\hat{X}|$,

$$\sum_{j=2}^{N} |\hat{X}_j| \cdot |\bar{Y}_j| < |\hat{X}_2| \cdot |\bar{Y}_2| + (N-2)|\hat{X}| \cdot |\bar{Y}_3| << |\hat{X}| \cdot |\bar{Y}_x|. \tag{3.38}$$

We can roughly estimate the running time by $O(|\hat{X}_2| \cdot |\bar{Y}_2| + (N-2)|\hat{X}| \cdot |\bar{Y}_3|)$. The summation $\sum_{w \in Y_j}$ in (3.37) can be computed directly or aggregated from the 2^M partial summations over intersections of $Y_{i,j}$'s using the non-overlapping sharing described in the preceding section, depending on which is more efficient. If distributed to 2^M summations, the complexity is strictly less than $O(N \cdot 2^M \cdot T_{max})$ where M here is the maximum number of overlapping features in any super-feature and T_{max} is as defined in Section 3.4.

3.5.2 Training a General ME Model Hierarchically

Now we show that the generalized hierarchical training method still applies without the above independence and nested assumptions.

We need to define some terms and notations that will be used later.

Definition (partial order set POSET):

A partial order "<" is an order defined for some, but not necessarily all, pairs of elements on a set $\mathcal{S}$.

Partial order set (POSET) $(\mathcal{S}, <)$ is a set of elements that are subject to a partial order $<$.

An element e_m in a POSET $(\mathcal{S}, <)$ is *a minimal element* if there is no element e such that $e < e_m$. It is also a *minimum element* if $e_m < e$ for any element e in the POSET. An element e_M in a POSET $(\mathcal{S}, <)$ is *a maximal element* if there is no element e

such that $e_M < e$. It is also a *maximum element* if $e < e_M$ for any element e in the POSET.

Let $x = u_1, u_2, \cdots, u_k$ be a history class determined by k symbols, and $s = u_{i_1}, u_{i_2}, \cdots, u_{i_{k'}}$ be a *subsequence* of x determined by k' symbols. All subsequences s of x create a *partial order set* (POSET) $\mathcal{S}$ of 2^k elements with the ordering relation $s < s'$ if s is a subsequence of s'. Any feature types in an ME model are a function of a subsequence s and a predicted symbol y.

Let G be the set of features. Obviously, the nested relation is a partial order relation on G. Therefore, all feature types form a partial order set with the *minimum element (or feature)* $g(y)$. There are some *maximal elements (or features)* in this POSET, but the *maximum element* may not exist[6]. The following (*maximal first*) algorithm provides a recursive way of hierarchizing ME models.

Algorithm (Hierarchizing)

Step 1: Create super-features $f(s', y)$ from all common maximal features $g(s_1, y), g(s_2, y), \cdots, g(s_j, y)$ where s' is the *shortest super-sequence*[7] for all $s_1, s_2, \cdots, s_j$.

$$f(s', y) = 1 \text{ iff some } f(s_i, y) = 1 \text{ for } i = 1, 2, \cdots, j.$$

Note that $f(s', y)$ is a super feature of the highest order.

Step 2: Construct the remaining feature set G' by removing all maximal features $g(s_1, y)$, $g(s_2, y), \cdots, g(s_j, y)$ from G *i.e.,*

$$G' = G - \{g(s_1, y), g(s_2, y), \cdots, g(s_j, y)\}.$$

Step 3: Set $G = G'$.
Repeat Step 1 and 2 recursively and create super-features $f(s'', y), \cdots, f(s^n, y)$.

∎

See Figure 3.11 for an example of hierarchizing.

It can be easily checked that all these super-features $f(s', y), f(s'', y), \cdots, f(s^n, y)$ are nested features. We need only to show that $f(s', y)$ is nested in $f(s'', y)$. If

[6]Maximum element exists if and only if there is a feature $g(s^*, y)$ such that $s < s^*$ for all $s \in \mathcal{S}$. Actually it is not important whether the maximum element exist here.

[7]s' is a super-sequence of s if and only if s is a subsequence of s'.

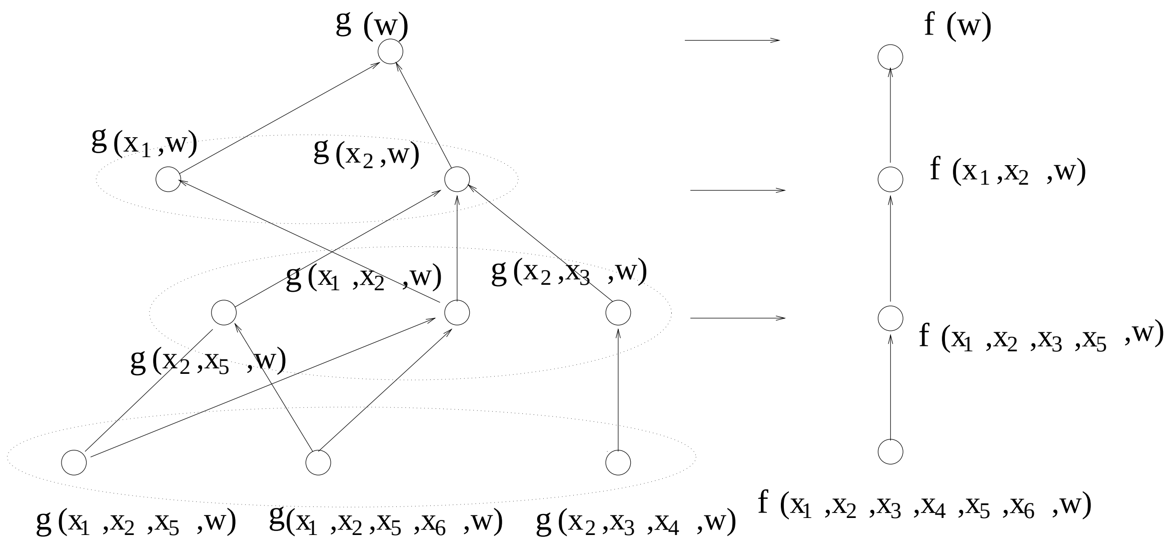

Figure 3.11: Hierarchizing: non-nested to nested

$f(s', y) = 1$, there must exist a $g(s_j, y) = 1$ where $g(s_j, y)$ is a maximal element. $g(s_j, y)$ must be nested in a maximal element $g(s_k, y) \in G'$ according to the feature hierarchical structure, and $g(s_k, y)$ is contained in $f(s'', y)$ according to the hierarchizing procedure above. $g(s_j, y) = 1 \Rightarrow g(s_k, y) = 1 \Rightarrow f(s'', y) = 1$. Therefore, $f(s', y)$ is nested in $f(s'', y)$. Of course, there are other ways of creating nested super-features from the original feature set. The above algorithm is only one example of hierarchizing. After hierarchizing, we can use the hierarchical training method in Section 3.2 together to train such an ME model.

The complexity analysis is similar to that of the MN-gram model. Let M be the maximum number of features a super feature can contain and N be the highest of order of a feature. We can use the non-overlapping sharing described in Section 3.4 in addition to the generalized hierarchical training method to deal with the overlapping features inside a super-feature, or use the generalized hierarchical training individually if the number of interacting features is too large. The complexity of the former is roughly $O(|\hat{X}_2|\cdot|\bar{Y}_2|+(N-2)|\hat{X}|\cdot|\bar{Y}_3|)$ and that of the latter is roughly $O(N\cdot 2^M\cdot T_{max})$. Overall, we can roughly estimate the complexity as $\min(O(|\hat{X}_2|\cdot|\bar{Y}_2| + (N-2)|\hat{X}|\cdot|\bar{Y}_3|), O(N\cdot 2^M\cdot T_{max})$ using the notation for MN-gram models. If $M\cdot N$ is a constant, the model with a large N can be trained more efficiently than the one with a large

M because the former has better feature hierarchy.

It is again worth mentioning that the cluster expansion method of Lafferty & Suhm (1996) does not take advantage of hierarchical structure among features, and treats the features of all orders as if on a flat level. Therefore, it results in less efficiency compared to GHT. The complexity for training an MN-gram model using cluster expansion is $O(2^N \cdot 2^M \cdot T'_{max})$ instead of $O(N \cdot 2^M \cdot T_{max})$, and it is easy to check that $T'_{max} > T_{max}$.

3.6 Divide-and-Conquer

The generalized hierarchical training method is designed for general purposes, but may not be the optimal method for some models. For instance, to train the following topic model

$$p(w_i|w_{i-2}, w_{i-1}, t_i) = \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)}}{z(w_{i-2}, w_{i-1}, t_i)}, \tag{3.39}$$

using (3.37), we compound $g(w_{i-1}, w_i)$ and $g(t_i, w_i)$ to generate a super-feature $f(w_{i-1}, t_i, w_i)$. We also need to define $f(w_{i-2}, w_{i-1}, t_i, w_i) = g(w_{i-1}, w_{i-1}, w_i)$ so that super trigram features can be nested within super bigram features. The computational load corresponding to $f(w_{i-1}, t_i, w_i)$ depends on the number of words having a bigram feature $g(w_{i-1}, w_i)$ activated and a topic feature $g(t_i, w_i)$ activated, i.e., the size of $Y_w \cap Y_t$. Even though $Y_w \cap Y_t < Y_w \cup Y_t$, and thus the generalized hierarchical training method, is more efficient than the baseline approach, it is still far more costly than the optimal one we can reach for the topic model.

Therefore, we will introduce another efficient training method for topic-dependent models. One may observe that within a given topic t_i the value of the topic feature $g(t_i, w_i)$ is independent of w_{i-2}, w_{i-1} and can be treated as a marginal feature. We can take advantage of this and use a divide-and-conquer approach [8] to train the topic-dependent model at a very low cost. This approach involves five major steps:

[8]Divide-and-conquer here is slightly different from the concept in computer algorithms which means dividing the computation recursively. Here the computation is only divided once.

Step 1: Partition the training text into topics $D_1, D_2, \cdots, D_K$ where K here is the number of topics.

Step 2: For the training data of topic t denoted as D_t, collect N-grams.

Step 3: Compute all normalization factors z in same topic t.

Define

$$f_t(w_i) = g(w_i) \text{ for } t = 1, \cdots, K$$

and

$$\alpha'_{w_i} = \begin{cases} \alpha_{w_i} \cdot \alpha_{t_i, w_i} & \text{if } g(t_i, w_i) = 1, \\ \alpha_{w_i} & \text{otherwise.} \end{cases} \tag{3.40}$$

then

$$z(w_{i-2}, w_{i-1}, t_i) = \sum_{w \in V} {\alpha'_{w_i}}^{f_t(w_i)} \cdot \alpha_{w_{i-1}, w_i}^{g(w_{i-1}, w_i)} \cdot \alpha_{w_{i-2}, w_{i-1}, w_i}^{g(w_{i-2}, w_{i-1}, w_i)}.$$

Step 4: Now $f_t(w_i)$, $g(w_{i-1}, w_i)$, and $g(w_{i-2}, w_{i-1}, w_i)$ are nested features. Use the formula (3.20) to compute all $z(w_{i-2}, w_{i-1}, t_i)$ for fixed t.

$$\begin{aligned} z(w_{i-2}, w_{i-1}, t_i) &= \sum_{w_i \in V} {\alpha'_{w_i}}^{f_t(w_i)} + \sum_{w_i \in Y_2} (\alpha_{w_{i-1}, w_i}^{g(w_{i-1}, w_i)} - 1) \cdot {\alpha'_{w_i}}^{f_t(w_i)} \\ &+ \sum_{w_i \in Y_3} (\alpha_{w_{i-1}, w_i}^{g(w_{i-1}, w_i)} \cdot \alpha_{w_{i-2}, w_{i-1}, w_i}^{g(w_{i-2}, w_{i-1}, w_i)} - 1) {\alpha'_{w_i}}^{f_t(w_i)}. \end{aligned}$$

Step 5: Collect partial feature expectations by topics. Details of this step will be given shortly in Section 3.7.

Analysis of Performance

The number of multiplication in (3.40) in Step 3 is exactly the number of topic features denoted as U_{topic}[9]. The running time in either Step 4 or Step 5 is roughly $O(U + B^*_k + T^*_k)$ for each topic k, where B^*_k is the number of the combinations of words w_{i-1} in topic k and words w_i following w_{i-1} in the training data, and T^*_k is the number of combinations of bigram w_{i-2}, w_{i-1} in topic k and their following words in the

[9] Assuming that $U_{topic} > U$.

training data. It should be noted that B_k^* here and B_k used in non-overlapping sharing and generalized hierarchical training are slightly different, because the subscript k in the former represents the order, whereas it represents the information source in the latter. B_k^* and T_k^* are usually in the same order of the numbers of bigrams and trigrams in topic k respectively. The total time is thus less than $O(U \cdot K + \sum_{t=1}^{K} B_k^* + T_k^*)$ [10] for all topics. This time is about the same as obtaining K individual different empirical estimates for K topics.

This method can also be applied to other models similar to the topic-dependent model (3.39) that have N-gram constraints and one other kind of constraint not nested in N-grams, e.g., the model with part-of-speech tag constraints:

$$P(w_i|w_{i-2}, w_{i-1}, pos_{i-1}) = \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{pos_{i-1},w_i}^{g(pos_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, pos_{i-1})},$$

where pos_{i-1} is the part-of-speech tag of the previous word w_{i-1}.

To train this model, we need only to partition the training data according to pos_{i-1} instead of t_i and then follow the steps of training the topic-dependent model. It should be noted that this method cannot be used when the number of partitions is too large, since the overhead time in Step 1, Step 2 and Step 3 would then be considerable.

It is worth mentioning that divide-and-conquer (DC) can be regarded as a variation of GHT. In the generalized hierarchical training, the regular bigram feature and the topic feature are combined into a *bigram super-feature*, while in divide-and-conquer the unigram feature and the topic feature are combined to create a *unigram super-feature*. This slight difference in organizing the computation results in different efficiencies for topic models between divide-and-conquer and the generalized hierarchical training. Figure 3.12 illustrates the differences between these two approaches in hierarchizing the topic model.

In Section 3.8, we will show that both divide-and-conquer and generalized hierarchical training can also be used together to train language models with both topic

[10]Actually no more than $O(U + U_{topic} + \sum_{t=1}^{K} B_k^* + T_k^*)$ because α' needs to be computed only for words with topic constraints in Step 3.

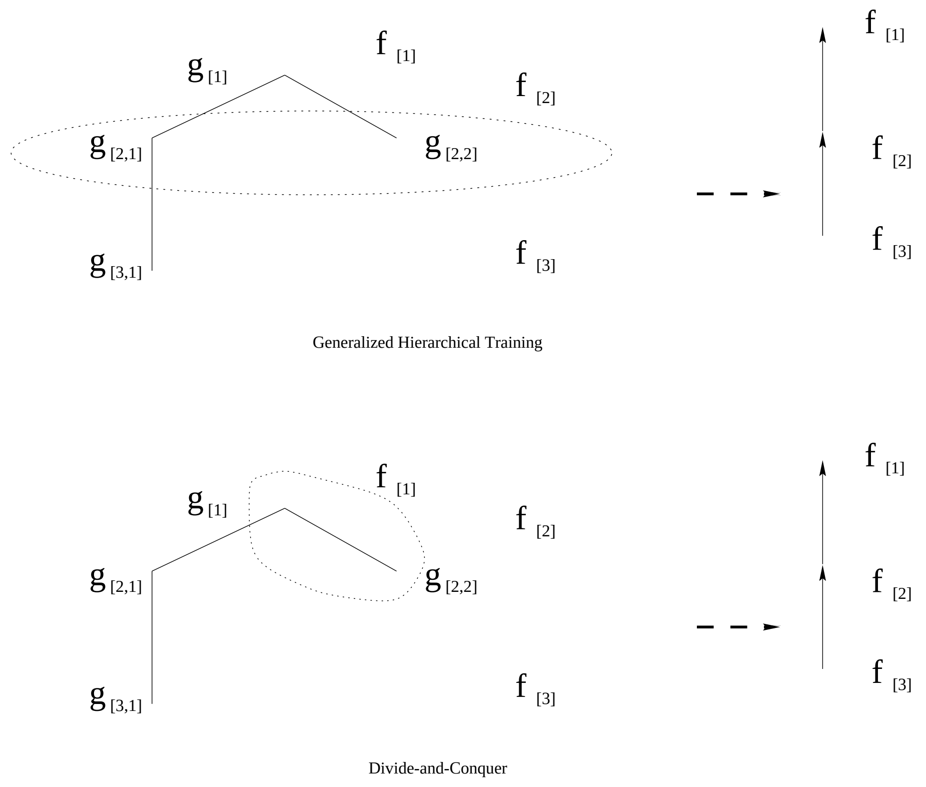

Figure 3.12: Generalized hierarchical training vs. divide-and-conquer

and syntactic constraints.

3.7 Feature Expectation

In previous sections, we focused exclusively on the computation of the normalization factor z in previous sections. In this section, we briefly describe the computation of feature expectations.

Any simplification in computing z in the previous sections can be applied to the computation of feature expectations. We will show, by the example of the topic model (3.39), how to compute the expectation of the N-gram features $g(w_i)$, $g(w_{i-1}, w_i)$, $g(w_{i-2}, w_{i-1}, w_i)$ and the topic feature $g(t_i, w_i)$. The computation of other feature expectations is similar. We will use the computational tricks of hierarchical training and divide-and-conquer in calculating the feature expectation.

The expectation $p[g(w_{i-2}, w_{i-1}, w_i)]$ of trigram features $g(w_{i-2}, w_{i-1}, w_i)$ is relatively easy to compute according to

$$p[g(w_{i-2}, w_{i-1}, w_i)] \quad = \quad \sum_{t_i} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \alpha_{w_i}^{g(w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)}.$$

It is easy to check that the time complexity for trigrams is only $O(\#[w_{i-2}, w_{i-1}, w_i, t_i])$.

Computing the expectation of bigram features $g(w_{i-1}, w_i)$ needs to use the idea of hierarchical training. It should be noted that w_{i-1} and w_i here can be regarded as fixed coefficients instead of variables.

$$\begin{aligned}
& p[g(w_{i-1}, w_i)] \\
&= \sum_{t_i, w_{i-2}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \alpha_{w_i}^{g(w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot g(w_{i-1}, w_i) \\
&= \sum_{t_i} \left[\sum_{w_{i-2}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \right] \alpha_{w_i}^{g(w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot g(w_{i-1}, w_i) \qquad (3.41) \\
&+ \sum_{t_i, w_{i-2}} \left[\frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \alpha_{w_i}^{g(w_i)} \alpha_{t_i,w_i}^{g(t_i,w_i)} \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} (\alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1) \right] \qquad (3.42)
\end{aligned}$$

$\left[\sum_{w_{i-2}} \frac{\hat{p}(w_{i-2},w_{i-1},t)}{z(w_{i-2},w_{i-1},t)}\right]$ in (3.41) can be pre-computed for all bigram features in $O(\#[w_{i-2}, w_{i-1}, t_i])$. The remaining of computation in (3.41) takes roughly $O(\#[w_{i-1}, w_i, t_i])$ time for all bigrams. For all bigrams w_{i-1}, w_i, the computation in (3.42) takes $O(\sum_{k=1}^{K} T_k^*)$ where K is the number of topics and T_k^*, as defined in the previous section, is the number of combinations of bigram w_{i-1}, w_i in topic k and their preceding word w_{i-2} in the training data. The overall complexity is the same as computing z for all histories.

Computing the expectation of unigram features $g(w_i)$ may take very long time if improperly implemented. We use a similar computational trick to that we have used in computing the normalization factor z.

$$\begin{aligned} & p[g(w_i)] \\ = & \sum_t \sum_{w_{i-2},w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t)} \alpha_{w_i}^{(w)} \cdot \alpha_{t_i,w_i}^{(t_i,w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} && (3.43) \\ = & \sum_t \left[\sum_{w_{i-2},w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)} \right] \alpha_{w_i}^{g(w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} && (3.44) \\ + & \sum_t \sum_{w_{i-2}} \left[\sum_{w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)} \right] \alpha_{w_i}^{g(w_i)} \alpha_{t_i,w_i}^{g(t_i,w_i)} (\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} - 1) && (3.45) \\ + & \sum_{t,w_{i-2},w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)} \alpha_{w_i}^{g(w_i)} \alpha_{t_i,w_i}^{g(t_i,w_i)} \alpha_{w_{i-1},w}^{g(w_{i-1},w_i)} (\alpha_{w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} - 1) && (3.46) \end{aligned}$$

Formula (3.43) shows the straight-forward implementation, and (3.44)-(3.46) illustrate the improved version. In particular, Formula (3.44) takes advantage of unigram-caching, while (3.45) and (3.46) take advantage of hierarchical training. It should be noted that

- $\sum_{w_{i-2},w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)}$ in (3.44) is fixed given $t_i = k$, so it can be computed once in $O(|X_k|)$ and then reused for unigram features $g(w_i)$ within the same $t_i = k$, where $|X_k|$ is the number of histories in topic k;

- $\sum_{w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)}$ in (3.45) is a byproduct of $\sum_{w_{i-2},w_{i-1}} \frac{\hat{p}(w_{i-2},w_{i-1},t_i)}{z(w_{i-2},w_{i-1},t_i)}$ and so needs no extra computation, and the number of terms to be summed for all $g(w_i)$ is $\sum_{k=1}^{K} B_k$ where B_k, as defined in the previous section, is the number of the combinations of words w_i in topic k and words w_{i-1} preceding it in the training data; and

- in (3.46), the number of terms to be summed for all $g(w_i)$ is $\sum_{k=1}^{K} T_k^*$.

Therefore, the overall time complexity is roughly $O(\sum_{k=1}^{K} B_k^* + \sum_{k=1}^{K} T_k^*)$, which, as shown in Section 3.6, is the same as computing the normalization factor z for all histories in the training data.

The expectation of the topic feature $g(t_i, w_i)$ is computed similarly to that of unigrams, but t_i here is a fixed coefficient instead of a variable.

$$\begin{aligned}
& p[g(t_i, w_i)] \\
= & \sum_{w_{i-2}, w_{i-1}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \alpha_{w_i}^{g(w_i)} \alpha_{w_{i-1}, w}^{g(w_{i-1}, w_i)} \alpha_{w_{i-1}, w_i}^{g(w_{i-2}, w_{i-1}, w_i)} \alpha_{t_i, w_i}^{g(t_i, w_i)} \\
= & \left[\sum_{w_{i-2}, w_{i-1}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \right] \alpha_{w_i}^{g(w_i)} \alpha_{t_i, w_i}^{g(t_i, w_i)} \qquad (3.47) \\
+ & \sum_{w_{i-1}} \left[\sum_{w_{i-1}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t)} \right] \alpha_{w_i}^{g(w_i)} \alpha_{t_i, w_i}^{g(t_i, w_i)} (\alpha_{w_{i-1}, w_i}^{g(w_{i-1}, w_i)} - 1) \qquad (3.48) \\
+ & \sum_{w_{i-2}, w_{i-1}} \frac{\hat{p}(w_{i-2}, w_{i-1}, t_i)}{z(w_{i-2}, w_{i-1}, t_i)} \alpha_{w_i}^{g(w_i)} \alpha_{t_i, w_i}^{g(t_i, w_i)} \alpha_{w_{i-1}, w}^{g(w_{i-1}, w_i)} (\alpha_{w_{i-2}, w_{i-1}, w_i}^{g(w_{i-2}, w_{i-1}, w_i)} - 1) \qquad (3.49)
\end{aligned}$$

All summations in (3.47), (3.48) and (3.49) can be obtained for free when computing $p[g(w_i)]$. It can be easily checked that the time of computing all feature expectations in the topic model (3.39) is the same as that of computing z for all seen histories as shown in Section 3.6.

3.8 Examples: The Training of Several Practical ME Models with Non-nested Features

We implement three kinds of ME models with non-nested features in this dissertation: the topic-dependent model, the syntactic model and the composite model with both topic and syntactic dependencies. The training of topic models has been described in the previous section. In this section we provide some details of training the syntactic model and the composite model.

3.8.1 Training ME Models with Syntactic Constraints

We can obtain meaningful syntactic information via parsing the sentence. For each word we can use its last two exposed head words (h_{i-2}, h_{i-1}) and their non-terminal labels $(nt_{i-2},\ nt_{i-1})$ of the partial parse T as predictors. We use them in combination with the immediate history $(w_{i-2},\ w_{i-1})$ to predict w_i. We will introduce details on how to extract syntactic heads and show why they are helpful in language modeling in Chapter 5. The syntactic heads are represented in the same form as N-grams. We will build a syntactic model with the form

$$p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}) \tag{3.50}$$
$$= \frac{\alpha_{w_i}^{g(w_i)} \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)} \alpha_{h_{i-2},h_{i-1},w_i}^{g(h_{i-2},h_{i-1},w_i)} \alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i)} \alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})}.$$

This is a 33-gram model. The hierarchical structure of the syntactic model is shown in Figure 3.13.

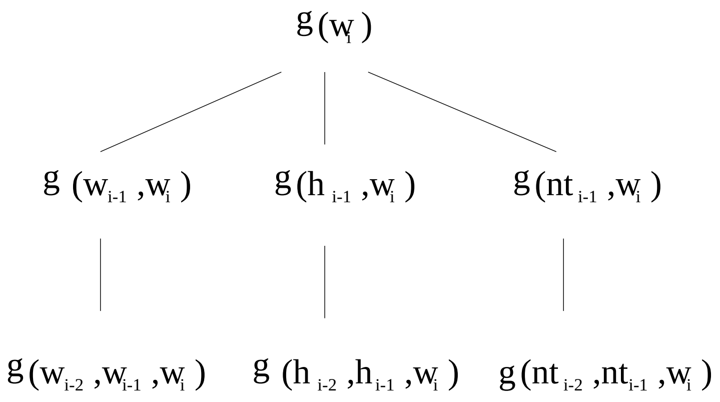

Figure 3.13: Structure of syntactic model

We group three order 2 features $g(w_{i-1}, w_i)$, $g(h_{i-1}, w_i)$ and $g(nt_{i-1}, w_i)$ in an order 2 super-feature $f_2(w_{i-1}, h_{i-1}, nt_{i-1}, w_i)$, three order 3 features $g(w_{i-2}, w_{i-1}, w_i)$, $g(h_{i-2}, h_{i-1}, w_i)$ and $g(nt_{i-2}, nt_{i-1}, w_i)$ in the super-feature $f_3(w_{i-2}, w_{i-1},\ h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, w_i)$, and set $f_1(w_i) = g(w_i)$. Thus the syntactic model (3.50) can be rewritten as

$$P(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}) \tag{3.51}$$
$$= \frac{\beta_1^{f_1(w_i)} \cdot \beta_2^{f_2(w_{i-1},h_{i-1},nt_{i-1},w_i)} \cdot \beta_3^{f_3(w_{i-2},w_{i-1},h_{i-2},h_{i-1},nt_{i-2},nt_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})}$$

where

$$\beta_1 = \beta_{w_i} = \alpha_{w_i} \tag{3.52}$$

$$\beta_2 = \beta_{w_{i-1},h_{i-1},nt_{i-1},w_i} = \alpha_{w_{i-1},w_i}\alpha_{h_{i-1},w_i}\alpha_{nt_{i-1},w_i} \tag{3.53}$$

$$\beta_3 = \beta_{w_{i-2},w_{i-1},h_{i-2},h_{i-1},nt_{i-2},nt_{i-1},w_i} = \alpha_{w_{i-2},w_{i-1},w_i}\alpha_{h_{i-2},h_{i-1},w_i}\alpha_{nt_{i-2},nt_{i-1},w_i} \tag{3.54}$$

and can be trained by the generalized hierarchical training method.

We also split the computation for bigram features to 2^3 parts according the method for overlapping features in Section 3.4.[11] However, we do not split the computation for trigram features for two reasons. First the straight-forward implementation is already quite efficient since Y_3 contains only a few words for each history. Furthermore, even though using the method in Section 3.4 may still reduce the computational load slightly, it requires an inordinate amount of space in experiments to save the intermediate results.

3.8.2 Training Composite ME Model with Both Topic and Syntactic Dependencies

Finally, we combine both topic dependencies and syntactic-structure dependencies with N-grams in one language model:

$$p(w_i|w_1^{i-1}) \approx p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, t) \tag{3.55}$$

where t_i is a deterministic function of the history w_1^{i-1}.

We build the following ME model to estimate the probability above.

$$p(w_i|w_{i-1}, w_{i-2}, h_{i-1}, h_{i-2}, nt_{i-2}, nt_{i-1}, t_i)$$
$$= \frac{\alpha_{w_i}^{g(w_i)}\alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)}\alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)}\alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)}\alpha_{h_{i-2},h_{i-1},w_i}^{(h_{i-2},h_{i-1},w_i)}\alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i}\alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)}\alpha_{t_i,w_i}^{g(t_i,w_i)}}{z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, t_i)}.$$

All our training simplification methods described in the preceding sections can be used in training this model. Since it is a topic-dependent model, we can use a divide-and-conquer scheme. We collect N-gram and syntactic statistics from different topics

[11] This brings about a twofold speed-up.

and merge $g(w_i)$ with $g(t_i, w_i)$. Now, within each topic, it looks like the syntactic model (3.50). Therefore, we can train this model as we did the syntactic model by using the GHT method.

3.9 Experimental Results for Speed-up

The efficiency of training methods is evaluated by the Switchboard and the Broadcast News data. We use 2.1 million words in the form of 1100 conversations on about 70 different topics in the Switchboard corpus. The Broadcast News corpus contain 125,000 stories amounting to about 130 million words that are clustered into 100 topics. Table 3.5 provide some numbers to show the complexity of the Switchboard and the Broadcast News tasks.

		Switchboard	Broadcast News
General Info	Vocabulary Size	22,000	64,000
	Training Size	2.1M	130M
	Topics	70	100
Trigram Model	N-grams	(22+300+180)K	64K+3.5M+5.6M
	Histories	300K	7.6M
	Tuples	830K	36M
Topic Model	Topic unigram	+15K	+250K
	Histories	770K	29M
	Tuples	1.4M	73M
Syntactic Model [12][13]	Syntactic N-gram	300K	2.6M
	Histories	760K	14M
	Tuples	2.5M	27M
Composite Model	Histories	8.4M	23M[14]
	Tuples	10M	42M

Table 3.5: Comparison of Switchboard and Broadcast News.

[12]Training size for syntactic models expands as multiple candidate parses are considered for each sentence.

[13]The syntactic model for Broadcast News is trained only on a subset of 14 million words of the corpus.

[14]Many low count histories have been merged into history equivalence classes.

3.9.1 Nominal Speed-up vs. Real Speed-up

The training time is roughly proportional to the number of terms to be summed up for all z's. We count this number and estimate the running time of a method according to it. The nominal speed-up is defined as

$$\text{nominal speed-up} = \frac{\#[\text{terms in the baseline method}]}{\#[\text{terms in the new method}]}, \tag{3.56}$$

where the baseline method is the Improved Iterative Scaling with unigram caching.

We measure the training time of four kinds of maximum entropy models: the trigram model, the topic dependent trigram model, the syntactic model and the composite model for the Switchboard and Broadcast News tasks. The real running time speed-up is the ratio of the training time of unigram-caching and that of the improved methods. Since some models cannot be trained at all by the baseline method, we do not know the real speed-up of them. The benchmark test for speed is based on 300MHz Ultra-Sparc II 64-bit systems with 1GB memory[15]. All numbers shown in this section are in CPU-hours per iteration.

3.9.2 Hierarchical Training for N-gram Models

The computational loads of straight-forward implementation and unigram-caching are first estimated for comparison. Table 3.6 illustrates that unigram-caching already reduces the computational load by about two orders of magnitude compared to the naive implementation. Therefore, we treat unigram-caching as the baseline method in our evaluation and compare it with the hierarchical training method.

The results for trigram models are illustrated in Table 3.6. Columns 3, 4 and 5 provide the number of terms in the straight-forward implementation, the baseline method and the hierarchical method, respectively. The last column is the nominal speed-up of (3.56). It is apparent that the hierarchical training method achieves a nominal speed-up of more than two orders of magnitude for both corpora, and the gain is greater as the size of the model increases.

[15]These machines were the fastest ones available when we started the work in this dissertation four years ago. Readers may keep in mind that the training time reported in this dissection can be reduced by two-threefold using current machines.

LM Task	Model Size	ME Algorithms			Nominal Speed-up
		Naive	Baseline	Hierarchical	
SWBD	500K	$6 \cdot 10^9$	$1.6 \cdot 10^8$	$9 \cdot 10^5$	170
BN	9.1M	$3.6 \cdot 10^{11}$	$2.4 \cdot 10^{10}$	$4.3 \cdot 10^7$	560

Table 3.6: Number of operations and nominal speed-up of trigram models

We also estimate the nominal speed-up of four-gram models (in Table 3.7).

LM Task	Model Size	ME Algorithms		Nominal Speed-up
		Baseline	Hierarchical	
SWBD	1.6M	$3.2 \cdot 10^9$	$2.1 \cdot 10^6$	1600
BN	13M	$3.4 \cdot 10^{12}$	$9.4 \cdot 10^7$	$3.6 \cdot 10^4$

Table 3.7: Number of operations and nominal speed-up of four-gram models

The computational load of training four-gram models using unigram-cache increases tremendously compared to that of training trigram models (20-120 fold), but it only increases slightly if hierarchical training methods are used (in about twofold). The nominal speed-up for four-gram models is much greater than that for trigram models.

The real training time of using different methods is also measured based on trigram models (Table 3.8). To train an ME trigram model for Switchboard, the baseline method takes about two CPU-hours, while the hierarchical training method needs only four CPU-minutes (with a speed-up of 30 fold)[16]. The speed-up for Broadcast News (85 fold) is greater than that for Switchboard. It is worth mentioning that the running time of less than one hour per iteration for such a huge corpus is quite fast and is even comparable to the time of training a back-off model[17].

[16]Real running time speed-up is less than the nominal one because of the overhead time, and I/O time are fixed no matter what training method is used.

[17]Using SRI LM toolkit V0.98.

LM Task	ME Algorithms		Real Speed-up
	Baseline	Hierarchical	
SWBD	2	0.06	30
BN	60	0.7	85

Table 3.8: Running time (in CPU-Hours) for ME trigram models

3.9.3 Generalized Hierarchical Training for Syntactic Model

The efficiency of generalized hierarchical training method is evaluated by syntactic models, which have both nested features and overlapping features. The nominal speed-up is shown in Table 3.9. For Broadcast News, we only provide statistics for a subset of the corpus because parsing the whole corpus of 130M words and storing the parsing results challenge our computer speed and storage capacity.

LM Task	ME Algorithms			Nominal Speed-up
	Baseline	Cluster Expansion	Hierarchical	
Switchboard	$6.9 \cdot 10^9$	$4.6 \cdot 10^9$	$7.5 \cdot 10^7$	$9 \cdot 10^1$
Broadcast News (14M)	$6.1 \cdot 10^{11}$	N/A	$4.2 \cdot 10^9$	$1.4 \cdot 10^2$

Table 3.9: Number of operations and nominal speed-up for syntactic models

The computational load increases considerably for the syntactic model (and will be even more for the composite model) compared to trigram models. The cluster expansion method described in Lafferty & Suhm (1996) is of little help here since the number of interacting features is so large. The hierarchical training will reduce the computational load by 90 fold in Switchboard and 140 fold in Broadcast News.

In Table 3.10, we compare the real running time of training syntactic models using the generalized hierarchical training method and that using the baseline method. We do not provide the baseline training time (without the speed-up) for Broadcast New, because it is impractical to train the syntactic model on such a large corpus using unigram-caching. It is apperant from the results in Switchboard that the hierarchical method achieves a considerable speed-up. Nevertheless, we may expect the speed-up in Broadcast News to be even greater than in Switchboard.

LM Task	ME Algorithms		Real Speed-up
	W/O Hierarchical	W/ Hierarchical	
Switchboard	100	6	17
Broadcast News	N/A	480 [18]	N/A

Table 3.10: Training time (in CPU-hours) for syntactic models

3.9.4 Divide-and-Conquer and Topic-Dependent Model

Divide-and-conquer is designed for and evaluated on topic-dependent models (Table 3.11). Comparing the trigram model (3.6) with the topic-dependent model (3.39), one can see that the number of operations in the baseline method will increase by about two orders of magnitude even though the model size increases by only 3%. This load can be decreased in about one order of magnitude by the generalized hierarchical training. However, this computational load is still heavy even for the Switchboard. Divide-and-conquer strategy individually will cut the computational load in Switchboard down to a manageable scale. Divide-and-conquer and hierarchical training can be applied together to make the training of the topic-dependent model practical for Broadcast News. Overall, the nominal speed-up for the topic-dependent model is about 400 fold for the Switchboard and is more than 1000 fold for the Broadcast News, if both divide-and-conquer and hierarchical training are used.

LM Task	ME Algorithms				Nominal Speed-up
	Baseline	Hierarchical	Divide-Conquer	H+DC	
SWBD	$1.6 \cdot 10^9$	$1.9 \cdot 10^8$	$6 \cdot 10^7$	$4.1 \cdot 10^6$	$4 \cdot 10^2$
BN	$1.8 \cdot 10^{11}$	$5.1 \cdot 10^9$	$6.6 \cdot 10^9$	$1.5 \cdot 10^8$	$1.3 \cdot 10^3$

Table 3.11: Number of operations and nominal speed-up for topic models

[18]The model is trained on 1.2GHz PIII machines in 180 $\sim$ 200 CPU-hours per iteration. One iteration was run on Sun Sparc machines to compare for the training speed with those of other models.

The training of a topic dependent model without any speed-up takes a very long time (160 CPU-hours) even for Switchboard and it is impractical for Broadcast News. However, it needs only 0.5 and 2.3 CPU-hours, respectively, to train the topic models in the corpora above. The real running time speed-up for Switchboard is more than two orders of magnitude based on unigram-caching.

We do not report the baseline training time for Broadcast News because unigram-caching is not practical here. However, we can expect that the real speed-up in Broadcast News is greater than that in Switchboard. We roughly estimate the training time of using generalized hierarchical training and divide-and-conquer individually by using about 5% of the data and increasing the resulting training time by 20 fold. The estimated time (numbers inside parentheses) is shown in Table 3.12.

It is also worth mentioning that the real running time of 2.3 CPU-hours per iteration for the topic model with 100 topics is quite fast and is about the same as that of training the corresponding interpolation topic models[19]. Table 3.12 shows detailed results.

LM	ME Algorithms				Real
Task	Baseline	Hierarchical	Divide-and-conquer	Both	Speed-up
SWBD	165	21	8.3	0.5	$3.3 \cdot 10^2$
BN	-	(85)	(100)	2.3	-

Table 3.12: Running time (in CPU-Hours) for topic models

3.9.5 Divide-and-Conquer Combined with Generalized Hierarchical Training

If the topic model can still be trained using either divide-and-conquer or generalized hierarchical training, the composite model with both the topic and the syntactic constraints must be trained using both of them. We train the composite model in less than 10 CPU-hours per iteration in Switchboard. We can train this model without

[19]Using SRI LM toolkit V0.98 by Andreas Stolcke.

Task &	ME Algorithms		Real
Model	DC, W/O Hierarchical	DC + Hierarchical	Speed-up
SWBD Composite	150	9.5	15

Table 3.13: Training time for composite model in Switchboard

generalized hierarchical training method in a much longer time (150 CPU-hours) but cannot train this model without divide-and-conquer even in Switchboard. Therefore, we only show the gain from the generalized hierarchical training method in Table 3.13.

We also implement a composite model on 14M Broadcast News data using both speed-up methods above. The training time is roughly 400 CPU-hours per iteration, which is quite long. However, this model is impractical to train without our speed-up methods.

3.10 Summary

Several rapid training methods for maximum entropy models have been presented in this chapter. In particular, N-gram models are trained hierarchically and almost as efficiently as back-off models, whose training time is just linear in the size of training data. Experimental results on Switchboard and Broadcast News show that computational load and real training time reduce tremendously by hierarchical training methods. The speed-up of a factor of tens has been achieved compared to the baseline unigram-caching method.

Other models with overlapping features, e.g., syntactic models, can also benefit from the hierarchizing of features and can thus be trained effectively by the generalized hierarchical training. The time of training the syntactic model and the composite model (with both the topic and syntactic constraints) reduces by more than one order of magnitude after this method is used.

Divide-and-conquer can be used in addition to the generalized hierarchical training method in any model with topic constraints. The combination of these two methods makes it possible to train large topic-dependent models and composite models that otherwise are intractable to train.

It is worth mentioning here that the computational load of training ME models can be distributed among many computers using the idea of divide-and-conquer. The speed-up is almost linear in the number of computers.

Chapter 4

Topic Dependent Language Models

In this chapter, we explore topic dependencies between words and documents and improve the performance of language models by exploiting topic dependencies. We cluster the training data into topics, extract topic-dependent salient features from each topic and build maximum entropy models with both topic constraints and regular N-gram constraints. The training issues have been stated in Chapter 3. We will provide experimental results with detailed analysis based on Switchboard and Broadcast News. We also compare the maximum entropy method with other approaches reported in the literature.

4.1 Introduction

N-gram language models treat words in sentences simply as abstract symbols without considering their semantic roles in the language and their syntactic roles in sentences. The speech recognizer with a regular N-gram model may not be able to choose the correct word from two words with close pronunciations and similar collocational frequencies. For example, *gas and guys* after the words *lot of* are difficult to recognize in Switchboard because they are acoustically similar and have the same relative frequencies: $f(gas|lot\ of)$ and $f(guys|lot\ of)$ both equal 0.0006.

Humans benefit from both semantic and syntactic knowledge in understanding natural language. If people hear the potentially confusable utterances *lot of gas/guys* when discussing buying a car, they may tend to choose "gas" instead of "guys" since the former one is relevant to the topic of discussion while the latter one is less so. Actually the relative frequency of $f(gas|lot\ of)$ is boosted 15 fold to 0.009 in the topic *buying a car* in Switchboard while $f(guys|lot\ of)$ drops nearly to zero.

In general, the use of words in sentences depends on the topic of discussion. This relationship has been successfully used to discern the topic of a document in information retrieval (IR). In language modeling, an accurate estimation of the probability of these topic relevant words may be obtained via determination of the semantic content of the document, *e.g.*, by topic detection.

To understand the influence of topics on word frequencies quantitatively, we show a concrete example from the Switchboard corpus, a human to human telephone speech database on 70 assigned topics such as *education, sports, child care* and *automobile.* Each topic contains several hundred to several thousand utterances, and thousands to tens of thousands of words. The whole corpus comprises about 260,000 sentences, amounting to about 3 million words. A vocabulary of about 22,000 words is chosen to cover all words in the training and test data. The frequency of many content-bearing words in a specific topic varies significantly from that in the whole corpus. Table 4.1 lists the relative frequency of some words in the topic *clothing* and that in the whole Switchboard training corpus. The former (in Column 2) is at least one order of magnitude greater than the latter (in Column 3).

A straightforward way of adding the topic information in language models is to classify the training data according to the topic and estimate a topic-specific N-gram model $p_t(w_i|w_{i-N+1}, \cdots, w_{i-1})$ for each topic t from the training data of this topic. Unfortunately, there exists a severe data sparseness problem. In Switchboard, no topics have more than one hundred thousand training samples (words), which is in the same order of magnitude as the vocabulary size. The estimate of a trigram probability $p_t(w_i|w_{i-2}, w_{i-1})$ (with 8 trillion parameters) will be very unreliable based on such a small training set. Of course, such a probability estimate can be interpolated with or backed-off to an overall trigram probability (or the topic-dependent bigram probability) as in Iyer & Ostendorf (1996) and Florian & Yarowsky (1999). However, this implementation is not space efficient since each topic must have its own language

Word	Freq in Clothing	Freq in whole Corpus	log difference
APPEARANCE	0.00045181	1.57886e-05	1.4566
ATTIRE	0.00045181	3.94714e-06	2.0586
ATTORNEYS	0.00060241	2.76300e-05	1.3385
AVON	0.00030121	7.89428e-06	1.5815
BACKLESS	0.00030121	2.63143e-06	2.0587
BAKERY	0.00060241	1.05257e-05	1.7576
BLOUSE	0.00060241	1.05257e-05	1.7576
BLOUSES	0.00090361	1.05257e-05	1.9337
BOOTS	0.00090361	1.97357e-05	1.6607
BAGGY	0.00030121	3.94714e-06	1.8847
BOOTS	0.00090361	1.97357e-05	1.6626

Table 4.1: Sensitive words for the topic "CLOTHES".

model.

Instead of using either the interpolation or the back-off method, we build a maximum entropy model to capture the topic-dependencies. In our method, unigram frequencies in a specific topic t are treated as *topic-dependent* salient features, just as overall N-gram frequencies are *topic-independent* salient features. Topic-dependent models $p(w_i|w_{i-N+1}, \cdots, w_{i-1}, t)$ are constructed in which the probability of a word w_i depends not only on the N-1 preceding words, $w_{i-N+1}, \cdots, w_{i-1}$, but also on the topic t of the history. The maximum entropy method is used to select the "smoothest" statistical model from these models.

4.2 Exploiting Topic Dependence

To obtain the topic-sensitive features, we first classify the training data by topic and for each topic extract words whose frequencies within the topic deviate significantly from their overall distribution. Robust topic detection algorithms used in information retrieval can be applied to the clustering of training data and topic assignment for the test data.

4.2.1 Topic Classification

We construct topic-dependent language models for both Switchboard and Broadcast News (BN). The Switchboard corpus has been introduced in Section 4.1. Here, we briefly introduce the Broadcast News corpus. The Broadcast News corpus consists of news dispatches (stories) broadcast between 1994 and 1996. The training set contains about 125,000 stories amounting to about 130 million words (including the special *end-of-sentence* token). We treat each conversion side in Switchboard or story in Broadcast News as an individual document that is the basic granule for topic clustering. For each document, we create a term frequency/inverse document frequency (TF-IDF) vector of all words in the vocabulary excluding stop words[1], where the TF-IDF unigram frequency x_i for the word w_i is defined as

$$x_i = f(w_i) \cdot \log \frac{D}{D_{w_i}} \tag{4.1}$$

in which D and D_{w_i} are the total number of documents and the number of documents containing w_i, respectively. The factor $\log \frac{D}{D_{w_i}}$ boosts the frequency of topic-specific words while lessening the weight for those words appearing in most of the topics.

We let $X = x_1, \cdots, x_{|V|}$ be the TF-IDF vector of the document where $|V|$ is the vocabulary size and Y be the similar vector for the centroid. We then define the smoothed cosine distance following Florian & Yarowsky (1999) as

$$D(X,Y) = 1 - \frac{X \cdot Y + \epsilon \sum_{x_i>0} x_i + \epsilon \sum_{y_i>0} y_i + \epsilon^2}{(\parallel X \parallel + \epsilon) \cdot (\parallel Y \parallel + \epsilon)} \tag{4.2}$$

where $|| \cdot ||$ denotes the L_2 norm for the parameter and ϵ is a small positive real number [2] less than 1.

We use the following standard K-means clustering method to classify conversations into different topics.

Algorithm (K-means Clustering)

Initial Step:
Determine a set of K centroids according to pre-assigned topic labels of

[1] A stop word is any of a set of function words with low semantic content that will be ignored by the topic classifier.

[2] $\epsilon = 0.2$ in our experiments.

each document.
Let $C^0(X_i)$ be the initial class for the document $X_i, i = 1, 2 \cdots, D$, and set

$$Y_j^0 = \sum_{i:C^0(X_i)=j} X_i \text{ for } j = 1, 2, \cdots, K.$$

In Switchboard, the initial cluster assignments are derived from the 70 manually assigned topics of the conversations, whereas in Broadcast News, initial topic tags are assigned by the bottom-up hierarchical clustering algorithms provided by Florian & Yarowsky (1999).
Iteration Steps:
- Assign each document to its nearest centroid according to cosine distance.
Let $C^n(X_i)$ be the new centroid for $X_i, i = 1, 2, \cdots, D$, where

$$C^n(X_i) = \arg \min_{j=1}^{K} D(X_i, Y_j^{n-1}).$$

Note that $D(X_i, Y_j^{n-1}) = \infty$ if Y_j is empty.
- Recalculate the centroids based on the new clustering of dialogues.

$$Y_j^n = \sum_{i:C^n(X_i)=j} X_i \text{ for } j = 1, 2, \cdots, K.$$

Repeat the iteration steps until the centroids are "stable" [3].

∎

Sixty-seven of seventy initial clusters are obtained for the Switchboard. Only 3% of the conversation sides out of 2200 are assigned different topics from their initial ones after re-clustering. For the Broadcast News, 100 clusters are generated and about 20% of stories change topics after the re-clustering.

4.2.2 Selection of Topic Sensitive Words

We design a non-singleton word w as a topic-related word if its relative frequency $f_t(w)$ in topic t differs significantly from its relative frequency $f(w)$ in all training

[3] If a document X_i of topic Y_j is closer to another cluster $Y_{j'}$ but the difference of cosine distances is not greater than a threshold (very small) , it does not switch to $Y_{j'}$ but stays in the original Y_j.

data, *i.e.*,

$$|D(f_t(w)||f(w))| = \left| f_t(w) \cdot \log \frac{f_t(w)}{f(w)} \right| \geq T_t$$

where D is the I-divergence and T is a threshold.[4] In Switchboard, we find 8459 different topic-sensitive words (40% of the vocabulary) and 15442 topic-word combinations (t, w pairs). Each topic has 231 sensitive words on average, and each topic-sensitive word belongs to 1.8 topics on average (and 8 maximum). In Broadcast News, 95% of words (59989 out of 63032) in the vocabulary are topic-sensitive words. Each topic has about 2,500 topic-sensitive words on average and each words belongs to 5 topics on average (with the maximum number being 15). There are a total 249295 topic-word combinations.

4.2.3 Formulation of the ME Topic Model

We use the long-range history $w_1, \ldots, w_{i-1}$ to assign a topic $t_i = t(w_1, \ldots, w_{i-1})$ to a test utterance. The sufficient statistic of the history is thus the triple $\phi(h_i) = [w_{i-1}, w_{i-2}, t_i]$, and we set

$$p(w_i|w_1, \ldots, w_{i-1}) \approx p(w_i|w_{i-1}, w_{i-2}, t_i). \tag{4.3}$$

Of course, not every word in the vocabulary will have strong dependence on the topic. Estimating a separate conditional *pmf* for each $[w_{i-1}, w_{i-2}, t_i]$ fragments the training data and may result in poor estimates for these words. In additional, topic-related words may not appear in every word-context $[w_{i-1}, w_{i-2}]$. We therefore seek a model which, in addition to topic-independent N-gram constraints,

$$\sum_{t_i} p(w_i|w_{i-1}, w_{i-2}, t_i)p(w_{i-2}, w_{i-1}, t_i) = \frac{\#[w_{i-2}, w_{i-1}, w_i]}{\#[training\ data]}, \tag{4.4}$$

$$\sum_{t_i, w_{i-2}} p(w_i|w_{i-1}, w_{i-2}, t_i)p(w_{i-2}, w_{i-1}, t_i) = \frac{\#[w_{i-1}, w_i]}{\#[training\ data]}, \tag{4.5}$$

$$\sum_{t_i, w_{i-2}, w_{i-1}} p(w_i|w_{i-1}, w_{i-2}, t_i)p(w_{i-2}, w_{i-1}, t_i) = \frac{\#[w_i]}{\#[training\ data]}, \tag{4.6}$$

[4]We choose $T_t = \frac{3}{C_t}$ where C_t is the number of words in document t.

meets topic-dependent *marginal* constraints

$$\sum_{w_{i-1},w_{i-2}} p(w_i|w_{i-1},w_{i-2},t_i)p(w_{i-1},w_{i-2},t_i) \quad = \quad \frac{\#[t_i,w_i]}{\#[training\ data]}. \tag{4.7}$$

Note that these marginal probabilities are much more reliably estimated from counts $\#[\cdot]$ of the corresponding events observed in the corpus than are the conditional probabilities in (4.3). For example, only 2% of 4-tuples $(w_{i-2},w_{i-1},w_i,t_i)$ in the Switchboard test data occur in the training data. However, for 63% of the test data, either (w_{i-2},w_{i-1},w_i) or (t_i,w_i) does not appear in the training set.

Unreliable marginal probabilities, *e.g.*, those based on one or two observations, may be completely left out of the model's requirements. In addition to leaving out some constraints, the relative frequency estimates of the marginal probabilities on the right-hand sides of equations (4.4)-(4.7) may be replaced by their corresponding Good-Turing (Good, 1953) estimates. Finally, since our primary interest is in the conditional model $p(w_i|w_{i-1},w_{i-2},t_i)$, empirically observed relative frequencies $\hat{p}(w_{i-1},w_{i-2},t_i)$ may be substituted for the model-based probabilities $p(w_{i-1},w_{i-2},t_i)$ in equations (4.4)-(4.7).

Linear constraints of the form described in (4.4)-(4.6) define a family of *pmfs*, and we choose the model in this family that has the highest entropy, corresponding qualitatively to the least additional assumptions on (or maximal smoothness of) the model. As explained in Chapter 2, the ME model has an exponential form, with one parameter α corresponding to each linear constraint placed on the model:

$$p(w_i|w_{i-1},w_{i-2},t_i) = \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{(t_i,w_i)}}{z(w_{i-1},w_{i-2},t_i)} \tag{4.8}$$

where $z(w_{i-1},w_{i-2},t_i)$ is a suitable normalization constant. The first three numerator terms correspond to standard N-gram constraints, whereas the fourth one is a topic-unigram parameter determined by word-frequencies in particular topics.

4.2.4 Topic Assignment to the Test Utterance

To use a topic-dependent model for rescoring, a topic must be assigned to test utterances automatically. Two issues arise when a topic-dependent language model for speech recognition is used. First, since the actual spoken words (or reference transcriptions) are not available for topic assignment, topic assignment must be based on recognition results. We study the impact of recognition errors on this process. Second, the topic of a document may change as the document progresses. We examine whether a topic should be assigned to an entire test conversation or to each utterance. If the topic assignment is on the utterance level, it can be based on

(i) that utterance only,
(ii) that utterance and a few preceding utterances, or
(iii) that utterance, a few preceding and following utterances.

The results will be presented in Section 4.3.2.

To detect topics automatically, the weighted TF-IDF vectors based on real hypotheses [5] of the test utterances (instead of truth) are generated for each document or each utterance. The closest centroid is chosen for each document or each utterance according to the cosine similarity between vectors of the test data and the centroids.

4.3 Switchboard Results

Most of the experiments are based on the Switchboard task. We use 1,100 conversations amounting to 2.1 million words to train the language model. The acoustic models used for recognition are state-clustered cross-word triphone HMMs with 12 Gaussian mixture output densities, trained with MF-PLP acoustic features from about 60 hours of speech data. The test set (WS97 dev-test set) has a little over 2 hours of speech in the form of 19 conversations containing about 2400 utterances amounting to 18,000 words.

For every test utterance, a list of the 100-best hypotheses is generated by an HTK-based recognizer (Young *et al.*, 1995) using a back-off trigram language model. The trigram model has about 500K constraints and the topic-dependent model contains 15K topic constraints besides N-grams. The recognition word error rate (WER) for *rescoring* these hypotheses and the average perplexity of the transcriptions of the test set are reported here.

[5]The 10 best-hypotheses are used in this research

4.3.1 Baseline Results

Table 4.2 shows the performance of standard Katz (1987) backoff trigram models built by SRI LM toolkits (Stolcke, 1996) and an ME model with only Ngram constraints. The minimum count for a bigram to be included in a model is indicated by B, and that for a trigram by T. All counts of less than 8 are replaced by their corresponding Good-Turing (Good, 1953) estimates here and also in the remainder of the dissertation. It should be noted that singleton trigrams have no constraints on their probability in the ME model.

The back-off model with no bigram cutoff (i.e., the bigram cutoff of 1) and the trigram cutoff of 2 performs best among all back-off trigram models. The ME model performs only slightly (but significantly[6]) better than the corresponding back-off model when only N-gram constraints are specified. Thus any further improvement of the language model by adding long-range dependence may be credited to adding those new features to N-gram constraints rather than to any inherent advantage of the ME method.

Model (N-gram cutoffs)	Perplexity	WER
Back-off (no cutoffs)	79.2	38.7%
Backoff ($B \geq 1$, $T \geq 2$)	78.8	38.5%
ME Trigram	78.9	38.3%

Table 4.2: Perplexity and WER of back-off trigram models and an ME model with the same constraints

4.3.2 Topic Assignment During Testing

The topic model (4.8) is trained next. The recognition performance depends on the detection of the topic. A hard decision is made by assigning the closest matching topic in the results presented here, although the formalism extends easily to soft topic decisions. We first fix the topic for the whole conversation and test the performance of our language models. Next, we let the topic shift inside a conversation.

[6]Based on the sign-test introduced in Appendix A.

Topic Assignment Based on Conversations

We investigate four options for this assignment:

(i) manual assignment of topics to the conversation,
(ii) automatic topic assignment based on the reference transcriptions, [7]
(iii) automatic topic assignment based on the 10-best hypotheses generated by the first recognition pass, and
(iv) *oracle* assignment to minimize perplexity of the test set.

Source of Text for Topic Classification	Perplexity	WER
None (Baseline)	78.8	38.5%
Manual Assignment	73.1	37.8%
Ref. Transcriptions	73.8	37.8%
10-Best Hypotheses	74.4	37.9%
Oracle (optimal)	72.5	37.7%

Table 4.3: Topic assignment based on erroneous recognizer hypotheses causes little degradation in performance.

Manual and automatic topic assignments are listed in Tables 4.4 and 4.5, in which any automatically assigned topic identical to the manual one is replaced by a $*$. It is apparent that 34 out of 38 conversation sides are assigned to the same topics (Column 3) as the manual tags (Column 2) when reference transcripts are used. The rest 4 conversation sides are assigned topics close to the manual ones. For example, conversation #2461A is assigned to the topic *choosing a college*, whereas the manual is *public education*. When the 10-best hypotheses are used instead of truth, 32 conversations are assigned to the same topics (Column 4) as the manual ones. These results indicate that the quality of automatic topic classification is quite good.

The results, presented in Table 4.3, clearly indicate that even with a WER of over 38%, there is only a small loss in perplexity and a negligible loss in WER when the

[7] The null topic, which defaults to a topic-independent baseline model, is available to the topic classifier as one of the choices.

Conv. Id	Manual	Ref.	N-best
2121A	air pollution	*	*
2121B	air pollution	*	*
2131A	music	*	*
2131B	music	*	*
2151A	universal public service	*	*
2151B	universal public service	*	*
2229A	exercise and fitness	*	*
2229B	exercise and fitness	*	*
2335A	crime	*	*
2335B	crime	*	*
2434A	gun control	*	*
2434B	gun control	*	*
2441A	universal public service	*	*
2441B	universal public service	*	*
2461A	public education	choosing a college	choosing a college
2461B	public education	*	choosing a college
2503A	pets	*	*
2503B	pets	*	*

Table 4.4: Topic assignments of test conversations using different methods.

topic assignment is based on recognizer hypotheses instead of the correct transcriptions. Comparisons with the oracle result indicate that there is little room for further improvement.

Topic Assignment Based on Utterances

A more precise topic assignment may be based on the utterance level. The topic of an utterance can be determined by

(i) itself, or
(ii) several preceding utterances and itself, or
(iii) several preceding and following utterances and itself.

Each utterance used for topic detection is assigned a positive weight ω. The maximum weight is assigned to the current utterance to emphasize its importance. The weight reduces linearly with an increase in the distance between that utterance and the

Conv. Id	Manual	Ref.	N-best
2632A	public education	choosing a college	choosing a college
2632B	public education	*	choosing a college
2724A	exercise and fitness	*	*
2724B	exercise and fitness	*	*
2752A	air pollution	*	*
2752B	air pollution	*	*
2753A	latin america	*	*
2753B	latin america	*	*
2836A	crime	*	*
2836B	crime	*	*
2838A	gun control	*	*
2838B	gun control	*	*
3528A	space exploration	*	*
3528B	space exploration	*	*
3756A	painting	home repair	home repair
3756B	painting	home repair	home repair
3942A	ethics in government	*	*
3942B	ethics in government	*	*
3994A	music	*	*
3994B	music	*	*

Table 4.5: Topic assignments of test conversations using different methods (Cont).

current one. In formal statements, the weight $\omega(u)$ of an utterance u is assigned as

$$\omega(u) = D - d(u)$$

where D[8] is the window size and $d(u)$ is the distance between u and the current utterance.

The experimental results show that even though a small window may result in lower perplexity than a longer one (Figure 4.1), the lowest word error rate is reached at the point of a moderate window size (Figure 4.2). It also shows that a two-sided window gives only slightly better results than the left-sided one[9] The difference between using a reference script and the 10-best list is relatively large for a small window since topic-sensitive words may appear only a few times in a small window. If some of them are missed, the topic assignment may not be accurate. However,

[8] We choose $D = 5$.

[9] In real time speech recognition, only preceding utterances and the current one are available.

this difference becomes small as the window size increases, since at least some topic-relevant words may appear enough times in the 10-best list in a window of moderate size.

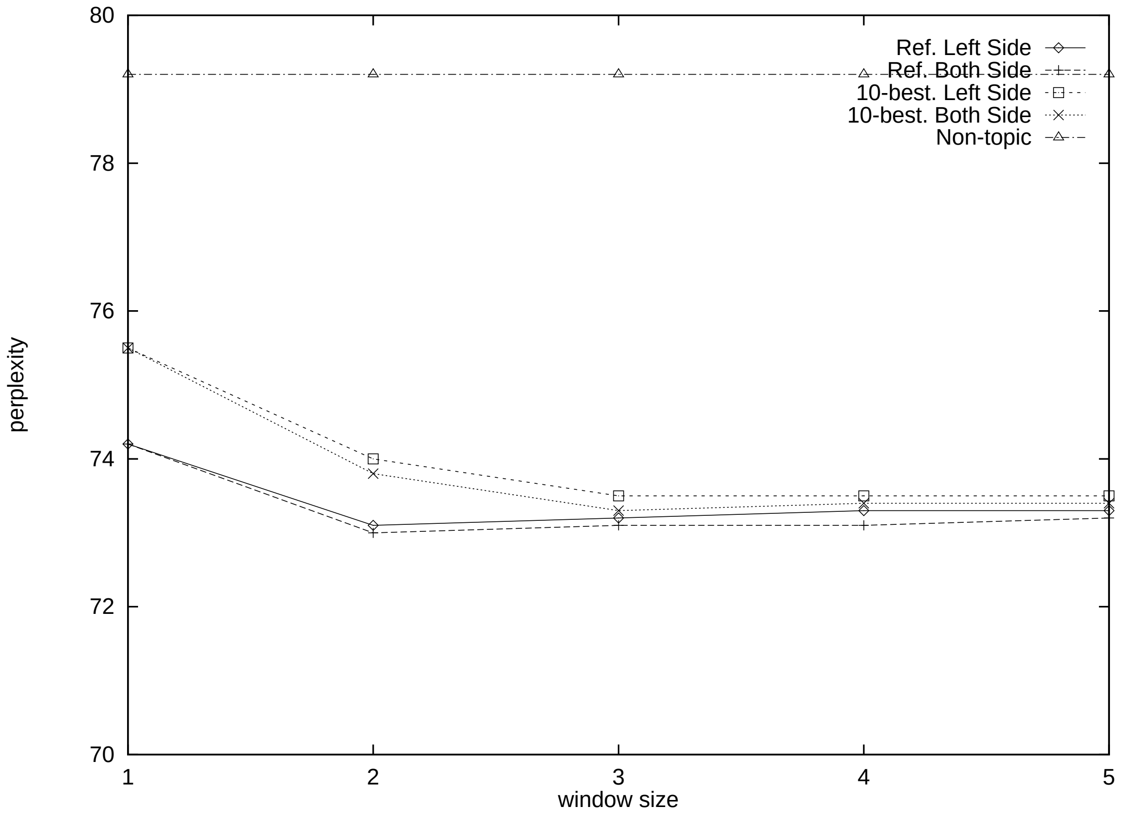

Figure 4.1: Perplexities for different window lengths, both left-sided and two-sided

The best recognition performance in WER (without looking at future utterances) is achieved by assigning a topic to each utterance based on the 10-best hypotheses of the current and the four preceding utterances. We fix the topic assignment based on this setup in the rest of our experiments.

We summarize the results of dynamic topic assignment in Table 4.6.

It is worth noting that utterance-level topic assignment of Table 4.6 is slightly more effective than the conversation level assignment (Table 4.3). Adding topic-dependent constraints reduces absolute WER by 0.7%, which is very significant[10], and relative perplexity by 7%.

[10]With a P-value of 10^{-5} in the sign-test, which will be introduced in Appendix A.

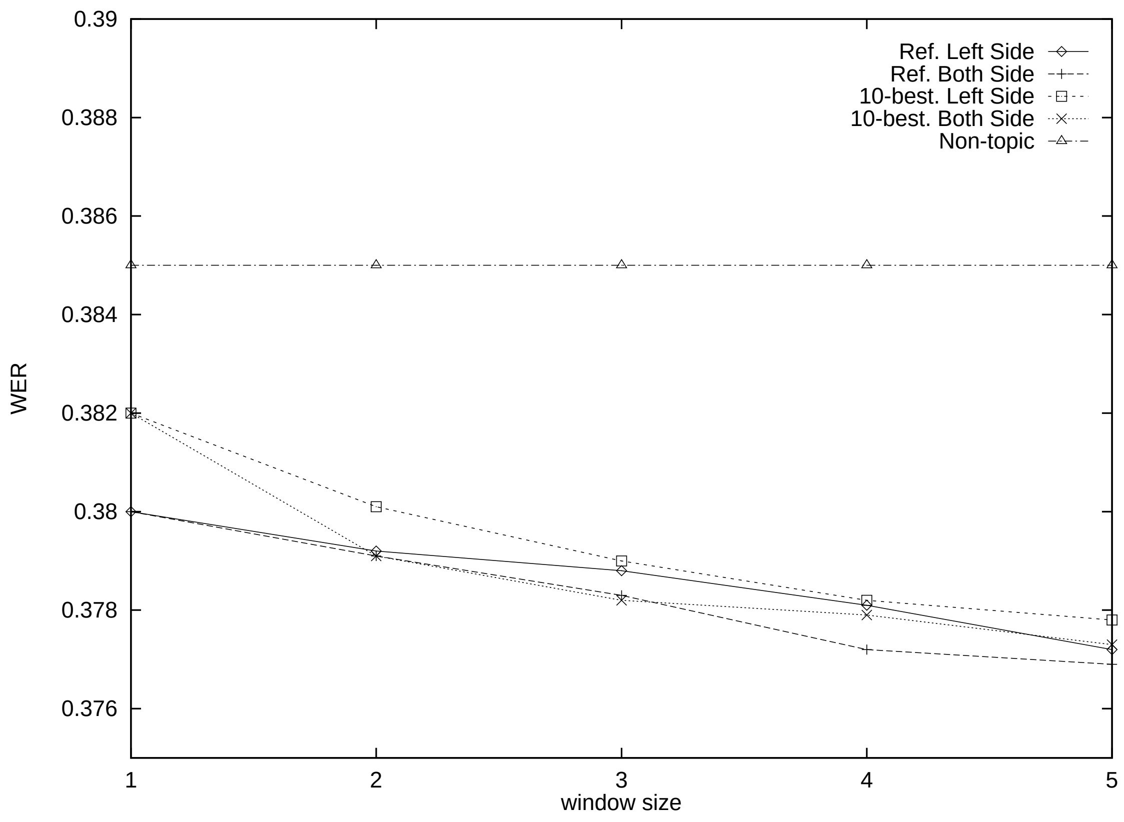

Figure 4.2: WER for different (left-sided) window lengths.

To gain insight into improved performance from utterance-level topic assignment, we examine agreement between topics assigned at the two levels. As illustrated in Table 4.7, 8 out of 10 utterances prefer the topic-independent model and are filler utterances, probably serving vital discourse functions (*e.g.* acknowledgments, back-channel responses). Of the remaining utterances, a majority (60%) are closest to the topic that was assigned at the conversation level. While a large fraction (40%) are closer to a topic other than the one preferred at the conversation level, this is not an equally remarkable result as, in many of these cases, the topic assigned at the conversation-level is a close second or the two topics are similar, *e.g.*, *public education* and *choosing a college.*

Source of Text for Topic Classification	Perplexity	WER
None (Baseline)	78.8	38.5%
Ref. Transcriptions	73.3	37.8%
10-Best Hypotheses	73.5	37.8%

Table 4.6: Dynamic topic assignment for individual utterances based on the current and four preceding utterances.

Source of Text for Topic Classification	Agreement of Conv. & Utt. Level Topics	Utt. Level Topic When Disagreeing With Conv.	
		Other Topic	No Topic
Ref. Trans.	12.7%	7.1%	80.3%
10-Best Hyps.	9.9%	7.0%	83.1%

Table 4.7: Topic dynamics viewed through (dis)agreement of utterance-level and conversation-level topic assignment.

4.3.3 Analysis of Recognition Performance

Topic-dependent constraints are chosen to manipulate the probabilities of words whose frequency within conversations on a particular topic are dramatically different from their frequency in the whole corpus. These are typically content-bearing words that are interrelated by the topic of the conversation. Thus, we expect improvements from the topic model to be concentrated on such words.

To see if we indeed improve the model where we aim to improve it, the vocabulary is divided into two sets: all words that have topic-conditional unigram constraints for any of the topics, and the others. About 7% of the tokens in the test set have topic-dependent constraints. Errors in recognizing each test token is classified in the same way, *i.e.* insertions or deletions of topic-dependent words as well as a substitution that involves a topic-dependent word counts towards an error in the first category. Table 4.8 shows a breakdown of the results over the set of topic-dependent and topic-independent words for ME models with and without topic-dependent constraints. All perplexity reduction comes from the topic-dependent words. The WER reduction on those topic-dependent words is 2.2%, which is much higher than the overall WER reduction.

Language Model (N-gram cutoffs)	Topic Words		Non-topic Words	
	Perplexity	WER	Perplexity	WER
ME (B$\geq$1, T$\geq$ 2)	3795	37.7%	62.2	38.5%
ME-Topic (B$\geq$1, T$\geq$ 2)	356	35.5%	66.6	37.9%

Table 4.8: Analysis of performance gains from the topic-dependent model.

Since the model is designed to benefit topic-dependent words, a fairer comparison may be based on all content-bearing words vs. stop words. Under this partition, about 23% of tokens in the test set are content-bearing and the remainder are stop words. Table 4.9 presents the performance gains analyzed for this partition.

Language Model (N-gram cutoffs)	Content Words		Stop Words	
	Perplexity	WER	Perplexity	WER
ME (B$\geq$1, T$\geq$ 2)	8941	42.2%	36.4	37.6%
ME-Topic (B$\geq$1, T$\geq$ 2)	3923	40.8%	37.1	37.0%

Table 4.9: Performance improvement on content-bearing words.

As expected, Table 4.9 shows that the gain in perplexity comes predominantly from content-bearing words, and the 1.4% improvement in WER on these words is greater than the overall WER improvement. The perplexity for stop words slightly increases, but the WER is also reduced because the correction of any content words will also help to avoid errors in its neighborhood. This may be a more important performance measure than the overall WER for tasks such as spoken document retrieval and automatic content extraction.

4.4 Comparison with Related Methods

4.4.1 Maximum Entropy vs. Linear Interpolation

Compared to the back-off trigram model, which has about 500K parameters, the topic-conditional ME models introduce only about 15K additional parameters that modify probabilities of a few hundred words in the context of each topic. An alternative to this modeling approach is to partition the training data, build separate N-gram models p_t for each topic t and, since each topic N-gram is trained on a much smaller data set, interpolate this topic-specific model with a topic-independent model trained on all the data to obtain a smooth topic-dependent model. This is comparable to the approach described, *e.g.,* in Clarkson & Robinson (1997), Iyer & Ostendorf (1996), Florian & Yarowsky (1999) and Martin, Liermann & Ney (1997).

We construct back-off unigram, bigram and trigram models specific to each topic using the partitioning of the 2.1 million word corpus used for the ME models as described in Section 4.2.3. We interpolate each topic-specific N-gram p_t with the baseline (topic-independent) trigram model denoted as p_3 to obtain smooth topic-dependent N-gram models p where

$$p(w_i|w_{i-2}, w_{i-1}) = \lambda p_3(w_i|w_{i-2}, w_{i-1}) + (1-\lambda)p_t(w_i|w_{i-k}, w_{i-k+1}),\ 0 \leq \lambda \leq 1 \quad (4.9)$$

for $k = 0, 1$ or 2 (for unigram, bigram and trigram models, respectively).

Usually, one would tune the interpolation coefficient λ on some held-out set. In this case, however, we (cheat and) choose the interpolation weight to minimize the perplexity of the test set under each interpolated model. Table 4.10 shows the recognition performance of the interpolated models. The topic for each test utterance for the interpolated model is the same as the one used for the ME topic model.

Column 2 in Table 4.10 shows the number of parameters in different models. Even though the ME model has far fewer parameters than interpolation models, it outperforms the latter ones. It may thus be argued that the ME approach permits us to combine via unigram constraints as much effective information as one would get by interpolating topic-specific trigram models. This, we argue, is due to the systematic integration of topic-dependent and topic-independent constraints in our model.

Model (N-gram cutoff)	#Params	Perplexity	WER
Back-Off (B≥1, T≥2)	499K	78.8	38.5%
Back-Off + Topic 1-gram	+70×11K	78.4	38.5%
Back-Off + Topic 2-gram	+70×26K	77.3	38.3%
Back-Off + Topic 3-gram	+70×55K	76.1	38.1%
ME-Topic (B≥1, T≥ 2)	+15K	73.5	37.8%

Table 4.10: Comparison with 70 interpolated topic N-gram models.

4.4.2 Topic Dependent Model vs. Cache-Based Model

The cache-based method (Kuhn & De Mori, 1990) is another widely used topic/domain adaptation technique for language modeling. We have implemented a cache-based language model for comparison with the topic-dependent maximum entropy model. Specifically, at each position i in the test corpus, we estimate a unigram probability p_c based on all the words seen so far in that particular conversation side. This unigram probability is then interpolated with the back-off trigram probability p_3 as

$$p(w_i|w_{i-2}, w_{i-1}) = \lambda p_3(w_i|w_{i-2}, w_{i-1}) + (1 - \lambda)p_c(w_i) \ , 0 \leq \lambda \leq 1 \tag{4.10}$$

and used in rescoring.

Since the correct (spoken) words are not available to the recognizer, all the words in the N-best hypotheses for the preceding utterances in a conversation side are considered in estimating the cache model. This has the beneficial effect that words appearing in multiple hypotheses are likely to be correct and hence get counted multiple times, while words that do not appear in many hypotheses are potentially due to recognition errors and are naturally discounted.

Usually, one would tune this value of N as well as the interpolation coefficient on some held-out set. However, in this case as well, we choose the value of N (100) and the interpolation weight (0.1) of the cache language model to minimize the perplexity of the test set. The results in Table 4.11 show that although interpolation with the cache model results in reduced perplexity, there is no reduction in WER from cache model use. Cache models have, however, been shown to reduce WER on other tasks.

Model (N-gram cutoffs)	Perplexity	WER
Back-off (B≥1, T≥2)	78.8	38.5%
Back-off + 1-gram Cache	75.2	38.9%
ME-Topic (B≥1, T≥2)	73.5	37.8%

Table 4.11: Perplexity and WER of a cache-based model and a maximum entropy model with topic constraints.

While it is a little surprising to see the WER increase slightly[11] with a cache model, this is perhaps explained by the relatively high error rates in the hypotheses from which the model is estimated. To support this argument, we compare the recognition output of the baseline trigram model and the cache-interpolated model of Table 4.11 in the following manner. Each recognition error is first marked as an *insertion*, *substitution* or *deletion* in the usual way. Next, each error token is marked as a *repeated* error if the word in question has undergone the same kind of error in the portion of the history used by the cache model. *e.g.*, if a word w in the reference transcription was deleted from the hypothesis for the first time in a conversation side at some position i and deleted again at position $j > i$, then j is a case of a repeated deletion of w, although i is not. Repeated insertions and substitutions of words are labeled in a similar manner. The word error rate is split into repeated error rate and non-repeated error rate (Table 4.12).

Model	Repeated	Non-Repeated	Overall
Trigram	35.8%	2.7%	38.5%
Cache-Based	36.3%	2.6%	38.9%
Difference	+0.6%	-0.2%	+0.4%

Table 4.12: Repeated errors vs. non-repeated errors.

Of the total error rate of 38.5% for the trigram model, 35.8% is accounted for by such repeated errors. For the cache-interpolated model, 36.3% of the total error rate of 38.9% is due to repeated errors. We find that compared to the baseline trigram model,

[11]But only marginally significantly, with a P-value of 0.034.

the cache-based model has about a 0.6% higher rate of repeated errors. Comparing this to the overall increase in error rate of only 0.4%, it seems reasonable to conclude that a small reduction of non-repeated errors is offset by an increase in the number of repeated errors, pointing to the detrimental effect of using a cache model at high error rates.

4.5 Broadcast News Experiments

With the help of hierarchical training and divide-and-conquer, we also construct a topic-dependent ME model for a large task - the Broadcast News task. The overall experimental results have been presented in Wu & Khudanpur (2002). We provide more detailed results in this section.

We use the whole training set of 130 million words contained in about 125,000 stories. The vocabulary of 64K words is chosen to cover all words in the acoustic training data of 70 hours and those words that occur 11 times or more in the language model training text. All out-of-vocabulary words that count 0.33% of total tokens in the training data are mapped to a special token $< unk >$. The acoustic model is a state-clustered cross-word triphone model with 39 dimension MFCC features. No speaker adaptation technique is used. The test set is the HUB-4 1996 eval set, containing about 570 utterances amounting to 22K words. In the experiments, word lattices for the utterances are first generated by the AT&T finite state machine (FSM) decoder (Mohri, Pereira & Riley, 2002), and then 100-best lists are created from lattices. The acoustic model scores and the language model scores of each of these 100-best hypotheses are obtained by aligning these hypotheses with the acoustic data using HTK. Finally, the 100-best lists are rescored using the new language models.

4.5.1 Performance of ME models

The baseline language model is a back-off trigram model with the bigram cut-off of 2 and the trigram-cut-off of 3. This is the trigram model with the lowest cut-offs that we can implement. The model contains about 64K unigrams, 3.5 million bigrams and 5.5 million trigrams. The perplexity of the baseline model is 174.3 and the corresponding word error rate is 34.6% [12] (Table 4.13). When computing the perplexity, 1.2% out-of-vocabulary words (OOVs) in the test data are mapped to the special token ($< unk >$). The topic-independent maximum entropy language model (with the same constraints as in the baseline model) is also built for comparison with the corresponding back-off model. As we have expected, this maximum entropy model almost duplicates the performance of the baseline back-off model (174.3 vs. 174.8 in perplexity and 34.6% vs. 34.5% in WER). The improvement from the topic-dependent model to the language model should be considered the result of adding topic dependencies.

To train the topic-dependent model, we obtain initial topic labels for documents in the training data from Florian & Yarowsky (1999), and then re-cluster the corpus into 100 topics by the K-means method mentioned in Section 4.2. In topic detection for the test utterances, we guess the best topic assignment using the same method as we did for the Switchboard experiments - assigning a fresh topic to each individual test utterance according the 10-best hypotheses of the current utterance and the four preceding ones (cf. Section 4.3.2). The topic-dependent model achieves a perplexity reduction of about 10% and a significant[13] WER reduction of about 0.6% absolute compared to the baseline trigram model. These results match those in the Switchboard experiments (a reduction of 7% for perplexity and a WER reduction of 0.7%) and show that the topic-dependent model is effective in different tasks.

Model	Perplexity	WER
Back-off 3-gram (B$\geq$2, T$\geq$3)	174.3	34.6%
ME 3-gram(B$\geq$2, T$\geq$3)	174.8	34.5%
ME Topic(B$\geq$2, T$\geq$3)	156.8	34.0%

Table 4.13: Perplexity and WER of language models for the BN corpus

[12]Although the perplexity of BN is higher than that of the Switchboard, its word error rate is relatively lower since the quality of speech is better that that of Switchboard.

[13]With a P-value of 10^{-4}.

4.5.2 Comparison with Interpolated Models

We also build interpolated topic-dependent models for comparison as we have done for Switchboard. We interpolate topic-dependent unigram, bigram and trigram models with the baseline topic-independent trigram model. Topic tags for the test utterances are the same as those for the ME model. The results are presented in Table 4.14. The first 4 rows show the results of the baseline back-off model and the interpolated models. The last 2 rows duplicate the results of ME models from Table 4.13 for comparison. It is apparent that the performance of the best interpolation models (topic specific trigram models interpolating with the baseline model) almost reach the performance of the ME model. However, the size of interpolation models in a number of parameters increases about eightfold compared to the baseline trigram model, whereas that of the ME model increases by only 3%. Therefore, we argue that the ME method is more practical than interpolation for very large tasks when the size of the model is huge.

Model (N-gram cutoff)	#Params	Ppl	WER
Back-Off 3-gram	9.1M	174.3	34.6%
BO + Topic 1-gram	+100×60K	167.6	34.5%
BO + Topic 2-gram	+100×400K	162.3	34.3%
BO + Topic 3-gram	+100×600K	156.1	34.1%
ME 3-gram	9.1M	174.8	34.5%
ME Topic	+250K	156.8	34.0%

Table 4.14: Comparison with 100 interpolated topic N-gram models.

4.6 Summary

We have constructed topic-dependent models using the maximum entropy method for a small task of telephone conversations (Switchboard) and a large task of Broadcast News. By incorporating a small number of topic constraints with N-grams, we have successfully reduced both the perplexity (7% for the Switchboard and 10% for the Broadcast News) and WER (0.7% absolute for the Switchboard and 0.6% absolute for the Broadcast News). We have studied the influence of different ways of topic assignment in recognition. The experimental results show that using N-best hypotheses causes little degradation (in perplexity and WER) and assigning topics at utterance level is slightly better than that at the conversation level. We have also compared the maximum entropy method with linear interpolation. The maximum entropy method is more efficient than linear interpolation in combining topic dependencies with N-grams. Finally, we have compared the topic-dependent models with cache-based ones and found that the former is better than the latter in reducing WER when the baseline is poor.

Chapter 5

Syntactic Language Models

In this chapter, we will show how to incorporate syntactic constraints with N-grams into a language model using the maximum entropy approach. We will briefly introduce, but will not delve deeply into, syntactic parsing because we mainly reuse the parser described by Chelba & Jelinek (1998). We will show experimental results with detailed analysis based on Switchboard and a subset of Broadcast News, and compare the maximum entropy model with the relevant baseline.

5.1 Syntactic Parsing

Many natural language applications including speech recognition may benefit from parsing sentences. A typical parsing problem is to map from a sentence to a tree that indicates the syntactic relations between words. For instance, the tree in Figure 5.1 represents the syntactic parse for the sentence *Bob saw the guy on the hill.* Multilevel information is embedded in the parse tree. Each subtree in the parse corresponds to a syntactic component in the sentence. The non-terminal immediately above each word in the sentence is the grammatical part-of-speech tag of that word. The upper level nodes describe words grouped as phrases and clauses. For example, the label *NN* on top of the word *guy* indicates that it is a *noun*, and the label *PP* on the root of the subtree for *on the hill* indicates that this word string constitutes a *prepositional* phrase. The label for the root of the complete parse tree is usually *S* indicating a

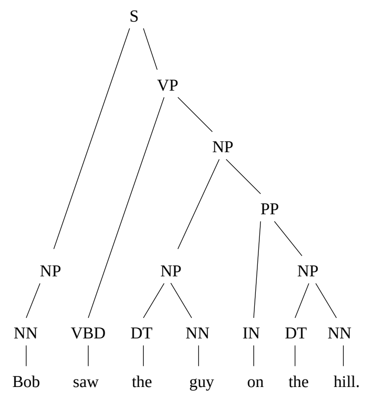

Figure 5.1: A sample parse tree.

complete sentence.

The parse tree also represents grammatical relations between words, phrases or clauses in a sentence. For example, the root of the tree has two children, NP and VP, indicating that a sentence can be generated by a noun phrase and a verb phrase by a generation "rule": S $\rightarrow$ NP VP.

Ambiguity exists in parsing; some sentences may have more than one meaningful parse. For instance, Figure 5.2 shows another candidate parse for the sentence *Bob saw the guy on the hill.* The propositional phrase *on the hill* may be considered to

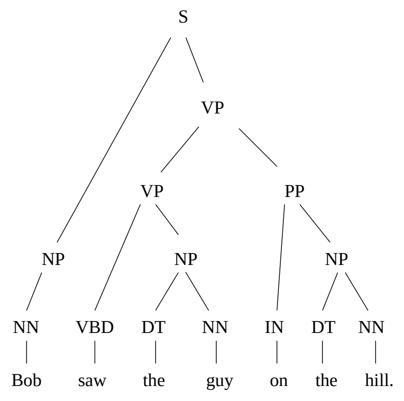

Figure 5.2: Another candidate parse.

modify *saw* rather than *the guy.*

5.2 Quantitative Measurement for Syntactic Dependencies

Although it is widely acknowledged that the syntactic structure of a sentence should be helpful in predicting words, many challenges must be met to integrate syntactic structure into a language model.

The structure of statistical language models and that of syntax are substantially different. Statistical language models are represented as multidimensional tables of conditional probabilities, while the syntactic structure of a sentence is embedded in trees. Therefore, a probabilistic model for syntactic information is required before it can be employed in statistical language modeling. Furthermore, parsers that generate syntactic structures are usually designed to work on correct sentences, and they must be modified to parse erroneous partial hypotheses in speech recognition. Finally, each sentence prefix may have many candidate parses instead of only a fixed one due to syntactic ambiguity, which has been shown in Section 5.1.

Initial efforts of overcoming the above challenges appear in Chelba & Jelinek (1998). All sentences in the training data are parsed by the left-to-right parser presented in Chelba (2000). Their parser generates a stack of candidate parse trees $\mathcal{S}_i = \{T_{ij} : 1 \leq j \leq J\}$ for a sentence prefix $w_1^{i-1} = w_1, w_2, ..., w_{i-1}$ at position i.

Figure 5.3 shows an example of one possible partial parse for the sentence prefix

the contract ended with a loss of 7 cent after $\cdots$.

Each subtree of the parse corresponds to a syntactic constituent in the sentence. For example, the triangle on the left corresponds to the noun phrase *the contract* and the one on the right to the verb phrase *ended with a loss of 7 cents.* Each non-terminal node in the parse tree T_{ij} has a syntactic head made up of a non-terminal label indicating the syntactic role of the corresponding constituent and an *exposed* lexical head-word[1]. This syntactic head dominates the subtree rooted at this node and may be regarded as the representative of the syntactic constituent spanned by the subtree. For instance, the word “ended” in the verb phrase *ended with a loss of*

[1]Traditional parsers do not assign the lexical head-word for each subtree.

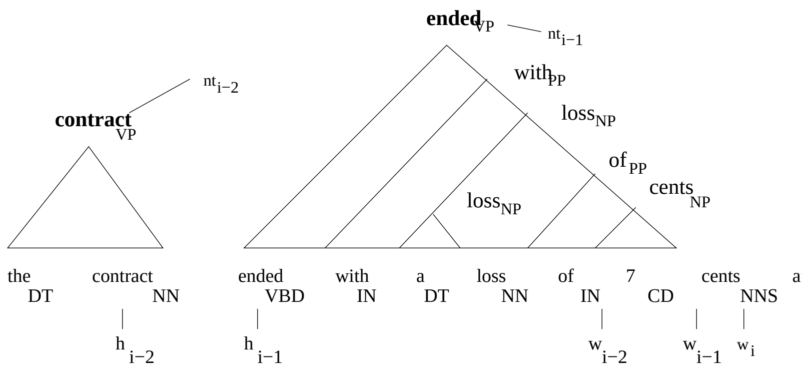

Figure 5.3: An example of a partial parse reproduced from Chelba & Jelinek (1998).

7 cents is assigned as the head-word of the entire phrase. In a parse tree spanning a complete sentence, the head-word at the root node is typically the main verb of the sentence. For a sentence prefix, the syntactic analysis is partial, and more than one constituent may await higher level parsing decisions as shown in Figure 5.3.

The exposed head-words preceding the word "after" in Figure 5.3, for instance, are "contract" and "ended." For a word w_i at position i, we label the exposed head-words from right to left as h_{i-1}, h_{i-2}, etc, dropping the subscript j for simplicity of notation. The reader should bear in mind that the preceding head-word exposed at position i will depend on the choice of the j^{th} candidate partial parse T_{ij}. Details of parse generation and identification of head-words may be found in Chelba & Jelinek (1998) and Chelba (2000).

The parser also internally computes a probability $p(w_i|w_1^{i-1}, T_{ij})$ for each possible word w_i given the j-th partial parse T_{ij}, and a likelihood function $\rho(w_1^{i-1}, T_{ij})$ for the j^{th} partial parse, according to

$$p(w_i|w_1^{i-1}) = \sum_{T_{ij} \in \mathcal{S}_i} p(w_i|w_1^{i-1}, T_{ij}) \cdot \rho(w_1^{i-1}, T_{ij}) \tag{5.1}$$

where

$$\rho(w_1^{i-1}, T_{ij}) \quad = \quad \frac{P(w_1^{i-1}, T_{ij})}{\sum_{T_{ij} \in \mathcal{S}_i} P(w_1^{i-1}, T_{ij})}. \tag{5.2}$$

We assume that the immediate history (w_{i-2}, w_{i-1}) and last two exposed syntactic heads (non-terminal labels and head-words) nt_{i-2}, h_{i-2} and nt_{i-1}, h_{i-1} of the partial parse T_{ij} carry most of the information useful in predicting w_i. Of course, some syntactic information will be lost under this assumption. However, it leads to a very important simplification:

$$P(w_i|w_1^{i-1}, T_{ij}) \approx p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}). \tag{5.3}$$

It is worth noting that the syntactic information is represented in the same form as N-grams by this representation! The main difference is that unlike w_{i-2}, w_{i-1}, the syntactic heads nt_{i-2}, h_{i-2} and nt_{i-1}, h_{i-1} are *hidden* variables, *i.e.*, unlike w_{i-2} and w_{i-1}, which are directly observable at any position i, the identities of nt_{i-2}, h_{i-2} and nt_{i-1}, h_{i-1}, which would be known only from the correct syntactic analysis of the utterance, must be treated as missing data.

The "correct" exposed heads are not known even for the training sentences. Therefore, we use the top J candidate partial parses as potentially correct parses with the appropriate probability to accumulate counts for each observed 7-tuple $(w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, w_i)$ in the training data as follows. If a particular position i has the trigram (w_{i-2}, w_{i-1}, w_i) and the j-th partial parse T_{ij} exposes heads (nt_{i-2}, h_{i-2}) and (nt_{i-1}, h_{i-1}) with probability ρ, then the 7-tuple $(w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, w_i)$ gets a fractional count[2] of ρ. The relative frequencies obtained in this manner may be set as constraints in the syntactic ME model.

During testing, each hypothesis is parsed by the parser and the ME model is invoked for each partial parse with its appropriate exposed head-words. The word probability is calculated by equations (5.1), (5.2) and (5.3), where the probability ρ of a partial parse is provided by the parsing model (5.2) of Chelba & Jelinek (1998).

5.3 Formulation of the Maximum Entropy Model

It is almost impossible to obtain enough statistics to estimate a separate conditional *pmf* for each history $(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$ in (5.3). For instance, there are 760K different histories for the Switchboard corpus of 2.1M words, and each

[2]Thus, the number of distinct training samples is much greater than the number of words in the training corpus, even though their (fractional) counts still add up to the number of words.

history occurs less than 3 times on average. We use the maximum entropy method to avoid this severe data sparseness problem. An outline of our initial efforts in this direction appears in Wu & Khudanpur (1999) and Khudanpur & Wu (2000).

The maximum entropy model for the dependency structure of (5.3) is done in much the same manner as was done for the topic-conditional model in Section 4.2.3. In particular, we impose the following marginal constraints on the model,

$$\sum_{h_{i-2},h_{i-1},nt_{i-2},nt_{i-1}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[w_{i-2},w_{i-1},w_i]}{\#[\text{training data}]}, \tag{5.4}$$

$$\sum_{w_{i-2},w_{i-1},nt_{i-2},nt_{i-1}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[h_{i-2},h_{i-1},w_i]}{\#[\text{training data}]}, \tag{5.5}$$

$$\sum_{w_{i-2},w_{i-1},h_{i-2},h_{i-1}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[nt_{i-2},nt_{i-1},w_i]}{\#[\text{training data}]} \tag{5.6}$$

where $\phi(x_i) = w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}$ is shorthand for the equivalence class of the history. The first kind of constraints involve regular trigram counts, and the last two involve what we call *head-word trigram* counts and *non-terminal trigram* counts. Of course, bigram constraints

$$\sum_{w_{i-2},h_{i-2},h_{i-1},nt_{i-2},nt_{i-1}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[w_{i-1},w_i]}{\#[\text{training data}]}, \tag{5.7}$$

$$\sum_{w_{i-2},w_{i-1},nt_{i-2},nt_{i-1},h_{i-2}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[h_{i-1},w_i]}{\#[\text{training data}]}, \tag{5.8}$$

$$\sum_{w_{i-2},w_{i-1},h_{i-2},h_{i-1},nt_{i-2}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[nt_{i-1},w_i]}{\#[\text{training data}]}, \tag{5.9}$$

and unigram constraints

$$\sum_{w_{i-2},w_{i-1},h_{i-2},h_{i-1},nt_{i-2},nt_{i-1}} p(w_i|\phi(x_i)) \cdot p(\phi(x_i)) = \frac{\#[w_i]}{\#[\text{training data}]} \tag{5.10}$$

are also imposed to build a smooth model. It should be noted that we approximate the probability $p(\phi(x))$ of histories by the empirical distribution $\hat{p}(\phi(x))$ of histories in the constraints above. To obtain reliable estimates of the marginal probabilities for constraining the model, Good-Turing discounting is applied to the low counts before they are used on the right-hand sides of the equations above. Further, bigram and trigram constraints based on very low counts are completely dropped. We then seek a model that satisfies the remaining constraints and has maximum entropy.

As we have shown, the maximum entropy model has the following exponential form

$$\begin{aligned} & p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}) \\ = \; & \frac{1}{z}\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \\ & \alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)} \cdot \alpha_{h_{i-2},h_{i-1},w_i}^{g(h_{i-2},h_{i-1},w_i)} \cdot \alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i)} \cdot \alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)} \end{aligned} \quad (5.11)$$

where $z = z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$ is a normalization constant. Again, that the first three factors in the numerator correspond to standard N-gram constraints. The next two represent head-word N-gram constraints, and the last two represent non-terminal N-gram constraints.

The parameters α of the model are again trained by the generalized hierarchical training method described in Chapter 3. The computational load in the training procedure is distributed over many computers by the parallel training method for further speed-up.

5.4 Tokenization Issues (Switchboard vs. Penn Treebank)

Compound words such as *it's, he's* and *you're* are treated as single words in the Switchboard transcriptions but are regarded as pairs of words in the Penn Treebank (Marcus, Santorini & Marcinkiewicz, 1993), because they correspond to more than one syntactic constituent in sentences. In our work, we convert all transcriptions to the Penn Treebank representation first before we train the syntactic language models [3]. To generate the language model score (log-probability) of a word in the Switchboard tokenization, we first compute the *partial* scores of each part (based on the Penn Treebank tokenization) of this compound word and then merge these partial scores

[3]N-gram models and topic-dependent models do not encounter this tokenization problem.

according to the following formula:

$$\log p(w_{swbd}) = \sum_k \log p(w_{tb}^k) \tag{5.12}$$

where w_{swbd} is a word in the Switchboard tokenization and w_{tb}^k is its k^{th} part (or the k^{th} word in the TreeBank tokenization). For example,

$$\log p(\textsf{he's}_{swbd}) = \log p(\textsf{he}_{tb}) + \log p(\textsf{'s}_{tb}). \tag{5.13}$$

5.5 Experimental Results in Switchboard

Most experiments for the syntactic language model are based on Switchboard. We parse about 2.1 million words of training data[4] by the parser in Chelba (2000). Because each sentence prefix may have more than one candidate partial parse, the total number of different training samples for the syntactic model is 2.5 million, which is greater than the number of words in the training data.

5.5.1 Head-Word Model

We first augment the N-gram constraints in the ME model with syntactic head-word dependencies. A model that incorporates head-word constraints (5.5) and (5.8) with N-gram constraint (5.4) and (5.7)

$$\begin{aligned} & P(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}) \\ = & \frac{\alpha_{w_i}^{g(w_i)} \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)} \alpha_{h_{i-2},h_{i-1},w_i}^{g(h_{i-2},h_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1})} \end{aligned} \tag{5.14}$$

was presented in Wu & Khudanpur (1999). The cut-offs for N-grams are the same as the baseline models (only singleton trigrams are dropped), and the cut-offs for head-word bigrams and trigrams are 2 and 1, respectively. For the remaining counts, we apply Good-Turing discounting[5] An alternative for Good-turing discounting is

[4]The same data as we used to trained the topic-dependent model.

[5]Since the counts r for syntactic constraints are functional, the orginal Good-Turing discounting method, which only applies to integer counts, cannot be directly used here. We modified the Good-turing discounting by gathering N_r, as stated in Section 1.2.2, from all counts $r < r \leq r$.

the Guassian prior smoothing described in Chen & Rosenfeld (2000), which can be regarded as an extention for the absolute discounting. A marginal improvement in perplexity based on N-gram models is achieved on small corpora. No word error rate improvement is reported in the paper. This method, however, requries a held-out set to tune the discounting values. Since we mainly focus on using syntactic features to improve language models, we do not compare the performance of these two kinds of discounting method here. We refer to the model (5.14) as the *head-word model.* Both the perplexity and the word error rate (Row 2 in Table 5.1) of this model reduce compared to those of the baseline trigram model (Row 1).

Then we slightly modify this model as follows: a head-word N-gram feature is deemed to be active only when the syntactic head-words h_{i-2}, h_{i-1} *do not* coincide with the two preceding words w_{i-2}, w_{i-1} in the history. If they coincide, no head-word trigram constraint is said to apply. We refer to the ME model with such constraints as the *complemented head-word model.* Table 5.1 compares the performance of these two models.

Model (#Params)	Perplexity	WER
3-gram (480K)	79.0	38.5%
3-gram + HW (+250K)	73.5	37.7%
3-gram + Comp HW(+160K)	74.5	37.7%

Table 5.1: The effect of dropping head-word constraints when they coincide with the two preceding words.

It is quite clear that even if head-word dependencies are considered only when the head-words lie beyond the range of trigrams, the resulting gain in perplexity is only slightly smaller than, and in WER almost the same as, when head-word dependencies are accounted for at all times. The complemented head-word model, on the other hand, has considerably fewer parameters. We will therefore use this notion of complemented head-word constraints in subsequent experiments.

5.5.2 Non-Terminal Model

We next augment the maximum entropy trigram model with non-terminal label constraints (5.6) and (5.9) instead of syntactic head-word constraints resulting in a *non-terminal model*

$$p(w_i|w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}) \tag{5.15}$$
$$= \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i)} \cdot \alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1})}.$$

The cut-offs for NT bigrams and NT trigrams are 2 and 5, respectively. By adding only 90K such non-terminal constraints, we obtain an improvement similar as that of head-word model (Table 5.2).

Language Model (# Params)	Perplexity	WER
3-gram (480K)	79.0	38.5%
3-gram+NT (+90K)	75.1	37.8%

Table 5.2: Dependence on non-terminals (NT).

5.5.3 Full Syntactic Model

Finally, we integrate both non-terminal label constraints and head-word constraints with N-grams in one syntactic model (5.11), and obtain more improvement than for the model using any of these two kinds of syntactic dependencies.

Language Model (# Params)	Perplexity	WER
3-gram (480K)	79.0	38.5%
3-gram+NT (+90K)	75.1	37.8%
3-gram+HW(Comp) (+160K)	74.5	37.7%
3-gram+NT+HW (+250K)	74.0	37.5%

Table 5.3: Dependence on components of the syntactic heads: non-terminals (NT), head-words (HW) and both.

It is worth noting that while the use of the head-word dependencies and the dependencies on the non-terminal tags together provides a 6.3% reduction in overall perplexity and a 1.0% absolute reduction in WER, a large fraction of this gain can be obtained by the gradual back-off effect provided by the non-terminal tags of the syntactic heads; the NT model alone reduces perplexity by 4.9% and WER by 0.7% absolute. To illustrate why this is a back-off effect even though there is no explicit notion of backing-off in ME models, we consider the example of Figure 5.3 again. If the head-word trigram *contract ended after* was not seen in the training corpus but the head-word bigram *ended after* was seen often, the HW model (Table 5.3) would not have an active feature $g(h_{i-2}, h_{i-1}, w_i)$ and the only syntactic influence would be through $g(h_{i-1}, w_i)$, which may be interpreted as *backing-off* to a head-word bigram. The NT model, on the other hand, may still activate $g(nt_{i-2}, nt_{i-1}, w_i)$, which is a coarser classification of the history. The syntactic model in this case would have *both* $g(nt_{i-2}, nt_{i-1}, w_i)$ and $g(h_{i-1}, w_i)$ active, which amounts to something in between a bigram and a trigram classification of the history.

5.6 Analysis of Recognition Performance

Our choice of syntactic constraints is clearly aimed at manipulating the probabilities of words that are displaced away from their predictors by some intervening syntactic components. Syntactic dependencies are thus expected to be most helpful in positions where the best predictors of the following word are not within N-gram range due to an intervening phrase or clause. We perform some analysis in this section to see if this hypothesis is true.

To see whether we indeed improve the model when syntactic head-words are different from N-grams, all histories in the test sentences are divided into two categories: those histories whose head-words (h_{i-2}, h_{i-1}) *coincide* with the two immediately preceding words (w_{i-2}, w_{i-1}) (*e.g.*, Figure 5.4) and those whose head-words do not (*e.g.*, Figure 5.3). About 77% of the histories in the Switchboard test set (and 68% in Broadcast News) belong to the former category, perhaps explaining the considerable power of the trigram model. The perplexity and WER for these two sets are counted

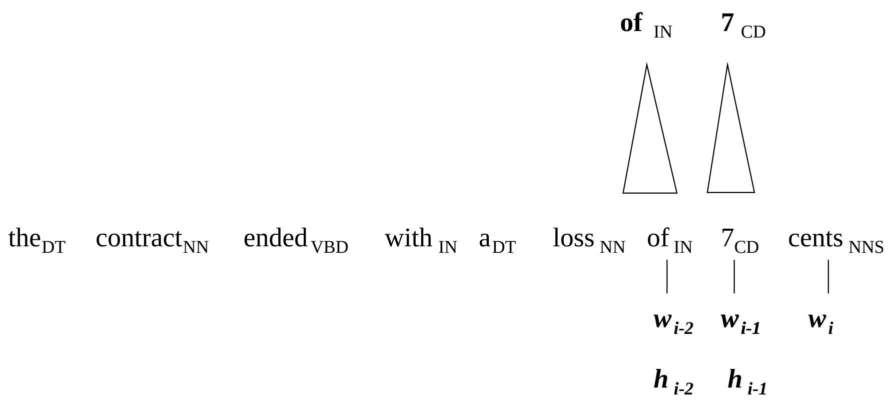

Figure 5.4: w_{i-2}, w_{i-1} and h_{i-2}, h_{i-1} are identical.

separately. Table 5.4 presents the breakdown results.

Language Model	Overall Performance		$h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$		$h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$	
	Ppl	WER	Ppl	WER	Ppl	WER
3-gram	79.0	38.5%	78.8	37.8%	79.7	40.3%
NT	75.1	37.8%	75.5	37.2%	73.8	39.4%
HW	74.5	37.7%	74.8	37.4%	73.6	38.8%
Both	74.0	37.5%	75.7	36.9%	70.6	38.9%

Table 5.4: Splitting results: $h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$ vs $h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$.

When $h_{i-1} \neq w_{i-1}$ or $h_{i-2} \neq w_{i-2}$, the reduction in perplexity and WER resulting from the head-word model is greater than the overall reduction (7.7% vs. 5.6% in perplexity and 1.5% vs. 0.8% in WER). This result is consistent with the intuition exemplified by Figure 5.3, and supports our claim that the syntactic head-words carry meaningful information that is not captured by N-grams. However, the NT model achieves only slightly more improvement in this case (0.9% in WER) than that when $h_{i-1} = w_{i-1}$ and $h_{i-2} = w_{i-2}$ (0.6% in WER). This suggests that non-terminal labels may help the model by providing some smoothing effect. Of course, for the syntactic model with both head-word constraints and non-terminal constraints, the improvement (11.4% in perplexity and 1.4% absolute in WER) in the case when $h_{i-1} \neq w_{i-1}$ or $h_{i-2} \neq w_{i-2}$ is more than the overall improvements, and this difference is mainly due to the head-word constraints.

To further study the role of syntactic heads (in particular non-terminal labels) in the syntactic language model, in the next section we compare the syntactic models with the class-based model with *part-of-speech* (POS) constraints.

5.7 Comparison with Related Methods

5.7.1 Syntactic Model vs. Class-Based Model

A question to ask after seeing the results of Table 5.4 is whether one could obtain a back-off effect similar to the non-terminal tags without resorting to syntactic analysis of the history $w_1, w_2, \ldots, w_{i-1}$. A substitute for performing the syntactic analysis and using the non-terminal tags of the exposed head-words is to simply work with the two preceding words w_{i-2}, w_{i-1} instead of the head-words, and their part-of-speech (POS) tags instead of non-terminal labels. An outline of our initial efforts in this direction appears in Wu & Khudanpur (2000a). We impose the following marginal constraints,

$$\sum_{w_{i-2},w_{i-1}} p(w_i|\phi(x_i)) \cdot \hat{p}(\phi(x_i)) = \frac{\#[pos_{i-2}, pos_{i-1}, w_i]}{\#[\text{training data}]}, \tag{5.16}$$

$$\sum_{w_{i-2},w_{i-1},pos_{i-2}} p(w_i|\phi(x_i)) \cdot \hat{p}(\phi(x_i)) = \frac{\#[pos_{i-1}, w_i]}{\#[\text{training data}]} \tag{5.17}$$

where $\phi(x_i) = w_{i-2}, w_{i-1}, pos_{i-2}, pos_{i-1}$ are imposed on the relative frequencies of the POS tags much as in (5.1), and the resulting ME model has the functional form

$$\begin{aligned} & p(w_i|w_{i-1}, w_{i-2}, pos_{i-2}, pos_{i-1}) \\ = & \frac{\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{pos_{i-1},w_i}^{g(pos_{i-1},w_i)} \cdot \alpha_{pos_{i-2},pos_{i-1},w_i}^{g(pos_{i-2},pos_{i-1},w_i)}}{z(w_{i-2}, w_{i-1}, pos_{i-2}, pos_{i-1})}. \end{aligned} \tag{5.18}$$

To estimate the ME model with POS N-gram constraints, we tag the training corpus automatically using a tagger developed by Ratnaparkhi (1996) and then follow the same steps as in training the model with non-terminal N-gram constraints. The results of evaluating this model are shown in Table 5.5, where the results of the trigram model and the model (5.15) with non-terminal N-gram constraints obtained by syntactic analysis are repeated for comparison.

Language Model	Overall Performance		h_{i-2}, h_{i-1} $= w_{i-2}, w_{i-1}$		h_{i-2}, h_{i-1} $\neq w_{i-2}, w_{i-1}$	
	Ppl	WER	Ppl	WER	Ppl	WER
3-gram	79.0	38.5%	78.8	37.8%	79.7	40.3%
3-gram+POS	75.9	38.0%	75.8	37.6%	76.1	39.2%
3-gram+NT	75.1	37.8%	75.5	37.2%	73.8	39.4%

Table 5.5: Conditioning on non-terminal (NT) vs part-of-speech (POS) tags: effect of syntactic analysis.

First, POS constraints are useful, but not as useful as non-terminal labels from the syntactic analysis; their overall gains are smaller both in perplexity and WER.

history Class	Found 3-grams	Not Found 3-grams
$h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$	63.2%	36.8%
$h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$	45.6%	54.4%

Table 5.6: Trigram coverage: $h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$ vs $h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$.

It may be noted in this context that when the syntactic heads are within trigram range, 63.2% of the trigrams in the test set have been observed in the training corpus as shown in Table 5.6. In cases when the syntactic heads are beyond trigram range, this coverage is only 45.6%. Thus, $h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$ corresponds to higher trigram coverage than $h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$. Therefore, it is not surprising to find in Table 5.5 that the gentler back-off effect provided by the POS constraints is much more effective in reducing WER when $h_{i-2}, h_{i-1} \neq w_{i-2}, w_{i-1}$ than when $h_{i-2}, h_{i-1} = w_{i-2}, w_{i-1}$ (1.1% vs. 0.2%). This reinforces the intuition about improvements from POS N-gram constraints.

We argue further that the non-terminal N-gram constraints operate in a similar manner, except that their contribution is linguistically more meaningful and therefore more effective in supplementing N-gram constraints. This is reinforced by the fact that when information about the lexical identity of the head-word is added on top of the non-terminal label, performance improves further, as seen in the last line of Table 5.5. The lexical identity in the POS case, w_{i-2}, w_{i-1}, is already present in the model and there is no further room for similar improvements.

5.7.2 Maximum Entropy vs. Linear Interpolation

Analogous to Section 4.4.1, it is of interest to compare our maximum entropy technique of combining syntactic and N-gram dependencies with more traditional interpolation techniques. Due to more than sheer coincidence, the experimental results reported so far are on the same language model training corpus, baseline recognizer and test set as used by Chelba and Jelinek (1999), who employed deleted interpolation instead of maximum entropy. Thus, such a comparison is easily made. In their work, the standard trigram model and trigram models based on the syntactic heads, *i.e.* the 5tuple $(h_{i-2}, nt_{i-2}, h_{i-1}, nt_{i-1}, w_i)$, where the non-terminal labels are used for a coarser classification of the history, are combined via deleted interpolation. A minor difference between the experimental conditions is that they rescore a firstpass lattice using an A* search algorithm, while we rescore the 100-best hypotheses from the first recognition pass. However, the results are still very comparable. We therefore simply reproduce their experimental results in Table 5.7 for comparison. It is worth noting that the maximum entropy technique slightly but consistently outperforms interpola-

Language Model	Perplexity	WER
Baseline Trigram	79.0	38.5%
Interpolated Syntactic	75.5	37.9%
Maximum Entropy	74.0	37.5%

Table 5.7: ME v/s interpolated syntactic models (cf. Chelba & Jelinek (1999)).

tion, as also reported in other work on maximum entropy models. However, the main argument for maximum entropy remains its ability to combine many diverse sources of statistical dependence in an elegantly unified manner.

5.8 Broadcast News Results

We also train syntactic models on a subset of Broadcast News. We choose about 14 million words of training data from the Broadcast News in 1996. These data, which account for about 10% of the overall training set in Broadcast News, are parsed by the left-to-right parser of Chelba (2000). Because each partial sentence may have many candidate parses, the number of distinct 7-tuples $(w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, w_i)$, which is about 60 million, is much more than the number of words (14 million) in the training set. We cluster those very low frequency histories into history equivalence classes and also merge the 7-tuples according to the history classes. This eventually results in about 14 million different history classes and 28 million different training tuples. It should be noted that even this size is already more than one order of magnitude larger than the size of the Switchboard corpus. We use the generalized hierarchical training method described in Chapter 3 to train syntactic models using these data. The major purpose of this work is to see whether we can process large corpora, which would be otherwise computationally intractable without the generalized hierarchical training methods. Another purpose is to prove that maximum entropy models with syntactic constraints can be widely used in different tasks.

5.8.1 Baselines

There are two baselines to which the syntactic model should be compared: trigram models trained on the 14M word subset on which we train the syntactic model and those trained on the entire Broadcast News corpus of 130M words. Table 5.8 gives the perplexity, measured based on original Broadcast News tokenization, and the word error rate of baseline models. All counts of less than 8 are replaced by their corresponding Good-Turing estimates. The cut-offs for language models are also indicated in the table.

The back-off trigram model trained on 14 million words and the corresponding maximum entropy model have almost the same perplexity and word error rate, which are worse than those of models trained on the whole corpus. Most degradation comes

Language Model, Cut-off #[Training data]	Perplexity	WER
BO 3-gram $B \geq 1, T \geq 2$ (14M)	214	35.3%
ME 3-gram $B \geq 1, T \geq 2$ (14M)	217	35.4%
BO 3-gram $B \geq 2, T \geq 3$ (130M)	175	34.6%
ME 3-gram $B \geq 2, T \geq 3$ (130M)	174	34.5%

Table 5.8: Baseline perplexities and WERs for BN.

from the lack of training data, whereas a (small) portion may come from the difference in tokenization (treebank vs original[6])

5.8.2 Syntactic Models

Next, we build two separate syntactic models using 14M training data, one with only head-word constraints and the other with only non-terminal constraints as we have done to Switchboard. With the help of syntactic head-word constraints, the perplexity and the word error rate of the head-word model reduce by 5.0% and 0.5%, respectively, compared to the baseline back-off model trained on the same data set (Table 5.9). The non-terminal model achieves a similar improvement as the head-word model.

Language Model (# of Parameters)	Perplexity	WER
BO 3-gram (2.3M)	214	35.3%
ME 3-gram (2.3M)	217	35.4%
3-gram+NT (+600K)	202	34.9%
3-gram+HW (+2M)	204	34.8%
3-gram+NT+HW(+2.6M)	199	34.6%

Table 5.9: Performance of syntactic models for BN.

Finally, we combine both head-word constraints and non-terminal constraints with

[6]A trigram in the treebank tokenization may be a bigram in the original text, because the compound word in the original text is split into several words in the treebank tokenization. Therefore, the trigram model built on the treebank tokenization has slightly weaker predicting power than the model built on the original one.

N-grams in one model. This maximum entropy model outperforms the one with only one type of syntactic constraint. Overall, the perplexity is reduced by 7% and the word error rate is reduced by 0.7% absolute (Row 5 in Table 5.9) compared to the trigram model trained on the same data set. This improvement in Broadcast News is compatible to the results in Switchboard. With the use of the interpolated syntactic model as reported in Chelba (2000) instead of the ME model, the WER reduction on this test set is 0.5%.

5.8.3 Limitations of Computer Speed and Memory Space

More training data and larger language models will result in lower perplexity and also lower word error rate than using less data and smaller models. A question that readers may ask is why it is not viable to use all data to train the syntactic model. The major reason is the limitation of space and speed of computers. Parsing these 14M Broadcast News data already takes roughly 1500 CPU-hours. The parsing results (*i.e.*, syntactic heads) need more than 3.5GB of disk space for storage, even in compressed files. The pre-processing involved in extracting constraints and collecting training tuples from the parsing results of these 14M words already challenges the computational power of current computers. It should be noted that the complexity of training a back-off or interpolated syntactic language model is roughly the same as the pre-processing, since most of the time spent on training either back-off or interpolated models is that of collecting counts from training tuples. This time of tens of hours for Broadcast New is definitely not negligible.

Table 5.10 illustrate that from Switchboard to the Broadcast News subset, the model size and training data increase about sevenfold, but the training time and space requirement increase 80 fold and 10 fold, respectively. We may expect an increase of tens of folds in both the space requirement and the running time if we use the whole corpus. The memory requirement would be at least 10GB[7] and the training time would be more than 200 CPU-days per iteration. Consequently, it is currently impossible to train the syntactic model using the whole Broadcast News data.

[7]Unix for 32-bit machines can handle at most 4GB for memory.

Corpus	Data	Model Size	Memory	CPU-Hours
SWBD	2.1M	800K	100M	6
BN(10%)	14M	5.5M	1G	480
BN	*130M*	*20M*	*6-8G*	*>4800*

Table 5.10: Memory requirement and training time per iteration for SWBD and BN. The data in the last row are estimated numbers.

Instead of using more data and building a larger language model, we want to improve the performance of the language model by exploiting from the current training data other meaningful constraints, such as topics constraints in addition to syntactic constraints. The size of the language model can thus be kept to manageable size. This work will be described in Chapter 6.

5.9 Summary

We have shown that the contribution of syntactic heads in statistical language models is fairly complementary to N-gram statistics: the model improves significantly when the syntactic heads are beyond N-gram range. A modest but statistically significant[8] reduction in word error rate on the Switchboard corpus and the subset of the Broadcast News corpus has been achieved by using language models with statistical dependencies on syntactic heads. The experimental results in both tasks agree with each other, showing that the syntactic model can be widely applied to different tasks.

We have also shown that a large portion of this gain comes from using the non-terminal labels alone, with a smaller fraction coming from the lexical identity of the head-word. However, the gain from non-terminal labels is different from simply using POS tags of preceding words to obtain smoothing effects in case of unseen N-grams. This points to the value of the statistical analysis of the history.

We also compared the maximum entropy technique with the interpolation method in building syntactic language models. The experimental results showed that the former one provides a better means of incorporating syntactic dependencies in a

[8]With P-value of 10^{-5} and 10^{-3} for Switchboard and Broadcast News respectively.

language model than the latter one.

Chapter 6

Composite Language Model with Both Topic Dependencies and Syntactic Dependencies

In this chapter, we will show how to incorporate topic as well as syntactic constraints with N-grams into one unified language model. The initial attempts of this work on Switchboard have been introduced in Wu & Khudanpur (1999) and Khudanpur & Wu (2000). The model introduced in these two papers combines syntactic head word constraints and topic constraints without considering non-terminals. Here, we first expand the model by adding non-terminal constraints. Further, we apply this method to a much larger Broadcast News task. The experimental results in this chapter contain considerable more details than those in the papers.

6.1 Introduction

The major motivation to use the maximum entropy method is to combine different sources of information in one language model. We have explored topic and syntactic constraints for language models. Now, we want to estimate the probability of the current word w_i conditioned on previous words w_{i-2}, w_{i-1}, syntactic heads $nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}$ and the topic tag t_i, *i.e.*, $p(w_i|w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, t_i)$.

This conditional probability looks like an "8-gram" probability. Obviously, it is impractical to collect enough training data to estimate such a model. Most of the 8-tuples $w_{i-2}, w_{i-1}, w_i, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, t_i$ will not occur more than once in the training data. Although the data sparseness problem can be avoided by either interpolation or back-off, the optimal interpolation and back-off strategy is hard to find. There are $7! = 5040$ possible back-off strategies[1]. The interpolation strategies are more complicated. Of course, it is not practical to try all these candidate models. The maximum entropy method, on the other hand, avoids this data sparseness problem by estimating the joint probability from "reliable" marginal distributions. We simply need to find these "reliable" marginal constraints and choose the maximum entropy distribution satisfying these constraints.

6.2 Formulation

To build a language model with N-gram, syntactic and topic constraints, we look for the maximum entropy model satisfying the following marginal constraints:

$$\begin{aligned}
\sum_{nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, t_i} p(w_i|\phi(x_i)) \cdot \hat{p}(\phi(x_i)) &= \frac{\#[w_{i-2}, w_{i-1}, w_i]}{\#[\text{training data}]}, \\
\sum_{w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, t_i} p(w_i|\phi(x_i) \cdot \hat{p}(\phi(x_i)) &= \frac{\#[h_{i-2}, h_{i-1}, w_i]}{\#[\text{training data}]}, \\
\sum_{w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, t_i} p(w_i|\phi(x_i) \cdot \hat{p}(\phi(x_i)) &= \frac{\#[nt_{i-2}, nt_{i-1}, w_i]}{\#[\text{training data}]}, \\
\sum_{w_{i-2}, w_{i-1}, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}} p(w_i|\phi(x_i) \cdot \hat{p}(\phi(x_i)) &= \frac{\#[t_i, w_i]}{\#[\text{training data}]}
\end{aligned}$$

where $\phi(x_i) = w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, t_i$ is the history equivalence class. It should be noted that the first kind involves regular N-gram counts, the second and the third involve what we call *head-word N-gram* counts and *non-terminal N-gram* counts, and the last one constrains unigram frequencies within specific topics. Lower

[1]We can drop any symbol from the history 7-tuple $(w_{i-2}, w_{i-1}, w_i, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, t_i)$ and get 7 different 7-gram models to be backed-off for the 8-gram model $p(w_i|w_{i-2}, w_{i-1}, w_i, nt_{i-2}, nt_{i-1}, h_{i-2}, h_{i-1}, t_i)$. Similarly, we have six candidate 6-gram models for each of these seven-gram models, and so on. Totally, there are 7! possible combinations.

order (unigram and bigram) constraints are set similarly. The combination of these constraints results in an exponential model of the form

$$\begin{aligned} & p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-1}, nt_{i-2}, t_i) \\ = \; & \frac{1}{z}\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)} \cdot \alpha_{h_{i-2},h_{i-1},w_i}^{g(h_{i-2},h_{i-1},w_i)} \\ & \alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i)} \cdot \alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)}, \end{aligned} \tag{6.1}$$

where $z = z(w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-1}, nt_{i-2}, t_i)$ is a normalization constant. Comparing the composite model (6.1) with the syntactic model (5.11), one can see that the only difference between them lies in the last topic-dependent feature in the composite model. This shows the flexibility of the maximum entropy method in adding constraints from a new information source to an existing model. The training of this composite model has been described in Chapter 3.

6.3 Overall Experimental Results for Switchboard

The perplexities and word error rates of ME models with topic and/or syntactic dependencies are shown in Table 6.1. Once again, we list the baseline perplexity of 79 and WER of 38.5% in the first row in the table. The performance of the maximum entropy model with any of the topic constraints, head word constraints and non-terminal constraints alone follows in Row 2, 3 and 4, respectively. As we have shown in the previous chapters, these three models all outperform the baseline model. Next, models with two kinds of non-local constraints are built and evaluated (in Row 5, 6 and 7, respectively). They achieve lower perplexity and word error rate than the models with one kind of non-local dependency. For example, the model with both topic and head word constraints outperforms both the topic-dependent model and the head-word dependent model, and the model with topic and NT constraints outperforms both the topic-dependent model and the NT model. It should be noted that the results in Row 7 are what have been reported in Wu & Khudanpur (1999) and Khudanpur & Wu (2000). Finally, the composite model combining all three types of non-local dependencies is built (Row 8) and it outperforms other models using any one or two types of non-local constraints. Overall, the model with topic, head word and non-terminal dependencies achieves a perplexity reduction by 13% and a WER reduction by 1.5% absolute compared to the results of the trigram model (Row 1).

No.	Model Constraints	Perplexity	WER
1	3-gram	79.0	38.5%
2	3-gram + NT	75.1	37.8%
3	3-gram +HW	74.5	37.7%
4	3-gram +Topic	73.5	37.8%
5	3-gram +HW + NT	74.0	37.5%
6	3-gram +Topic + NT	70.3	37.3%
7	3-gram +Topic + HW	69.9	37.2%
8	3-gram +Topic + HW+NT	67.9	37.0%

Table 6.1: Performance of ME language models with N-gram, head-word (HW), non-terminal label (NT) and topic dependencies.

It is worth mentioning that gains of the composite model from both syntactic and topic dependencies are almost additive; the topic dependencies (topic) and the syntactic dependencies (NT+HW) help to reduce the WER by 0.7% and 1.0%, respectively (Row 4 and 5), and they together reduce the WER by 1.5%. Each of the two types of syntactic dependencies, NT or HW, is also individually complementary to the topic dependencies (topic); the non-terminal dependencies (NT) and the topic dependencies together reduce the WER by 1.2% while each reducing the WER by 0.7% and 0.7%, respectively; the head word dependencies (HW) and the topic dependencies together reduce the WER by 1.3% while each reducing the WER by 0.8% and 0.7%, respectively. It is easy to check that the perplexity reduction from both information sources is almost additive.

Therefore, we conclude that the topics and the syntactic information are complementary. The results also show the benefits of integrating various sources of information under the ME framework for improving language modeling.

6.4 Analysis of Performance for Switchboard

We have shown in Chapter 4 that the topic dependencies in the topic model benefit content-bearing words more than stop words. We have also shown in Chapter 5 that the head word and non-terminal constraints in the syntactic model compensate for the N-gram model when the syntactically meaningful information is beyond N-gram range. The composite language model integrates both topic dependencies and syntactic dependencies and is thus expected to inherit these two properties from both the topic-dependent model and the syntactic-dependent model.

To see if this assumption is true, we analyze the performance of the composite model in the same way as we have done for the topic-dependent model and the syntactic model. First, we divide the vocabulary into two sets, content-bearing words and stop words, and compare the performance of the composite model on these two subsets. Next, we divide the histories into two sets according to whether the syntactic heads fall within trigram range or not and compare the performance. Finally, we focus only on the subset of test data in which syntactic heads locate outside trigram range in the history and the future word is a content word, and we expect the improvement of the composite model to be the most significant on this subset.

6.4.1 Role of Topic Dependencies

Table 6.2 shows a breakdown of the results for content words and stop words for the composite model. The results of the baseline trigram model, the topic model and the syntactic model are also shown for comparison. It should be noted that the trigram model and syntactic model are topic independent while the topic model and composite model are topic dependent.

It is apparent that, similar to the topic model, the composite model helps to reduce the word error rate on topic-dependent words at a higher rate than the overall improvement (2.1% vs. 1.5%). Therefore, we argue that the performance gains (of the composite model) on content words mainly come from topic information. The word error rate of stop words also reduces when topic-sensitive models are used,

Language Model	Content Words		Stop Words		Overall	
	Ppl	WER	Ppl	WER	Ppl	WER
Trigram	8941	42.2%	36.4	37.6%	79.0	38.5%
Topic	3923	40.8%	37.1	37.0%	73.5	37.8%
Syntactic	4840	41.9%	36.6	36.3%	74.0	37.5%
ME Composite	3773	30.1%	35.5	36.2%	69.2	37.0%

Table 6.2: Perplexity and WER for content words and stop words.

because it is influenced by the word error rate of content words in the vicinity of stop words. Therefore, we conclude that the composite model inherits this property of the topic-dependent model.

It is of interest that most of the word error rate reduction by the (topic-independent) syntactic model comes from stop words rather than from content words. This may be caused by the lack of syntactic constraints for content words. Most content words w_i are low frequency words and even their low counts are distributed among many distinct training tuples $h^j_{i-2}, h^j_{i-1}, nt^j_{i-2}, nt^j_{i-1}, w_i$ for $j = 1, \cdots, J$ where J is the average number of candidate parses for each sentence prefix. Therefore, many content words have no syntactic features activated in the model[2]. Stop words, however, are high frequency words and do not suffer from this problem. Table 6.3 shows the percentages of content words and stop words with any type of syntactic constraints.

Constraints	Cont. Wds	Stop Wds.
HW 2-gram	52.8%	99.5%
HW 3-gram	33.0%	79.5%
NT 2-gram	56.7%	98.6%
NT 3-gram	44.2%	93.9%

Table 6.3: Percentage of content/stopwords with syntactic constraints.

It is apparent that almost all stopwords have some types of syntactic features activated, while a far fewer percentage of content words have the same type of features

[2]After we cut low counts off to keep the model in a manageable size.

activated. This may explain why syntactic models help to reduce WER more on stop words than on content words.

6.4.2 Role of Syntactic Dependencies

Table 6.4 gives the breakdown perplexity and word error rate across different history classes: those histories whose head words (h_{i-2}, h_{i-1}) *coincide* with the two immediately preceding words (w_{i-2}, w_{i-1}) and those whose head-words do not. Besides the splitting results for the composite model, we also list the results for the baseline model and the syntactic model copied from Chapter 5 and provide the splitting word error rate for the topic model for comparison. It should be noted that the trigram model and the topic-dependent model do not use syntactic constraints, whereas the syntactic model and the composite model do. As expected, all long-range dependencies, especially syntactic heads, help when *head words* $\neq$ *trigram*. The WER reduction for the syntactic model and the composite model in this case is 1.5% and 2.3%, respectively, which is much higher than the overall improvements. This supports our claim that the improvement in the cases when the predicting information is beyond trigram range mainly come from the syntactic heads, indicating that the composite model inherits this property of syntactic models.

Language Model	$h_{i-1} = w_{i-1}$ $h_{i-2} = w_{i-2}$		$h_{i-1} \neq w_{i-1}$ *or* $h_{i-2} \neq w_{i-2}$		Overall	
	Ppl	WER	Ppl	WER	Ppl	WER
Trigram	78.8	37.8%	79.7	40.3%	79.0	38.5%
Topic	73.0	37.3%	74.4	39.1%	73.5	37.8%
Syntactic	73.1	36.9%	74.2	38.8%	73.5	37.7%
Composite	69.1	36.7%	69.2	38.0%	69.2	37.0%

Table 6.4: Performance improvement on different history classes.

We are surprised to see that the topic dependencies also help more in the case when syntactic heads are non-local (1.2% WER reduction). We argue that this improvement is caused by the relative low trigram coverage when the syntactic predicting information is beyond the local range and the relative poor performance of the trigram model (cf. Table 6.7 for the trigram coverage in these cases). Any extra information will benefit the model in this case.

6.4.3 Four-Way Analysis

Finally, we further split the test data four ways according to the combinations of history classes and word classes. The proportions of different history-word sets are shown in Table 6.5. The composite language model should have the most advantage when the syntactic information is beyond the trigram range and the following word is a content word, and it is expected to correct relatively more errors in this case than in the other cases.

	$h_{i-1} = w_{i-1}$ and $h_{i-2} = w_{i-2}$.	$h_{i-1} \neq w_{i-1}$ *or* $h_{i-2} \neq w_{i-2}$	All Histories
Stop Words	58.1%	22.1%	80.1%
Content Words	14.3%	5.6%	19.9%
All Words	72.3%	27.7%	100%

Table 6.5: Proportion of test data of different word classes and/or history classes.

We test the performance of the language models on each set in the same way as in the previous experiments. The breakdown of the WERs is shown in Table 6.6.

	head words = trigram		*head words ≠ trigram*	
Language Model	Stop Wds	Content Wds	Stop Wds	Content Wds
ME Trigram	37.3%	40.0%	38.3%	48.6%
ME Topic	36.8%	39.0%	37.3%	46.1%
ME Syntactic	36.3%	39.4%	36.9%	47.2%
ME Composite	36.0%	38.1%	36.4%	46.0%

Table 6.6: WER based on different history classes and word classes.

- First, we find that the WER is relatively high on head words and/or head words beyond trigram range. This is due to the low coverage of trigram in these cases (Table 6.7).

	$h_{i-1} = w_{i-1}$ and $h_{i-2} = w_{i-2}$.	$h_{i-1} \neq w_{i-1}$ *or* $h_{i-2} \neq w_{i-2}$	Average
Stop Words	71.3%	50.5%	63.9%
Content Words	26.6%	11.9%	22.8%
Average	63.2%	45.6%	57.0%

Table 6.7: Coverage of trigrams of different word classes and/or history classes.

- Next, when $h_{i-1} = w_{i-1}$ and $h_{i-2} = w_{i-2}$, topic dependence and/or syntactic dependence bring some marginal improvements in WER on stop words. This result is reasonable since our models are not designed to improve the WER in this case. However, the WER on content words reduces (by 1.0%) more than average when topic dependencies are applied as we have expected. When $h_{i-1} \neq w_{i-1}$ or $h_{i-2} \neq w_{i-2}$, both syntactic dependence and topic dependence are helpful. The syntactic model improves the WER evenly on both content and stop words, whereas the topic model is effective mainly on content words.

- Finally, when $h_{i-1} \neq w_{i-1}$ or $h_{i-2} \neq w_{i-2}$ and the predicted word is a content word, the reduction of WER by using the composite model is considerable (2.6%), since both kinds of long-range dependencies are active in this case. The composite language model inherits this advantage from both the topic-dependent model and the syntactic-dependent model.

6.5 Broadcast News Results

With the help of both the divide-and-conquer and the generalized hierarchical training described in Chapter 3, we also build the ME composite language model with both topic and syntactic (HW+NT) constraints for Broadcast News. The training data is still the 14M-word subset we used for training the syntactic model.

Unlike the topic model, which is trained on 100 topics using 130M words, the composite model is trained on 60 topics instead. One reason for reclustering the training data into fewer topics is to avoid data sparseness problems. Another consideration is to save the model estimation time[3]. We keep those 60 topics that have the largest amount of data among the original 100 topics, and reassigned the documents in the remaining 40 topics to one of these 60 clusters. Next, we apply the K-mean clustering algorithm described in Section 4.2 to recluster the training data. Sixty-five thousand topic-dependent unigram features are selected by the criteria described in Section 4.2.

We first build a new baseline trigram model with the trigram cut-off of 3. We set higher cut-offs for the experiments in the section, because we need to limit the composite model to a manageable size. The perplexity of this trigram model increases slightly compared to the model with the trigram cut-off of 2 built in Section 5.8. However, the word error rate remains almost the same (Table 6.8). Next, we augment the trigram model by adding either topic-dependent features or syntactic features individually. Both the perplexity and the word error rate of the topic-dependent model reduce by 11.5% and 0.7%, respectively, compared to the corresponding trigram model, whereas those of the syntactic model reduce by 7.4% and 0.7%, respectively. It is worth noting that the cut-off for head word trigrams is 2.5 here instead of 2 as we used in Section 5.8.

No.	Model Constraints	Perplexity	WER
1	BO 3-gram (14M)	216	35.3%
2	ME 3-gram (14M)	218	35.4%
3	3-gram + Topic (14M)	191	34.6%
4	3-gram + HW + NT (14M)	200	34.6%
5	3-gram + Topic + NW + NT(14M)	177	34.1%
6	BO 3-gram(130M)	174	34.6%

Table 6.8: Performance of ME language models with N-gram, syntactic and topic dependencies.

Finally, the model with both topic constraints and syntactic constraints is built and evaluated (Row 5 in Table 6.8). It achieves almost additive improvement from topic and syntactic constraints (compared to the baseline trigram model) - as either the topic or the syntactic constraints help to reduce the word error rate by 0.7%,

[3]The training time for the composite model with 60 topics is already about 3 times as much as that to train the syntactic model using the same amount of data.

they together bring a word error rate reduction of 1.2% (compared to the trigram model trained on the same data). These results agree with what we have obtained in Switchboard.

Overall, the perplexity of the composite model reduces by 18.0% to 177, and WER reduces to 34.1%, which is even lower than the word error rate (34.6%) of the trigram model trained on the larger 130M-word Broadcast News corpus. Furthermore, this improvement in WER is significant (with a P-value of 10^{-4}).

6.6 Summary

The maximum entropy method has the advantage of incorporating different sources of information together into one unified model. We have constructed composite language models integrating both topic and syntactic dependencies for the Switchboard and the Broadcast News tasks using the maximum entropy method. These models outperform models with either of these two kinds of non-local dependencies above. Overall, the word error rate reduces by 1.5% in Switchboard and 1.2% in a subset of Broadcast News. It is worth mentioning that the word error rate of the composite model trained by 14M Broadcast News data is even lower than that of the trigram model trained by 130M BN data.

The gain from the topic dependencies and the syntactic dependencies is almost additive, showing that these two different information source are close to complementary. It is also shown, from the experiments in Switchboard, that the composite model inherits the properties of both the topic-dependent and the syntactic dependent models: it decreases the WER on content words more than on stop words, and it benefits more than does the overall case, where syntactic structure information is beyond the local range.

Chapter 7

Improving the Computation when Using ME Models

We have discussed how to simplify the computation in training maximum entropy models in Chapter 3. There is another problem that may prevent one from using ME models in a real speech recognition system: the amount of computation required to compute the language model probabilities when recognition. In this chapter, we will show how to simplify the computation in using ME models by three techniques: pre-normalization, approximation and history-caching.

The straightforward way of calculating the probability $p(y|x)$ for any $\langle x, y\rangle$ pair using an ME model is via the following formula

$$\begin{aligned} p(y|x) &= \frac{1}{z(x)} \prod_{k=1}^{K} \alpha_k^{g_k(x,y;A_k)}, \\ z(x) &= \sum_{y \in V} \prod_{k=1}^{K} \alpha_k^{g_k(x,y;A_k)} \end{aligned}$$

where g_k is the k^{th} feature in the feature set and α_k is its parameter. The computation for the numerator is simple. It requires a table look-up and a few multiplications of α's. However, computing the denominator z takes time, since z is the sum of hundreds or even thousands of terms[1]. Since histories in the test data cannot be organized

[1] Unigram-caching is applied; otherwise, computing z needs a summation over all words in the vocabulary.

hierarchically as they do in the training data, the speed-up methods introduced in Chapter 3 are not applicable. It is also not practical to pre-compute the normalization factor z for all histories since, as we have shown, the size of the history set is huge. As we will show later, it is also not necessary to do so.

We want to make ME models easy to use. We can reduce the computation in ME models in the following ways. For N-gram models, we pre-normalize them and convert them to the ARPA back-off format, a popular language model format that will be introduced later. N-gram models can thus be used as simply as back-off models. For topic models, we approximate their denominators z so that they can be estimated without much computation in speech recognition. For the rest of the models, we additionally apply some caching techniques to save the computation.

7.1 Converting ME Models to ARPA Format

ARPA format is the most popular language model format for back-off models in the speech recognition world. It is accepted by many speech recognition decoders such as HTK, and can be converted to the LM format of the popular AT&T finite state machine (FSM) decoder. ME models can be used in these recognizers so long as they can be converted to the ARPA format. In this section, we show how to transform ME models from their exponential form to the ARPA (back-off) form.

7.1.1 ARPA Format for Back-off Models

First, we briefly introduce how a back-off model is stored in the ARPA format. If $f(w_N|w_1,\cdots,w_{N-1})$ is the relative frequency after some smoothing and T_N be a pre-set cut-off for N-grams, then the back-off N-gram model can be written as

$$p(w_N|w_1,\cdots,w_{N-1}) = \begin{cases} f(w_N|w_1,\cdots,w_{N-1}) & \text{if } \#[w_1,\cdots,w_N] \geq T_N, \\ \mathsf{bow}(w_1,\cdots,w_{N-1})p(w_N|w_2,\cdots,w_{N-1}), & \text{otherwise} , \end{cases} \tag{7.1}$$

where the back-off weight bow[2] is the remaining probability mass for the all w_N whose $\#[w_1, \cdots, w_N] < T_N$. ARPA back-off models store all N-grams with their parameter values f and bow in one ASCII file. Table 7.1 shows the ARPA file format for back-off N-gram models (7.1).

\data\		
ngram	1=	number-of-unigrams
ngram	2=	number-of-bigrams
	⋮	
ngram	N=	number of N-grams
\1-grams:		
$f(w)$	w	[bow(w)]
	⋮	
\2-grams:		
$f(w_2 \mid w_1)$	w_1, w_2	[bow($w_1 w_2$)]
	⋮	
\N-grams:		
$f(w_N \mid w_1, ..., w_{N-1})$	$w_1, ..., w_N$	
	⋮	
\end\		

Table 7.1: ARPA format for back-off models.

The file is comprised of three segments: the header, the body and the tail. The header starts with the keyword "\data\" and lists the number of n-grams for $n = 1, \cdots, N$. The body of the model file consists of n-grams (one per line) grouped by length n. Each group begins with the keyword "\n-gram:", where n is the length of the n-grams that follow. Each line has either two or three columns: the logarithm (base 10) of the n-gram probability, the n-word tuple in ASCII and the logarithm

[2]Actually back-off weight bow's are not independent - they can be derived from f's.

(base 10) of its back-off weight. The third column can be omitted if the back-off weight is zero[3]. The tail of the file has only one line in which the keyword "\end\" concludes the model.

If an N-gram $(w_1, w_2, \cdots, w_N)$ is found in the model, then $p(w_N|w_1, \cdots, w_{N-1})$ is directly obtained by a table look-up, otherwise $p(w_N|w_1, \cdots, w_{N-1})$ is obtained by $\mathsf{bow}(w_1, \cdots, w_{N-1}) \cdot p(w_N|w_2, \cdots, w_{N-1})$ where $p(w_N|w_2, \cdots, w_{N-1})$ is computed recursively in the same manner.

7.1.2 Mapping the ME N-gram Model Parameters to ARPA Back-off Model Parameters

In a (recursive) back-off model $p(w_N|w_1, \cdots, w_{N-1})$, the relative frequency $f(w_N|w_1, \cdots, w_{N-1})$ is a scale function for each word *i.e.,* w_N given the history $w_1, \cdots, w_{N-1}$. If the history $w_1, \cdots, w_{N-1}$ is in the model (i.e. has been seen in the training data), the probability p is obtained directly from this scale function f since it is already normalized. If the history is not in the model, then the lower order scale function $f(w_N|w_2, \cdots, w_{N-1})$ is used instead, but the total probability mass becomes $\mathsf{bow}(w_2, \cdots, w_{N-1})$ instead of 1 now. Both $f(w_N|w_2, \cdots, w_{N-1})$ and $\mathsf{bow}(w_2, \cdots, w_{N-1})$ are available during the training procedure.

In the corresponding ME model

$$p(w_N|w_1, \cdots, w_{N-1}) = \frac{\prod_{i=1}^{N} \alpha_{w_{N-i+1}, \cdots, w_N}^{g(w_{N-i+1}, \cdots, w_N)}}{z(w_1, \cdots, w_{N-1})}, \tag{7.2}$$

the numerator is the scale function and plays a role similar to that of f in the back-off model. However, the total mass z is not explicitly given by the model parameters α. Nevertheless, in the training procedure for ME models, z for all observed histories have already been computed. For the enormous number of unseen histories, their normalization factors can also be computed indirectly during the last iteration of training. Actually, if $w_1, \cdots, w_{N-1}$ is not seen in the training data, the N-gram

[3] bow is zero for the highest order N-gram. It may also be zero for some lower order n-grams if they are not a prefix of longer n-grams.

feature function $g(w_1, \cdots, w_{N-1}, w_N)$ can never be activated for any w_N, therefore,

$$z(w_1, \cdots, w_{N-1}) = \sum_{w_N \in V} \prod_{i=1}^{N-1} \alpha_{w_{N-i+1}, \cdots, w_N}^{g(w_{N-i+1}, \cdots, w_N)}$$

is independent of w_1, or, in other words, it is a function of the history equivalence class $w_2, \cdots, w_{N-1}$, i.e, $z = z(w_2, \cdots, w_{N-1})$. Further, $z(w_2, \cdots, w_{N-1})$ is available during the hierarchical training if $w_2, \cdots, w_{N-1}$ is an observed history class; otherwise z is obtained from lower order history classes $z(w_3, \cdots, w_{N-1})$ etc.

We want to write the ME N-gram model in the form of the back-off model, *i.e.,*

$$p_{ME}(w_N | w_1, \cdots, w_{N-1}) = \mathsf{bow}_{ME}(w_1, \cdots, w_{N-1}) \cdot f_{ME}(w_1, \cdots, w_{N-1}, w_N) \quad (7.3)$$

and derive bow_{ME} and f_{ME} from α's.

We focus on the trigram model

$$p(w|u, v) = \frac{\alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)}}{z(u, v)}$$

where u, v are two preceding words for w for illustration. Setting

$$\begin{aligned}
\zeta &= \sum_w \alpha_w^{g(w)}, & (7.4)\\
\zeta(v) &= \sum_w \alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)}, & (7.5)\\
\zeta(u, v) &= \sum_w \alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)}, & (7.6)\\
f_{ME}(w|u, v) &= \frac{\alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)}}{\zeta(u, v)}, &\\
\mathsf{bow}_{ME}(u, v) &= \frac{\zeta(v)}{\zeta(u, v)}, &\\
f_{ME}(w|v) &= \frac{\alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)}}{\zeta(v)}, &\\
\mathsf{bow}_{ME}(v) &= \frac{\zeta}{\zeta(v)}, &\\
f_{ME}(w) &= \frac{\alpha_w^{g(w)}}{\zeta}, &
\end{aligned}$$

we get

$$p(w|u,v) = \frac{\alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)}}{z(u,v)} = \mathsf{bow}_{ME}(u,v) \cdot f_{ME}(w|u,v).$$

Replacing f and bow in the ARPA back-off model (in Table 7.1) by f_{ME}'s and bow_{ME}'s, we obtain an ME trigram model with ARPA back-off format.

For any ME N-gram model (7.2), we can pre-normalize it in the same way. Setting

$$\begin{aligned}
\zeta &= \sum_w \alpha_w^{g(w)}, \\
\zeta(w_1, \cdots, w_{n-1}) &= \sum_{w_n} \prod_{k=1}^{n} \alpha_{w_1,\cdots,w_n}^{g(w_1,\cdots,w_n)} \text{ for } n = 1, 2, \ldots, N-1, \\
f(w_n|w_1, \cdots, w_{n-1}) &= \frac{\prod_{k=1}^{n} \alpha_{w_1,\cdots,w_k}^{g(w_1,\cdots,w_k)}}{\zeta(w_1, \cdots, w_{n-1})} \text{ for } n = 1, 2, \ldots, N-1, \\
\mathsf{bow}(w_1, \cdots, w_{n-1}) &= \frac{\zeta(w_1, \cdots, w_n)}{\zeta(w_1, \cdots, w_{n-1})} \text{ for } n = 1, \ldots, N-1,
\end{aligned}$$

we rewrite (7.2) as (7.3), which has the back-off form.

7.1.3 Speed-up

We compare the time of rescoring 100-best hypotheses of the test data for Switchboard and Broadcast News, using the original maximum entropy model and the pre-normalized one (in Table 7.2). We use the ME trigram model described in Section 4.3 for Switchboard experiments here and the model in Section 4.5 for Broadcast News. We do not count the overhead time of loading the models in our experiments. Column 1 in the table indicates the test sets with their size (in number of utterances). The numbers in the second and the third columns represent CPU-mins used in rescoring the 100-best hypotheses using the original model and the pre-normalized model, respectively. The speed-up of the pre-normalized model, the ratio of Column 2 and Column 3, is shown in Column 4. It is apparently that the pre-normalization decreases the computation time tremendously (about 480 fold for Switchboard and 1000 fold for Broadcast News) because the calculation of z is omitted.

Task(# of Utts.)	Original	Pre-normalized	Speed-up
Swbd (2400*100)	960	2	$4.8 \cdot 10^2$
BN (1300*100)	2077	2	$1.0 \cdot 10^3$

Table 7.2: Speed-up of the pre-nominalized model.

7.1.4 Using the ME Trigram Model in the First Pass of Speech Recognition

We transformed the ME trigram model from the exponential form to the ARPA format in the previous section. Now we create FSMs using both the back-off model and the ME model, respectively. Then we recognize the test speech data of Switchboard-I[4] and Switchboard-II[5] using the above two language models and the same acoustic model by the AT&T FSM decoder.

The acoustic model in 2001 CLSP evaluation system (Byrne, 2001) is used here. The acoustic training set used contains 150 hours of Switchboard I and 15 hours of CallHome transcribed acoustic data. The acoustic features are 12 dimensional PLP-Cepstral coefficients with an energy term, and their first and second derivatives (39 coefficients per frame). Cepstral mean normalization, vocal tract normalization and variance normalization are performed for each conversation side on both the training and the test data. The models are continuous mixture density, tied state, cross-word, context-dependent word-boundary triphones based on the HTK toolkit.

Table 7.3 shows the perplexity and word error rate on these two Switchboard test sets using the back-off model and the ME model. The maximum entropy model achieves slightly better performance than the corresponding back-off models. However, it remains for arguments in favor of these arguments of these experiments to show the possibility of using ME models in the first pass speech recognition.

[4]The same kind of Switchboard data as we used in the experiments in the preceding chapters.

[5]Speech data collected in the same way as Switchboard-I but with poor quality in acoustics.

	SWBD-1 (00-test)		SWBD-2 (98-test)	
	PPL	WER	PPL	WER
BO	125	32.9%	145	47.7%
ME	123	32.7%	147	47.4%

Table 7.3: Perplexity and WER of using the BO and ME models in the first pass recognition.

7.2 Approximating Models with Topic Features

The major purpose of using maximum entropy methods is not to build N-gram models, but to build models with some non-nested features, *e.g.,* the topic model. There are no corresponding back-off models to which topic-dependent models can be mapped. Therefore the topic-dependent model cannot be pre-normalized and its normalization factors must be computed when the model is used. However, we can approximate the normalization factor z of the topic-dependent model so that it can be easily estimated. The basic idea is to apply the computational tricks introduced in Chapter 3 to rewrite z to a history-independent constant plus some terms that can be pre-computed. We show this approximation method using the topic-dependent model

$$
\begin{aligned}
p(w|u,v,t) &= \frac{\alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)} \cdot \alpha_{t,w}^{g(t,w)}}{z(u,v,t)}, \\
z(u,v,t) &= \sum_{w \in V} \alpha_w^{g(w)} \alpha_{v,w}^{g(v,w)} \alpha_{u,v,w}^{g(u,v,w)} \alpha_{t,w}^{g(t,w)}
\end{aligned}
$$

where t is the topic of the current utterance, u, v are the two immediately preceding words of w. We apply the computational trick in Chapter 3 and rewrite $z(u,v,t)$ as

$$
\begin{aligned}
z(u,v,t) &= -\sum_{w \in V} \alpha_w^{g(w)} + \sum_{w \in Y_w} \alpha_w^{g(w)} \cdot \alpha_{v,w}^{g(v,w)} \cdot \alpha_{u,v,w}^{g(u,v,w)} + \sum_{w \in Y_t} \alpha_w^{g(w)} \alpha_{t,w}^{g(t,w)} \\
&+ \sum_{w \in Y_w \cap Y_t} \alpha_w^{g(w)} (\alpha_{v,w}^{g(v,w)} \alpha_{u,v,w}^{g(u,v,w)} - 1) \cdot (\alpha_{t,w}^{g(t,w)} - 1),
\end{aligned}
$$

where Y_w is the set of words with bigram and/or trigram features activated and Y_t is the set of words with the topic feature activated.

Applying ζ and $\zeta(u,v)$ defined in (7.4) and (7.6) and setting

$$\begin{aligned}\zeta(t) &= \sum_{w\in V} \alpha_{t,w}^{g(t,w)}\alpha_w^{g(w)},\\ \zeta(u,v,t) &= \sum_{w\in Y_w\cap Y_t} (\alpha_{v,w}^{g(v,w)}\cdot\alpha_{u,v,w}^{g(v,u,w)}-1)(\alpha_{t,w}^{g(t,w)}-1)\alpha_w^{g_w},\end{aligned}$$

we can rewrite $z(u,v,t)$ by

$$z(u,v,t) = -\zeta + \zeta(u,v) + \zeta(t) + \zeta(u,v,t). \tag{7.7}$$

If the history u,v is unseen in the training data, $\zeta(v)$ is used instead of $\zeta(u,v)$.

Now we analyze the complexity of this implementation (7.7). The first term ζ is a history-independent constant. The second one $\zeta(u,v)$ depends only on u,v and the third one $\zeta(t)$ depends only on t. All these three terms can be pre-computed for all histories in the last iteration of training. However, the last term has $O(|\hat{X}|\cdot|T|)$ possible combinations of (u,v,t). It is impossible to pre-compute and store this term for all histories. We study the scale of this resident term $\zeta(u,v,t)$ relative to z and find that it is comparatively very small ($< 3\%$ on average). Figures 7.1 and 7.2 show the resident ratio of $\zeta(u,v,t)$ and z for histories in the test sets of Switchboard and Broadcast News, respectively.

Figures 7.3 and 7.4 show the accumulated percentage of histories vs. the resident ratio of $\frac{\zeta(u,v,t)}{z}$ for Switchboard and Broadcast News, respectively. The X-axle represents the ratio of $\zeta(u,v,t)$ and z, and the value on the Y-axle represents the percentage of histories whose ratio is less than the corresponding value on the X-axle. For example, one can see from the curve that 95% of histories has a very small resident ratio of $< 3\%$. Therefore, we can omit this term and estimate $p(w|u,v,t)$ by the approximated model

$$p(w|u,v,t) \approx \frac{\alpha_w^{g(w)}\cdot\alpha_{v,w}^{g(v,w)}\cdot\alpha_{u,v,w}^{g(u,v,w)}\cdot\alpha_{t,w}^{g(t,w)}}{\zeta+\zeta(u,v)+\zeta(t)}. \tag{7.8}$$

The topic-dependent probability can be estimated as fast as an N-gram probability. Of course, this approximation may result in some imprecision. The question is whether the error is bearable.

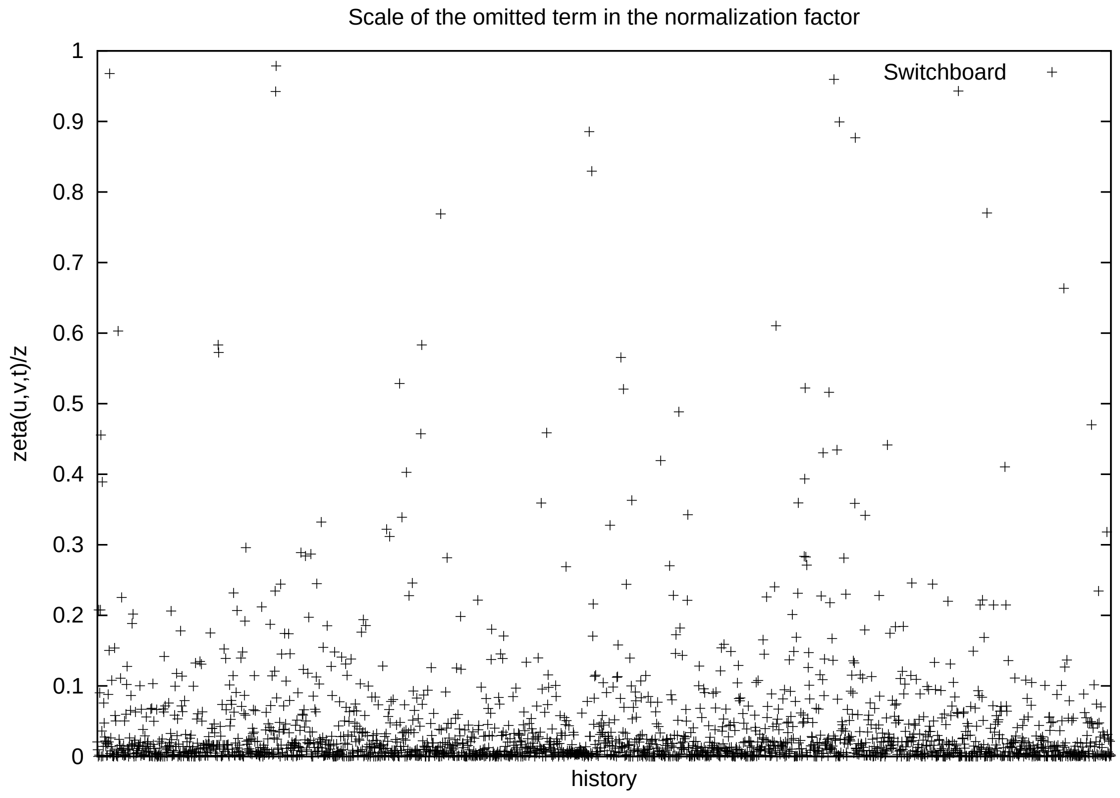

Figure 7.1: Ratio of $\zeta(u, v, t)$ and z in Switchboard.

7.2.1 Experimental Results

We approximate topic models described in Sections 4.3 and 4.5. The perplexity of the approximated model (7.8) is not available since the right-hand side is not a real probabilistic measurement. However, we can rescore the 100-best lists (used in Chapter 4) by this approximate model and compare the WER with that of the exact model. The results are shown in Table 7.4.

Task	Original WER	Approx WER	Speed-up
SWBD(97dev-test)	37.8 %	37.9 %	$6.8 \cdot 10^2$
BN(96ws)	34.0 %	34.0 %	$1.9 \cdot 10^3$

Table 7.4: Influence of approximation on topic models.

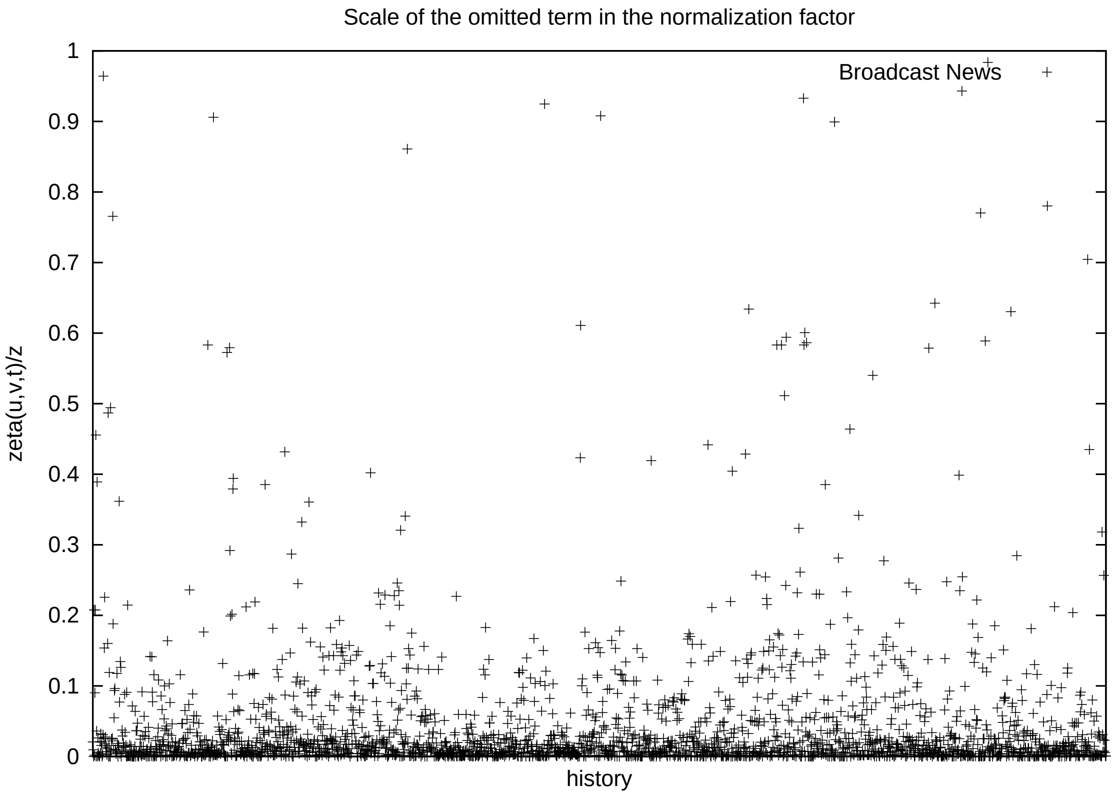

Figure 7.2: Ratio of $\zeta(u, v, t)$ and z in Broadcast News.

The original WERs for Switchboard and Broadcast News are duplicated from Tables 4.6 and 4.13, respectively. The WERs of approximated models follow in the table. The speed-up is defined as the ratio of the rescoring time for the 100-best hypotheses when using the original model and that when the approximated model is used. In Switchboard, the WER only degrades by 0.1% from that of the exact model if the approximate model is used. In Broadcast News, the WER of using the approximate model is the same as that of the original model. Therefore, the approximation is worth applying; it accelerates the decoding speed in orders of magnitude with almost no degradation in the speech recognition accuracy.

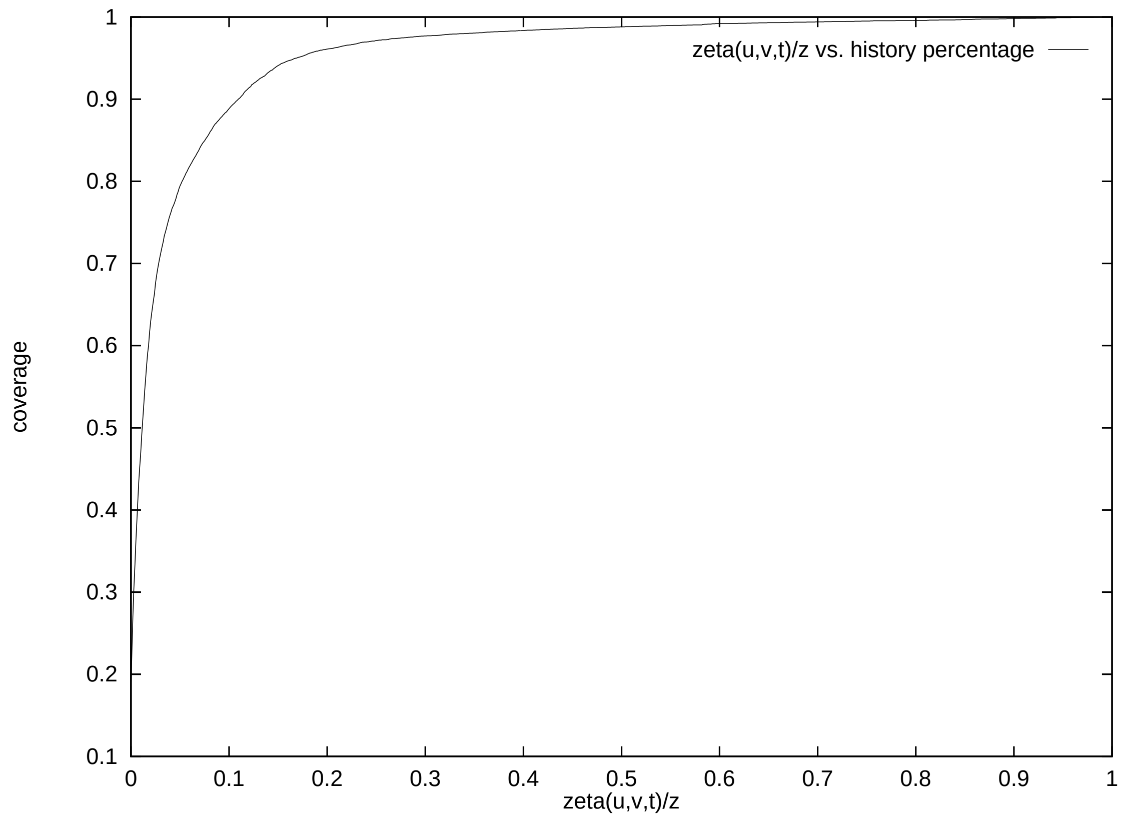

Figure 7.3: Percentage of histories vs. $(z - \zeta)/z$ (SWBD).

7.3 Caching Recent Histories

Pre-normalizing and approximating histories are practical for some special models either without non-nested features or with only one kind of non-nested feature. For ME models with many kinds of non-nested features, such as the syntactic model, pre-normalization is impossible and approximation is inaccurate. However, not all the time z needs to be computed to acquire probability p. For example, if $p(y_2|x)$ is estimated immediately after $p(y_1|x)$, $z(x)$ needs to be calculated only once instead of twice. We observe that the distribution of histories respects the so-called Zipf's Law[6]: a few histories occur very frequently and they cover much of the test data. Further, the occurrence of histories has the property of locality: in the N-best list or lattice

[6]The product of the frequency of occurrence $f(e)$ of some event e and the rank $i(e)$ of e in the element set is almost a constant.

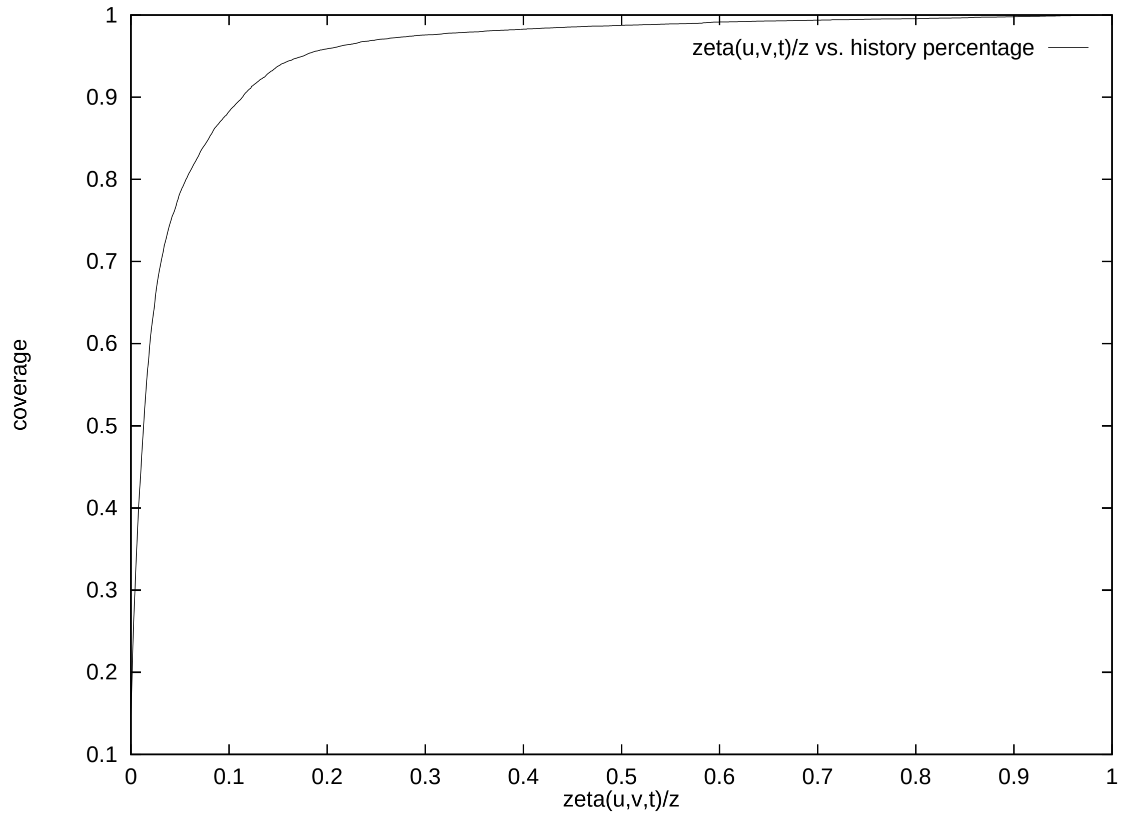

Figure 7.4: Percentage of histories vs. $(z - \zeta)/z$ (BN).

rescoring, a history tends to be referenced again soon if it is referenced. These two attributes of the distribution of histories allow us to "cache" the normalization factors of the most frequent and/or the last accessed histories in the memory and reuse them in the future. If a history is found in the cache, its normalization factor can be acquired directly without any computation; otherwise, z is calculated by summing over all numerators.

The efficiency of this method depends on the fraction of history accesses found in the cache. We define the *hit ratio* η as

$$\eta = \frac{\#[\text{history accesses found in cache}]}{\#[\text{total history accesses}]}.$$

Letting T_{old} be the average time of computing z by (2.30) without cache and T_{hit} be the table look-up time if z is found in cache, the average time required to compute z

with cache is

$$T_{new} = \eta \cdot T_{hit} + (1-\eta) \cdot T_{old}.$$

The speed-up of this method is thus

$$\frac{T_{old}}{T_{new}} = \frac{T_{old}}{\eta \cdot T_{hit} + (1-\eta) \cdot T_{old}}$$

because $T_{hit} << T_{old}$,

$$\text{speed-up} \approx \frac{1}{1-\eta}.$$

If $\eta < 50\%$, history caching is almost meaningless.

The remaining question is what histories should be stored in the cache. We inspect two solutions for cache management by
(1) statically saving the most frequent histories in the training set, and
(2) dynamically caching the last accessed histories in rescoring.
Figures 7.5 provides curves of the number of (the most frequent) histories (obtained from the training data) vs. the coverage rate of these histories in the test data. It is apparent that the coverage rate increases quite slowly the size of cache increases. The cache size must be at least 30K for Switchboard and 300K for Broadcast News to obtain a high coverage rate (of about 80%).

Therefore, we dynamically cache the normalization factors of the last visited histories in memory. We generate a hash function

$$h : X \rightarrow [1..K]$$

mapping the history set X to K integers from 1 to K where K is the size of the cache. Since $K << |X|$[7], many histories will map to the same cell in the cache. We solve this cache address conflict problem by saving history x as the *key* in each cell in the cache. The value $z(x)$ stored in the cache is valid if and only if the key matches x; otherwise, it needs to be calculated by (2.30).

The algorithm below shows the steps of computing z with a cache.

[7] Otherwise, the cache size is huge.

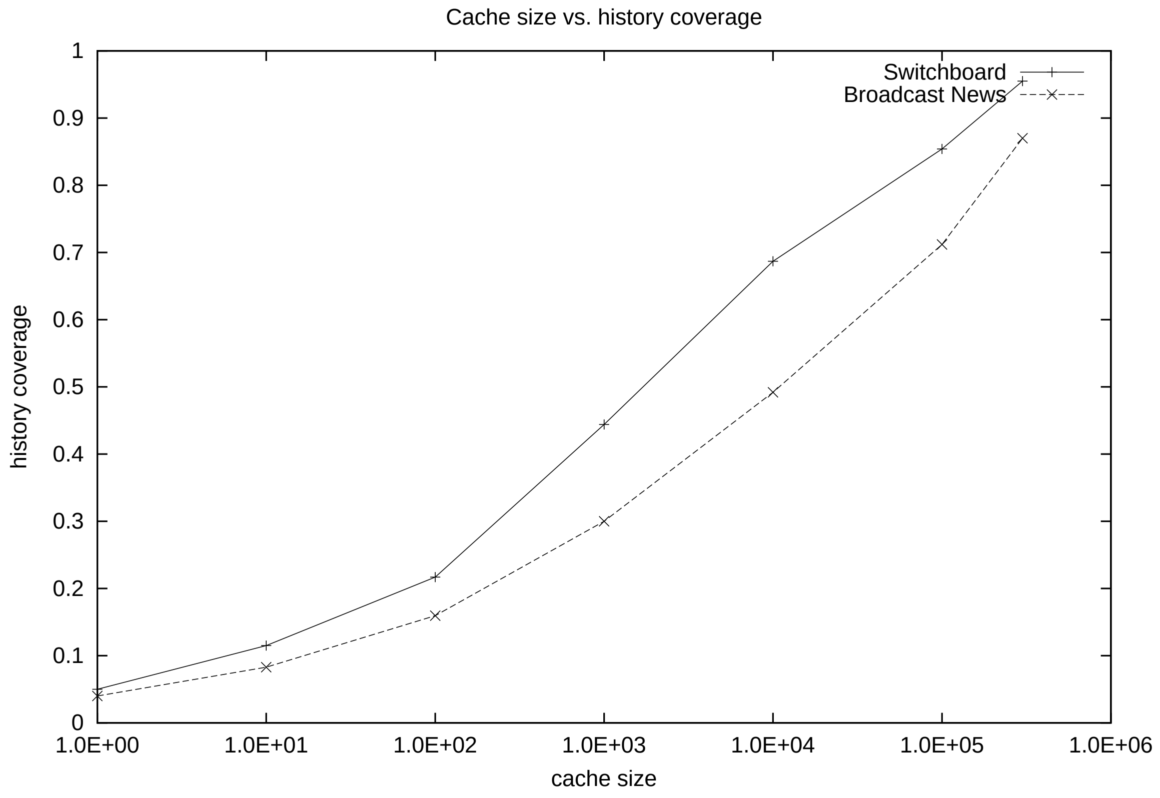

Figure 7.5: History coverage vs. cache size.

Algorithm (Computing the normalization factor with a cache):

```
Step 1:  Compute the cache address $h(x)$ for the history $x$.
                     $h(x) = (\sum_{i=1}^{d} r_i \cdot x_i) \bmod K$
         where
         $x = x_1 \cdots x_d$ is a d-dimension vector,
         $r_1 \cdots r_d$ are d pre-selected primes and
         K is the size of cache.
Step 2:  If $\mathsf{key}(h(x)) = x$ ,  then set $z = \mathsf{value}(h(x))$;
                                 otherwise, compute $z$ according to Equation (2.30) and update the cache by setting
                                 $\mathsf{key}(h(x)) = x$ and $\mathsf{value}(h(x)) = z$.
```

∎

7.3.1 Experimental Results

We use the above algorithms to compute z in rescoring the 100-best hypotheses and examine their cache hit rate. We first fix the cache size of 10,000 histories and check the hit rate for different models. The results are shown in Table 7.5. Overall, the hit rate is about 80% or more for all language models, resulting in a speed-up of about 5 fold or more. It is not surprising to see that trigram models have the highest hit rates, since these models have relatively small history space, whereas the composite models have the lowest hit rate, since the number of possible histories is huge. The composite model has a higher miss rate compared to the syntactic model because the most of misses in using the former occur when switching topics.

Task	3gram	Topic	Syntax	Comp
SWBD	96%	96%	87%	78%
BN	97%	96%	87%	77%

Table 7.5: Hit rate for different models.

Next, we adjust the size of the cache from 10 to 100,000 and measure the hit rate of the trigram model. See Figure 7.6 for the curve of the hit rate vs. cache size. It is apparent that the hit rate increases tremendously as the cache size grows from 10 to 10,000 but it increases slowly as the cache size further grows. Overall, the hit rate of dynamic caching is much higher than that of static caching with the same cache size. (Figure 7.5 vs. Figure 7.6).

7.4 Summary

We have discussed the computational issue of using maximum entropy models. We can pre-normalize some ME models such as N-gram models and transform them to the ARPA format of back-off models. Pre-normalized models are two to three orders of magnitude faster than the original ones.

For the models with non-nested features, there are no corresponding back-off models that we can map to. So a similar simplification is not possible. However, we can

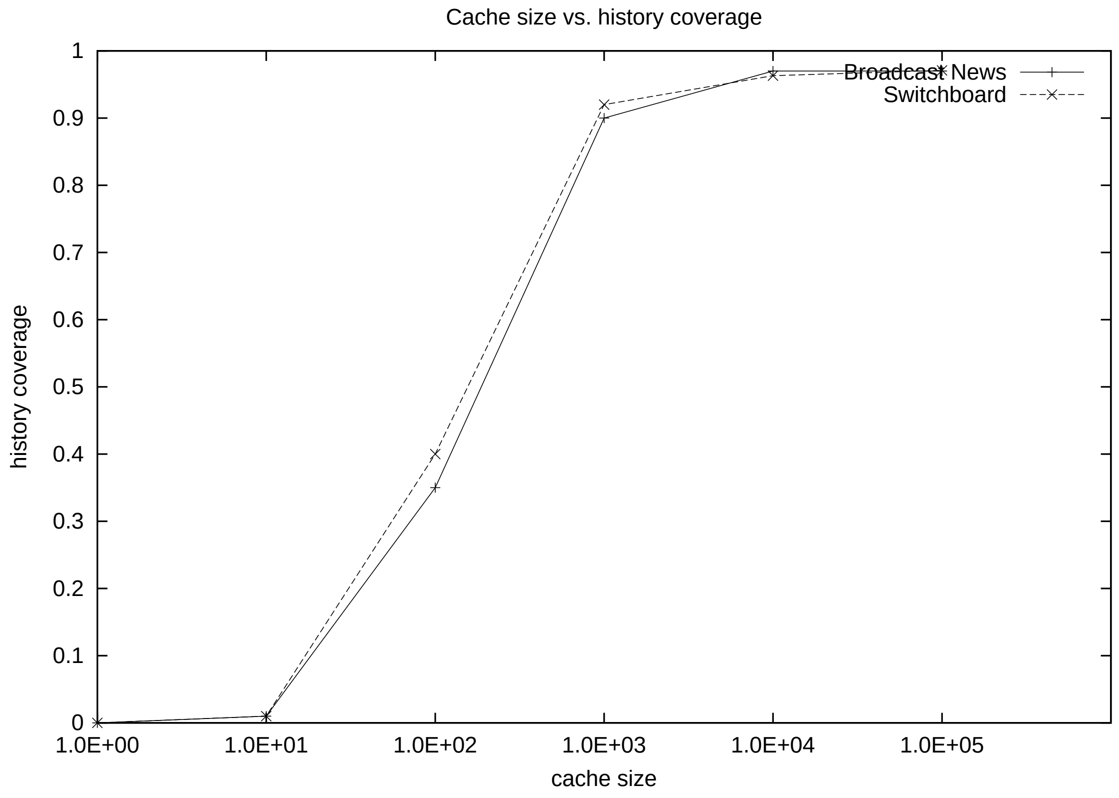

Figure 7.6: Hit rate vs. cache size (trigram model).

approximate the denominators in some of these models, such as the topic-dependent model, so significantly reducing the computation needed during recognition. Experimental results in Switchboard and Broadcast News show that the approximation of the denominator achieves a tremendous speed-up and creates almost no degradation in speech recognition performance.

The approximation error, however, may be significant when the number of non-nested features in a model is very large. Therefore, we design a history-caching method to save the computation for ME models in general without any approximation. The experimental results on Switchboard and Broadcast News show that the hit rate is about 80% with a moderate cache-size for all models described in this dissertation, indicating a speed-up of about 5 fold.

Chapter 8

Conclusion

Maximum entropy methods can combine constraints from different information sources into one unified framework. Maximum entropy models are accurate, smooth and efficient in size. In this dissertation, we have shown these advantages of maximum entropy methods via three examples of language modeling. Substantial improvement in speech recognition has been achieved by using maximum entropy models with both collocational and long-distance dependencies. The intensive computational load in estimating the parameters of maximum entropy models has also been decreased. In this chapter, we summarize the major contributions, both theoretical and experimental, and discuss the future extensions and other potential applications of this work.

8.1 Contributions and Summary

8.1.1 Accuracy

We show that maximum entropy models improve accuracy in two different large vocabulary speech recognition tasks, Switchboard and the Broadcast News. The maximum entropy models with non-local dependencies achieve a substantial improvement compared to the baseline back-off models. We have presented several kinds of maximum entropy language model and compared them with corresponding interpolated and back-off models with the same constraints. Maximum entropy models almost always achieve slightly but consistently better results than the latter. The comparison between maximum entropy trigram models and back-off models has been presented in Sections 4.3 and 4.5. The comparison between maximum entropy topic-dependent models and corresponding interpolated models has been presented in Sections 4.4.1 and 4.5.2, and that between maximum entropy syntactic models and corresponding interpolated models in Sections 4.5.2 and 5.8.2.

We also show that the high accuracy of maximum entropy models can be obtained with insufficient training data. We trained a maximum entropy model with 800K model parameters using only 2.1 million words of training data for Switchboard, and a model with 5M parameters using only 14 million words of data for Broadcast News. Maximum entropy methods avoid potential data sparseness problems by using only reliable (marginal) counts.

The high accuracy of maximum entropy models also come from properly choosing features for the model. For instance, the part-of-speech tags and syntactic non-terminal labels have the similar form. However, the latter are grammatically more meaningful than the former and comparatively will result in more improvement in speech recognition.

8.1.2 Efficiency

We demonstrate that maximum entropy models with various kinds of constraints are more space efficient than the corresponding interpolated models. We have shown in Chapter 4 that the topic-dependent maximum entropy models are only about 1/8 the size of their interpolated counterparts. The maximum entropy syntactic models also have few parameters than the corresponding interpolated models described in Chelba (2000), because the former use only marginal constraints while the latter need to store both marginal and joint counts.

We also claim that both the training and the use of maximum entropy models are efficient in time using the methods described in this dissertation. Proir to our work, the training of maximum entropy language models of large size was almost intractable. Several efficient training methods have been presented in Chapter 3 to simplify the intensive computation in estimating model parameters. Models with only nested features, such as N-gram models, have hierarchical structures and can be trained as fast as training back-off models. Other models can also be converted to equivalent hierarchical models by hierarchization. Topic-dependent models in particular can be trained by divide-and-conquer in addition to hierarchical training; their parameters are estimated in topics and then merged. A nominal speed up of hundreds to thousands fold and a real speed-up of ten to one hundred fold have been achieved compared to the baseline unigram-caching method.

The computational load in calculating probabilities using maximum entropy models is also reduced significantly.

It is worth mentioning that the efficient training methods described about can be apply to other application than language modeling. We show, by the syntactic parser presented in Charniak (2000), how to train other statistical models hierarchically.

Training Charniak (2000) Parser Efficiently

The top-down parser described in Charniak (2000) explored each constituent c in a parse π by first predicting the pre-terminal tag[1] t of c, then the lexical head word h and finally the non-terminal label l. According to Equation 1 in Charniak (2000), the probability of a parse π is assigned as

$$p(\pi) = \prod_{c \in \pi} p(t|l, H) p(h|t, l, H) p(e|l, t, h, H)$$

where

$$H = l_p, t_p, l_b, l_g, h_p$$

is the history. Subscripts p, b and g in the equation above represent, respectively, the parent, the sibling and the grandparent of c. All three probabilities above can be estimated by ME models. It is apparent that the computationally intensive ME models are $p(h|t, l, H)$ and $p(e|h, t, l, H)$. We illustrate how to train these models efficiently by the example of $p(h|t, l, H)$. The training of the remaining models is similar.

[1]Similar to the part-of-speech tag.

If we use the features suggested in Equation (7) in the paper, we can rewrite $p(h|t,l,H)$ as

$$\begin{aligned} p(h|t,l,H) &= \frac{1}{z(t,l,H)}\alpha_h^{g(h)} \cdot \alpha_{t,h}^{g(t,h)} \cdot \alpha_{l,t,h}^{g(l,t,h)} \cdot \alpha_{l_p,l,t,h}^{g(l_p,l,t,h)} \cdot \alpha_{t_p,l_p,l,t,h}^{g(t_p,l_p,l,t,h)} \\ &\cdot \alpha_{l_b,l_p,l,t,h}^{g(l_b,l_p,l,t,h)} \cdot \alpha_{l_g,l_p,l,t,h}^{g(l_g,l_p,l,t,h)} \cdot \alpha_{h_p,l_p,l,t,h}^{g(h_p,l_p,l,t,h)}. \end{aligned} \qquad (8.1)$$

The computational complexity of training this model using unigram-caching is considerably high (See Table 8.1). Therefore, Charniak (2000) used non-normalized models[2] instead. However, this approximation may cause degradation in performance. A better way of implementation is to train real ME models using hierarchical training methods. If the last three 5-gram features $g(l_b,l_p,l,t,h)$, $g(l_g,l_p,l,t,h)$ and $g(h_p,l_p,l,t,h)$ in Equation (8.1) are compounded into a superfeature $f(l_b,l_g,h_p,l_p,l,t,h)$, all features in (8.1) are nested, and thus the model can be trained hierarchically. Table 8.1 shows the computational load, in number of terms as described in Section 3.9.1, of unigram-caching and generalized hierarchical training, and the nominal speed-up of the latter. It is apparent that training such an ME parser will become practical (after

Model	Unigram-caching	Hierarical Training	Speed-up
$p(e\|l,t,h,H)$	$1.7 \cdot 10^9$	$2.9 \cdot 10^6$	580
$p(h\|l,t,H)$	$2.3 \cdot 10^9$	$2.9 \cdot 10^6$	790

Table 8.1: Nominal speed-up in training Charniak parser.

a speed-up of hundreds of fold) when the generalized hierarchical training is applied, even though its computational load is intractable using unigram-caching.

8.1.3 Flexibility

We claim that the maximum entropy method has the high flexibility of incorporating constraints from different information sources in a unified framework. In this dissertation, we started from a trigram model and then augmented it by adding semantically and syntactically meaningful constraints. Finally, we combined both topic

[2]Assuming $Z = 1$.

constraints and syntactic constraints in one model and obtained almost additive improvement from both kinds of constraints. All these maximum entropy models have the similar exponential form and differ only in feature sets. Moreover, maximum entropy methods provide the flexibility of further expanding these models by adding new meaningful features.

8.1.4 Summary of Experimental Results

We summarize the improvement achieved in this dissertation work in Table 8.2. The table shows the relative perplexity reduction and the absolute word error rate re-

Language	Switchboard				Broadcast News			
	Speed-up		Reduction		Speed-up		Reduction	
Model	Nom.	Real	Ppl	WER	Nom.	Real	Ppl	WER
Trigram	170	30	79	38.5%	560	85	174	34.6%
Topic	400	330	-7%	-0.7%	1300	-	-12%	-0.7%
Syntactic	90	17	-6.3%	-1.0%	140	-	-7.0%	-0.8%
Composite	-	-	-13%	-1.5%	-	-	-18%	-1.2%

Table 8.2: Summary of experimental results (SWBD and 14M BN).

duction of maximum entropy models compared with the results of the baseline trigram models. The nominal speed-up and the real running time speed-up of the efficient training method based on the unigram-caching are also shown for each model. The speed-up of some models is not available since their baseline training is intractable.

Overall, the perplexity of the maximum model with topic and syntactic constraints reduces by 13% - 18% compared to the back-off trigram models in Switchboard and Broadcast News. The word error rate reduces by 1.2% - 1.5% (absolute). The gain from the topic information and the syntactic structure is almost additive, showing that these two kinds of non-local dependencies are complementary in language modeling.

8.2 Future Work

The work in this dissertation can be extended in several directions.

First, topic-dependent modeling techniques can be extend to building multilingual language models in machine translation. In a statistical machine translator, *e.g.* from English to Chinese, the most probable Chinese sentence $\hat{C}$ given the English sentence E is sought to maximize the posterior probability $p(C|E)$, *i.e.,*

$$\hat{C} = \arg\max_{C} p(C|E) = \arg\max_{C} p(E|C) \cdot p(C).$$

The probability $p(C)$ is computed by the Chinese language model. As we have shown in this dissertation, a precise estimate for $p(C)$ should depend on the topic of C. However, this topic information is not available until C is known. Of course, we can use the first-pass translation result of C to detect the topic and then use the topic-dependent model to rescore hypotheses. However, a better way is to detect the topic directly from the English sentence E, since the English sentence and its Chinese translation must have the same topic. The topic-dependent model can thus be used in the first-pass decoding from English to Chinese.

Second, the syntactic parser used in syntactic language modeling can be trained by maximum entropy methods. In Section 5.2, the probability $p(w_i|w_1, \cdots, w_{i-1})$ is computed as the product of $p(w_i|W_1^{i-1}, T_{ij})$ and $\rho(W_1^{i-1}, T_{ij})$. The former is already computed by the maximum entropy model, but the latter is still estimated by using three interpolated models: the predictor $p(w_i|h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$, the part-of-speech tagger $p(pos_i|w_i, nt_{i-2}, nt_{i-1})$ and the parser $p(parse_i|h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$, where $parse_i$ are the operations of constructing the parse tree, such as *joining the left tree to construct a noun phrase* or *joining the right tree to construct a verb phrase.* Details of estimating $\rho(W_1^{i-1}, T_{ij})$ are described in Chapter 2 of Chelba (2000). It is apparent that all three probability models used in the parser can be estimated by the maximum entropy method. The algorithm of estimating the parser parameters using maximum entropy methods is outlined below.

Initial step: Parse all sentences in the training data using the current parser described in Chelba (2000), or other state-of-the-art parsers.

Iteration step 1: Collect counts such as $\#[h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, w_i]$, $\#[w_i, nt_{i-2}, nt_{i-1}, pos_i]$ and $\#[h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, parse_i]$ as described in Chapter 2 of Chelba (2000).

Iteration step 2: Train probability models $p(w_i|h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$, $p(pos_i|w_i, nt_{i-2}, nt_{i-1})$ and $p(parse_i|h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1})$ using the maximum entropy method. (Tuples and their counts obtained from the step above are training samples.)

Iteration step 3: Check the perplexity on held-out data. Stop if the perplexity has converged. Otherwise parse the training data using the new model and go to **Iteration step 1**.

Two challenges must be overcome in implementing the algorithm above. First, iteration steps 2 and 3 are computationally intensive. Furthermore, there is another iteration loop inside step 2, which is for training maximum entropy models. To avoid confusion, we regard the iteration loop described in the algorithm above as the *outer* loop and that inside step 2 as the *inner* loop. Second, the constraints in maximum entropy models may change from iteration to iteration. For example, the tuple $\langle h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, w_i \rangle$ with a high initial count may obtain quite a low count after some iterations and thus should be removed from the feature set. Of course, some new constraints are also added after each iteration. This will result in different maximum entropy models in different outer iterations. If maximum entropy models are always trained starting from uniform models, the training time, compared to that of regular syntactic models, will increase proportionally to the number of the outer iterations.

We can reduce the training time per iteration by using the generalized hierarchical training described in this dissertation. Chelba (2000) also noticed that the feature sets in two consecutive outer iterations are largely overlapped. We can take advantage of this fact and set the initial model parameters by the values from the previous outer iteration for common features. For new features, we still assign $\alpha = 1$ as initial parameters. The maximum entropy models will converge faster after the first outer iteration than in the first outer iteration.

Finally, maximum entropy modeling techniques can be used in many other applications, such as question answering and automatic summarization. In these applications, the predicting information comes from various sources, for example, the positions of particular key words, the similarity between the current sentence and the question, and the length of the sentence etc. These applications will benefit from sound models combining different kinds of constraints using maximum entropy methods.

Appendix A

Sign-Test

Today, improvements in speech recognition based on state-of-the-art systems are usually small. Unless we have systematic evidence of differences in a consistent direction between model A and model B, we cannot claim model A is significantly better (or worse) than model B, even though the results of A may be slightly better than those of B, because this marginal difference may come from randomness. The sign test is used by National Institute of Standards and Technology (NIST) to evaluate the significance of the difference between speech recognition systems.

The principle of the sign test is quite simple. To check whether model A is significantly better than model B, we compare, for each test utterance, the number of word errors in the hypothesis generated by model A to the number in that produced by model B. We exclude the utterances for which both models produce the same number of errors, and then assign the utterances with fewer errors A a positive sign (+) and the others a negative sign (-). If there is no significant difference between the two models, the number of positive signs and that of negative ones should be the same or similar. On the other hand, if these two numbers differs considerably, the probability of random generation of this event is extremely small, so this result is most probably due to model A outperforming model B.

The probability of the event *N out of M signs are positive* is

$$\mathcal{B}(M, N, 0.5) = \binom{M}{N} 0.5^N (1 - 0.5)^{M-N}$$

if these two models have no difference in performance (*i.e.* , the null hypothesis holds). If the *P-value*,

$$\sum_{n=N}^{M} \mathcal{B}(M, n, 0.5) < p$$

where p is a small positive number close to zero, then the null hypothesis is rejected, and therefore A is significantly better than B. NIST selects the significance level of $p = 0.05$, whereas for this dissertation, we select a more strict one of $p = 0.01$.

Table A.1 illustrates the difference in word error rate between model pairs compared in the Switchboard experiments, and the corresponding P-value $\sum_{n=N}^{M} \mathcal{B}(M, n, 0.5)$. Usually, a WER difference of 0.3% absolute is regarded as significant by NIST. It is apparent that all improvements acquired from non-local dependencies based on the baseline trigram models are significant. It is interesting that even though the WER

Model A	Model B	Diff. WER	P-value
BO 3-gram	ME 3-gram	-0.2%	0.0091
BO 3-gram	Interp. Topic	-0.4%	0.0022
BO 3-gram	Cache-based	0.4%	0.034
BO 3-gram	ME Topic	-0.7%	2.1e-6
BO 3-gram	ME Syntax	-1.0%	6.2e-4
BO 3-gram	ME Composite	-1.5%	6.6e-9
Interp. Topic	ME Topic	-0.3%	0.0011

Table A.1: Significance of WER difference between language models (SWBD). Negative values in Diff. WER indicate that Model B is better than Model A.

difference of the trigram model and the cache-based model is 0.4%, the P-value of 0.034 is still higher than the standard of 0.01 used in this dissertation. This statistic may be due to the high WER in the cache, which causes high randomness in the cache-based model.

Table A.2 shows the WER difference and the corresponding P-value for model pairs compared on Broadcast News. Again, the significance test results show that gains in WER are very significant for topic and syntactic models. It is worth noting that the composite model trained using 14M words is significantly better than the baseline trigram model trained using 130M words.

Model A	Model B	Diff. WER	P-value
BO 3-gram (14M)	ME Topic (14M)	-0.7%	0.00042
BO 3-gram (14M)	ME Syntax (14M)	-0.7%	0.00087
BO 3-gram (14M)	ME Composite (14M)	-1.2%	8.1e-6
BO 3-gram (130M)	Interp. Topic (130M)	-0.6%	0.00012
BO 3-gram (130M)	ME Composite (14M)	-0.5%	0.00065

Table A.2: Significance of WER difference between language models (BN). Negative values in Diff. WER indicate that Model B is better than Model A.

Appendix B

Documentations of Maximum Entropy Toolkits

One of the contributions of this dissertation is the production of ME toolkits for the training on and use of maximum entropy models. This appendix presents the brief design document for our toolkits.

Maximum entropy methods can be used to estimate any probability distribution on the joint space of history $\mathcal{X}$ and future $\mathcal{Y}$. However, our toolkits are designed to estimate and evaluate only conditional probability models, because we focus on the applications in natural language processing, which need conditional models rather than joint ones. In Section B.1, we first describe the implementation details of the training and evaluation algorithms. The former is very complicated, whereas the latter is straightforward. Therefore, we focus on the former. In Section B.2, we present the major data structures used in our programs.

B.1 Implementation of Hierarchical Training Algorithms

The training procedure of an ME model is split into two phases: computing the feature expectations and updating the model parameters. We use a concrete example of the composite model (6.1)

$$\begin{aligned} & p(w_i|w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-1}, nt_{i-2}, t_i) \\ = \; & \frac{1}{z}\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{h_{i-1},w_i}^{g(h_{i-1},w_i)} \cdot \alpha_{h_{i-2},h_{i-1},w_i}^{g(h_{i-2},h_{i-1},w_i)} \\ & \alpha_{nt_{i-1},w_i}^{g(nt_{i-1},w_i)} \cdot \alpha_{nt_{i-2},nt_{i-1},w_i}^{g(nt_{i-2},nt_{i-1},w_i)} \cdot \alpha_{t_i,w_i}^{g(t_i,w_i)} \end{aligned}$$

described in Chapter 6 to show how our training programs estimate the model parameters.

B.1.1 Feature Expectation

In hierarchical training algorithms, the feature expectations (cf. Equations (3.44), (3.45) and (3.46)) and the normalization factors (cf. Equation (3.5)) are computed recursively. In principle, the recursive depth depends on the highest order of models. However, we fix the recursive depth to three for all models, so that the training program will not become too complicated. We will show in Section C.5 how to deal with features of order four or above. In this and the following appendices, we use x and w_i to represent the history and the future token, respectively.

When training the composite model, the computation in the first recursion is history independent and involves only unigram feature, that in the second recursion depends on history equivalence classes $(w_{i-1}, h_{i-1}, nt_{i-1})$ and involves both unigram and bigram features, and that in the third one depends on the entire history $x = w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-1}, nt_{i-2}$ and involves all features. The feature expectations can be gathered from three parts, each of which is obtained from one step above. In our programs, we denote partial expectations obtained from these recursions as PartEi_u, PartEi_phi and PartEi_x, respectively. It is obvious that the computation of unigram feature expectations is involved in all three recursions and the expectations of bigram features and trigram features are obtained from two parts: PartEi_phi and PartEi_x, and one part PartEi_x, respectively.

It has also been noted in Chapter 2 that the normalization factor z is required for computation of feature expectations. The computation of z is also split into three summations involving only unigrams, unigrams and bigrams, and all features, respectively. We denote the values obtained from these recursions as z_of_u, z_of_phi and z_of_x, respectively.

Our programs compute the expectations for all features in four steps described below.

Step 1: Loading global data and initialization.

- Load all features, including $U/B/T$ regular unigrams/bigrams/trigrams, B_h/T_h head-word bigrams/trigrams, B_n/T_n non-terminal bigrams/trigrams, and U_t topic-dependent unigrams, from eight different files. Features in these files are sorted alphabetically according to their coefficients. Assign index to each feature, *e.g.*, index from 0 to $U-1$ ($[0 : U-1]$) for unigrams, $[U : U+U_t-1]$ for topic-dependent unigram features, $[U+U_t : U+U_t+B-1]$ for regular bigrams and $[U+U_t+B : U+U_t+B+B_h-1]$ for head-word bigrams, etc.

  Set the initial values of zero to the expectations[1] for all features.

- Read all model parameters α (of the last iteration) and their target expectations from the model file, whose format will be introduced in Appendix C, and then map features to their corresponding parameters.

Next, for each topic (or part of training set) $t = 1, \cdots, T$, do steps 2-4.

Step 2: Computing z_of_u, the history-independent part of z, and, for each equivalence class $\phi(x) = w_{i-1}, h_{i-1}, nt_{i-1}$ observed in the training data, computing z_of_class, the part of z depending only on history equivalence classes.

- Merge two unigram feature parameters, α_{w_i} and α_{t,w_i}, to

$$\text{prod\_m\_alphas} = \alpha_{w_i}\alpha_{t,w_i}.$$

  Compute the history-independent unigram term z_of_u in the normalization factor z by

$$\text{z\_of\_u} \quad += \quad \text{prod\_m\_alphas}$$

  for all w_i.

[1]Since a feature g in the composite model may apply simultaneously with seven other features, it needs $max(g_{\#}) = 8$ accumulators for the expectation.

- For each history class $\phi(x) = w_{i-1}, h_{i-1}, nt_{i-1}$:
  - Enumerate all w_i that have any bigram feature activated, and for each $\langle \phi(x), w_i \rangle$, compute

    $$\mathsf{prod\_c\_alphas} = \alpha_{w_{i-1},w_i} \cdot \alpha_{h_{i-1},w_i} \cdot \alpha_{nt_{i-1},w_i} - 1$$

    where α equals one if the corresponding g is zero.
  - Compute z_of_class for the history class $\phi(x)$ by

    $$\mathsf{z\_of\_class} \quad += \quad \mathsf{prod\_c\_alphas} \cdot \mathsf{prod\_c\_alphas}$$

    for all w_i.

Step 3 Computing z for each history, and gathering PartEi_x, the partial feature expectation.

- For each history $x \in \hat{X}$:
  - Find all words w_i that have any trigram feature activated, and for each w_i compute

    $$\begin{aligned} &\mathsf{prod\_c\_alphas} \\ &\quad = \quad (\alpha_{w_{i-1},w_i} \cdot \alpha_{h_{i-1},w_i} \cdot \alpha_{nt_{i-1},w_i} - 1) \\ &\qquad\quad \cdot \alpha_{w_{i-2},w_{i-1},w_i} \cdot \alpha_{h_{i-2},h_{i-1},w_i} \cdot \alpha_{nt_{i-2},nt_{i-1},w_i} \cdot \end{aligned}$$

  - Accumulate $z(x)$ by

    $$\mathsf{z\_of\_x} \quad += \quad \mathsf{prod\_c\_alphas} \cdot \mathsf{prod\_m\_alphas}$$

    for all w_i, and then

    $$\mathsf{z\_of\_x} \quad += \quad \mathsf{z\_of\_class} + \mathsf{z\_of\_u},$$

    where z_of_u and z_of_class are precomputed in Step 2.
  - For each w_i enumerated, collect expectations for all features[2] applied to $\langle x, w_i \rangle$ by

    $$\mathsf{PartEi\_x} \quad += \quad \frac{\hat{p}(x)}{\mathsf{z\_of\_x}} \mathsf{prod\_c\_alphas} \cdot \mathsf{prod\_m\_alphas}.$$

Step 4: Updating feature expectations by history-independent part PartEi_u and class-dependent part PartEi_u.

[2]Not only trigram features, but also unigram and bigram ones.

- For each history class $\phi(x)$, fetch all w_i[3] that has any bigram feature activated. Update expectations for unigram and bigram features applied to $\langle \phi(x), w_i \rangle$ by

$$\mathsf{PartEi} \quad += \quad [\sum_{x \in \phi} \frac{\hat{p}(x)}{\mathsf{z\_of\_x}}]\mathsf{prod\_c\_alphas}(\phi(x)) \cdot \mathsf{prod\_c\_alphas}(\phi(x)).$$

- For all unigrams (including topic-dependent unigrams), update feature expectations by

$$\mathsf{PartEi} \quad += \quad [\sum_{x \in \hat{X}} \frac{\hat{p}(x)}{\mathsf{z\_of\_x}}]\mathsf{prod\_m\_alphas}.$$

B.1.2 Merging Partial Feature Expectations and Updating Model Parameters

It is worth noting here that the feature expectations PartEi gathered above are only for one topic (or part) of any training sample. The overall values of these expectations are summations over all parts. Since we may use many machines to train the model in parallel and save partial expectations in different files, we need to merge these partial feature expectations before we use Newton's method as described in Section 2.5.3 to update model parameters.

B.1.3 Memory Concerning

In the algorithm we described above, the set of tokens w_i with some features activated for each history x (or history class $\phi(x)$) are dynamically generated. It may be argued that these words w_i need to be generated only once in the first iteration and saved to disk, and then reused for the remaining iterations. Actually, the ME toolkits of Ristad (1997) adapt to this implementation. The major drawback, however, is the enormous space required to save all combinations of histories and future tokens. The ME toolkits of Ristad (1997) cannot even train a regular trigram model for Switchboard without high cut-offs for N-grams. Another problem is the extremely long overhead time of loading the huge amount of data from disks. Since the disk

[3]Already generated in Step 2.

access time is thousands of times longer than the memory access time, reading these data is even slower than generating them dynamically. Therefore, we always generate the word set for each history dynamically in our programs.

B.1.4 Computing Conditional Probabilities Using ME Models

We have discussed details of the computational issues when ME models are used and have described the efficient algorithms in Chapter 7. Here, we only emphasize that the procedure of using ME models is very similar to that of training ME models. We summarize the steps of computing probabilities using ME models below.

Step 1: Loading data and initialization.
The same as in training.

Step 2: Computing z_of_u, which is history independent.

Step 3: For each test tuple $\langle x, w_i \rangle$, compute $z(x)$ and prod_m_alphas$\cdot$ prod_c_alphas, and set

$$p(w_i|x) = \frac{\text{prod\_m\_alphas} \cdot \text{prod\_c\_alphas}}{z(x)}.$$

It should be noted that if $z(x)$ is already available in cache, it need not be computed again; otherwise, it is calculated similarly as in the training.

B.2 Data Structures

We describe major data structures (C++ classes) for constraints, model parameters, histories and futures. We only show major data members and functions in these C++ classes, because the complete classes (in header files) total thousands of lines. We exemplify the role of class members in the comment lines preceding or following these members.

B.2.1 Model parameters

An ME model contains a list (or an array) of "constraints," which are defined as

```
class Constraint{
  // Collect feature expectations.
  // Index j is defined in Equation (2.28).
  void UpdatePart_Ei(int j, double e) {PartEi[j] += e;}

  // Update model parameters by the Newton method.
  void UpdateAlpha() {alpha= alpha * Newton(max_e_num);}

  // Other member functions, such as reading and writing
  // data members, are easy to implement and thus are omitted.

private:

  double target; // Target expectations.
  double alpha;  // Model parameters.
  double E_i;    // Feature expectations under the current model.

  // PartEi[ ] is the partial sum for the expectation of feature i.
  // MAX_C_NUM: Maximum # of constraints active simultaneously in
  // the model.
  double PartEi[MAX_C_NUM];

  // max_e_num is the maxinum # of constraints active simultaneously
  // with this constraint.
  // For example, MAX_C_NUM = 8 in the composite model. However,
  // max_e_num = 1 if this feature never applies with other features.
  int max_e_num;
};
```

B.2.2 Features

No features used in this dissertation are more complicated than trigram features. Therefore, all features are represented as N-grams in the ME toolkits. We show how to store and access these N-grams efficiently using the example of bigrams.

Bigrams (w_i, w_j) and their count $c(w_i, w_j)$ can be stored in a matrix $C_{i,j}$, where the subscripts i and j are the indices of token w_i and w_j, respectively. Since this matrix C is extremely sparse, *e.g.,* less than 2% of elements are non-zero in Switchboard, this implementation is extremely inefficient. Of course, the matrix can be compressed in a hash table in which the key is (w_i, w_j) and the value is $c(w_i, w_j)$. However, a hash table needs almost $O(1)$ data access time for any (w_i, w_j), so it is not efficient for the operation of *enumerating all* w_j *following* w_i, which is often used in the hierarchical training method. Therefore, we design the following data structure to store the bigram matrix.

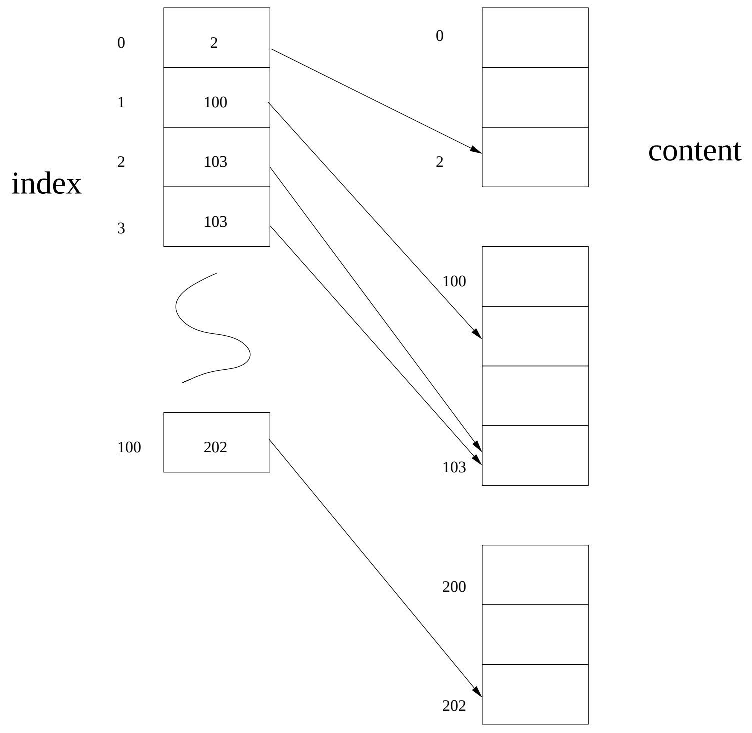

Figure B.1: Data structure for bigrams.

Figure B.1 illustrates the bigram data structure used in our programs. We use two one-dimensional arrays to represent bigrams, one for index and one for content. The index array itself is indexed by w_i, and it saves the position of the last bigram starting by w_i in the content array. For example, the first element (for w_0) in the index array is 2, meaning that the first two elements in the content array correspond to bigrams starting with w_0. w_j, and the count $c(w_i, w_j)$ will be stored as an element in the content array. To search for a bigram w_i, w_j, we first find positions for all bigrams starting with w_i (from index[i-1]+1 to index[i]), and then use binary search to locate the position of the pair $\langle w_j, c(w_i, w_j)\rangle$. This implementation has the advantage of efficiently finding all words following w_i (in elements from index[i-1]+1 to index[i] stored in the content array). Further, this implementation saves at least one-third of memory compared to a hash table. The typical bigram class is shown below.

```
class Bigram{
public:
  int Search(int w_i, int w_j);     // Search for bigram (w_i, w_j).
  int ReadBigram(const char* filename); // Read bigrams from a file.
  int GetConstraintId;          // Assign each bigram a feature index.

private:
  int *idx;         // Index array, sorted by w_j
  int *w2;          // w_j's
  float *freq;          // #[w_i,w_j]
  int *c_idx;           // constraint id for bigram (w_i, w_j)
};
```

Trigrams (w_i, w_j, w_k) are stored in the same way, the only difference is that the indices of trigrams are bigram (w_i, w_j) instead of single words.

B.2.3 History

The history class is the most important and also the most complicated class in the toolkits. It will dynamically generate all future tokens for a given history, and compute the normalization factor Z. The core of the history class is a group of Setby functions. These functions are the only model-dependent ones in the toolkit. Users can adding their own Setby functions for training and evaluating their own models. Here, we list some of these functions but focus only on SetbyHighNgrams, which is used in the algorithms we have described in Section B.1.

```
class History {
public:
  int SetbySynNgram;
  int SetbyNTSynNgram;
  int Setbyonly3gram;
  int SetbyTestNgram;
  int SetCache3gramTest;
  ...
  // Function SetbyHighNgrams will be used in Step 3 in the training
  // algorithms in Section B.1.
  // For each history, it enumerates all words that have some trigram
  // features.
  // Create a future class instance for each of these words.
  int SetbyHighNgrams(NTWorkSpace &temperate_word_set,
        NTHList &history_class_list,
        NgramHList &list_of_two_preceding_words,
        NgramHList &list_of_two_preceding_heads,
        NgramHList &list_of_two_preceding_nts,
        Unigram &list_of_unigrams,
        Bigram &list_of_regular_bigrams,
        Bigram &list_of_headword_bigrams,
        Bigram &list_of_nt_bigrams,
        Trigram &list_of_regular_trigrams,
        Trigram &list_of_headword_trigrams,
        Trigram &list_of_nt_trigrams,
        TupleTwo &list_of_training_tuples,
        HistoryClassList &history_classes_and_future_words,
        int &index_of_history_class_for_the_current_history);

  void UpdateHighNgramEi();     // For implementing Step 3 in the
              // training algorithm.

  // LM() will compute log probability for all words following this
  // history in either training and test data.
  // This function is not required to train ME models. However, we can
  // evaluate the log probabilities of the training data and see whether
  // the training procedure converges.
  double LM(int);

  // Create future classes for each word enumerated for this history.
  void SetMeFuture(int,MeFuture&);

  // Compute the history dependent part of normalization factor z.
  void SetHighNgramZ(double, int);

  // Other functions are omitted.
private:
  float p_of_x;     // p(x)
  MeFutureList futures; // Future list for the history.
  double z;         // z(x).
  int hc_id;        // History class_id.
  ...                   // Other unimportant data members.
};
```

The C++ class of the history equivalence class $\phi(x)$ described in the training algorithm is very similar to the class described above and is thus omitted.

B.2.4 Future

The set of words with some conditional features activated for a given history is represented as a list (or an array) of futures, which are described in the class below.

```
class MeFuture{
public:
  // Each word may have more than one marginal constraint, e.g., a
  // regular unigram and a topic-dependent unigram. This function will
  //  add a new constraint to the marginal constraint list.
  void AddMConstraint(Constraint);

  // Similarly, Each word may have more than one conditional
  // constraint.
  // This function will add a new constraint to the conditional constraint list.
  void AddCConstraint(Constraint);

  double SetCProd(ConstraintList&);     // Compute prod_c_alphas.
  double SetMProd();            // Compute prod_m_alphas.

private:
  float count;      // #[x, w_i]
  int c_n;      // Number of conditional features activated for <x,w_i>
  int m_idx;        // Unigram feature id.
  int* c_idx;       // Conditional feature ids.
  double prod_c_alphas;     // Product of all conditional features.
  double prod_m_alphas;     // Product of all marginal features.
};
```

Figure B.2 shows the relations among the history class, the future class and the constraint class. In this example, the history class has a list of M futures, and each future has a list of pointers to the constraint list.

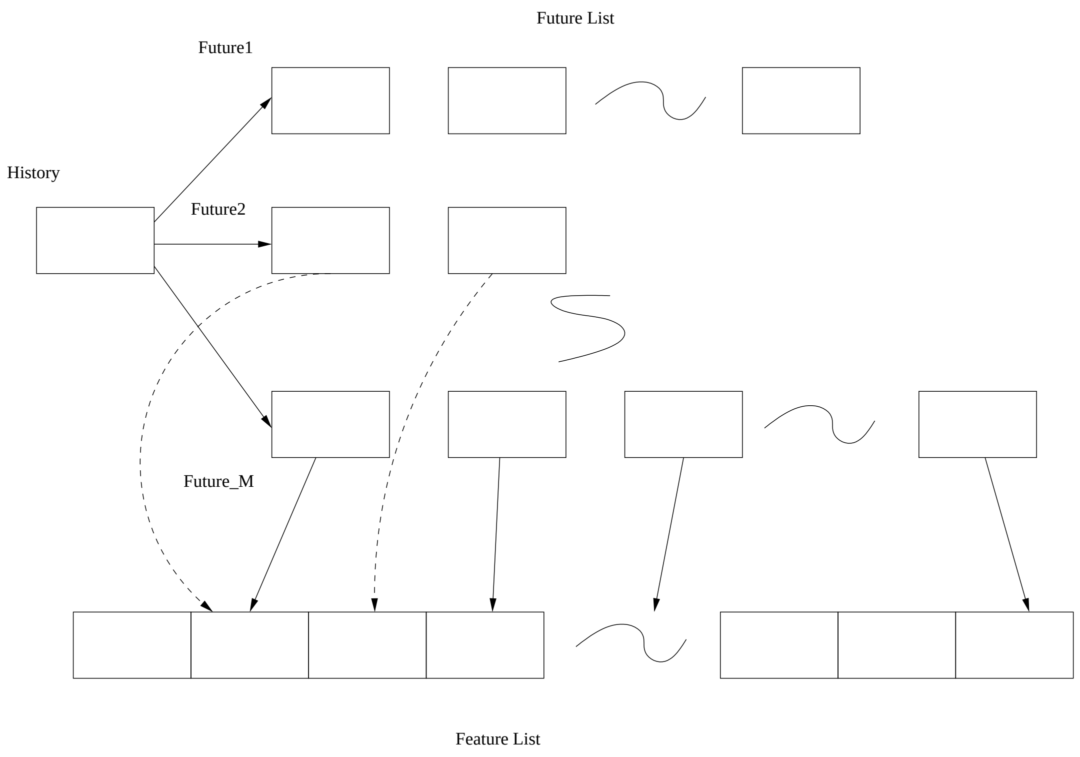

Figure B.2: History, future and constraint classes.

Appendix C

Users Manual

In this appendix, we describe how to use the ME toolkits to train ME models and how to compute conditional probabilities given an ME model. It is worth mentioning here that these toolkits are not a panacea for all ME modeling problems. However, they can solve many problems; especially, those in natural language processing, such as language modeling, part-of-speech tagging and syntactic parsing.

We expect that the readers of this manual either know the maximum entropy principle, or have read this dissertation.

C.1 Overview

In this section, we provide the availability, system requirements and the installation steps of our ME toolkits. We also briefly describe the functionality provided by the ME toolkit.

C.1.1 Availability

ME toolkits are available from

`http://www.clsp.jhu.edu/junwu/METK/Release-0.1.tar.`

Readers may contact the author by email

`junwu@clsp.jhu.edu.`

C.1.2 Functionality

The ME toolkits support the estimation and the prediction for maximum entropy models in discrete domains. The toolkits can be used to find the maximum entropy distribution for conditional probability $p(y|x)$, and to compute this probability using ME models. The current version of the toolkits can handle MN-gram (or similar) models, in which the condition x has less than or equal to M types of constraints and none of them are more complicated than N-gram constraints.

C.1.3 System Requirements

The toolkits will be run on the Unix/Linux systems. They have been tested on Sun Solaris and Linux for Intel CPUs.

The memory and disk space requirements are dependent on the size of the training data (or the number of constraints in the ME model). Today, the disk space is not a problem in building most ME models. The following table shows the approximated memory required for training trigram models in different tasks. Training complicated

Task(training size)	Memory Size
Switchboard ($< 3M$ words)	$< 128M$
WSJ/BN Subset ($\sim 10M$ words)	$< 512M$
BN Whole set ($\sim 100M$ words)	$1.5G$

Table C.1: Memory requirement for ME models.

models needs much more memory than training trigram models. For example, training the syntactic models described in Chapter 5 needs three-tenfold more memory than for training the trigram models on the same task.

C.1.4 Installation

After downloading the `tar` file to the local directory selected for installation, the user needs to expand the `tar` file and create training and evaluation programs by

```
tar -xvf Release-0.2.tar.
```

The user must then change to Release-0.2/src and build executable files by

```
make -f trainX.make, and
make -f testX.make
```

where X can be 3gram, _composite or _flat for trigram models, composite models and flat models, respectively.

Three pairs of training and evaluation executable files in total will be created in the Release-0.2/bin directory.

C.1.5 Tree Structure of Directories

The tree structure of directories is shown in Figure C.1.

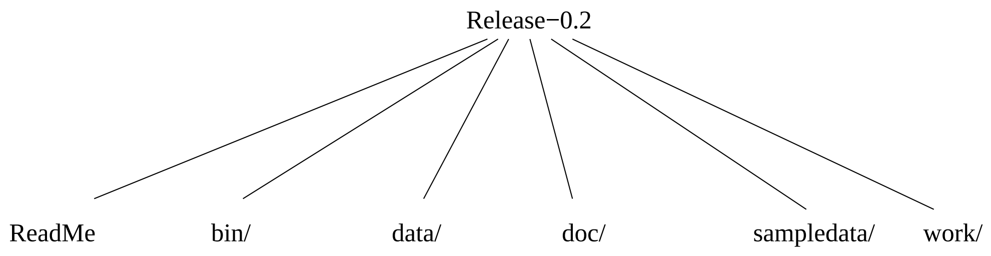

Figure C.1: Directory Tree.

Table C.2 shows the contents in each directory.

C.1.6 Executables

We build three groups of training and evaluation programs for users:

```
1. train3gram and test3gram
2. train_composite and test_composite, and
3. train_flat and test_flat.
```

They are used to train and evaluate regular topic-(in)dependent trigram models, syntactic/composite models with two types of syntactic features (and topic-dependent features) and flat models with no nested features for small vocabularies. To train

bin/	all executables.
data/	data (in the ME toolkit formats) used in training and evaluating models, such as N-gram files, index files and model parameter files.
doc/	this manual and release notes, etc.
sample_data/	sample data files that users can use to train some small ME models, and the answer files against which users can check their results.
work/	temperate files, such as initial models and files for debugging purposes.

Table C.2: Content of Directories.

a topic-dependent model, the argument file must have the name of a valid topic-dependent unigram file.

The command line for executing a program is

```
executable_file argument_file.
```

The argument file defines the model name, N-gram file names, training and test history-future $< x, w_i >$ tuple file names, etc.

C.2 Training an ME Model

There are three steps in training an ME model: data preparation, including feature selection; computation of feature expectations and update of model parameters. The first step is executed only once, whereas the last two steps need several iterations.

C.2.1 Data Preparation and Feature Selection

The sample data (*e.g.,* sentences, articles, etc.) need to be converted to the format that the toolkits can accept. Since the sources of samples vary tremendously, we cannot build a universal data preprocessing tool for all applications. Therefore, in the ME toolkits we provide the data formats to which users can convert their own data. For applications in natural language processing, this transformation can be done using Perl scripts. File formats will be introduced in Section C.4.1.

All features look like N-grams, whose format will be described in following sections. Users can select any feature that is observed[1] in the training data. Target expectations are usually empirical estimates of features after some smoothing, *e.g.,* Good-Turing discounting.

C.2.2 Computing Feature Expectations

An ME model is first initialized as a uniform model, *i.e.,* all model parameters $\alpha = 1$. The initialization is done by running

```
trainX argument-file -init
```

The initial uniform model must be renamed *old_para*, which is a reserved word in the toolkits. After the initial model is created, users can run

```
trainX argument-file.
```

to obtain the feature expectation file, whose name is defined in the argument file.

If the computation of feature expectations is distributed among many machines, each machine must have an individual argument file indicating which parts of the training data are used in this machine.

C.2.3 Updating Parameters

Since the computation of feature expectations may be distributed among several machines, several files of partial expectations may be generated. The following command is used to merge these files and also update model parameters.

```
UpdataAlpha old_parameters Ei_0 ... Ei_k new_parameters
```

The executable UpdataAlpha is obtained by running

```
g++ -o UpdataAlpha UpdataAlpha.cc
```

[1]Otherwise, that feature cannot be trained.

in the directory src/.

It should be noted here that steps C.2.2 and C.2.3 need to be executed for many iterations to train an ME model. Training programs will display the divergence between model expectations and target expectations of features. When this divergence is very small, the model is converged.

C.3 Computing Probabilities Using ME Models (Evaluation)

The ME toolkits support computation of the log-probability for a future token given the history using the ME model generated. The test data need to be converted to the format of the toolkits. This procedure is similar to the training. An argument file, whose format will be introduced later, needs to be generated to specify the setup of evaluation. Log-probabilities can be computed by running

```
testX test-arguments.
```

The log-probability is created in an ASCII file, whose name is defined in the argument file.

C.4 File Formats

In this section, we describe the file formats to follow when using the ME toolkits. We also show some real examples of these files.

C.4.1 Model Parameter File

We use the parameter format described in Ristad (1997) to represent model parameter files. If the model has V tokens, M marginal features (including topic-dependent features) and C conditional ones, then the model looks like

```
begin.constraints V C+M
begin.marginal M
```

```
1 alpha_1 a_1
...
M alpha_M a_M

end.marginal
begin.conditional M

M+1 alpha_{M+1} a_{M+1}
...
M+c alpha_{M+C} a_{M+C}

end.conditional
end.constraints
```

where alpha's are model parameters and a's are the corresponding target expectations.

The model is initialized as a uniform one, with the initial values of alpha's equal to one. The target expectations should be the counts (after some discounting) of features over the number of training samples. For example, the expectation of a bigram feature $g(w_1, w_2)$ can be $\frac{\#[w_1,w_2]}{\#[\text{training data}]}$.

C.4.2 N-gram File

All N-gram files have the common properties of

- one N-gram per line,
- sorted by N-grams in the increasing order, and
- no lines with zero counts.

Unigram File

A unigram file contains $\langle w_i, c(w_i)\rangle$ pairs, where $c(w_i)$ is the count of word w_i. A part of a unigram file may look like

```
...
20  37
21  40
```

```
24  1
...
```

Bigram File

A bigram file contains a list of $\langle w_i, w_j, c(w_i, w_j)\rangle$ triples. An example of bigram files may look like

```
...
13  41  20
13  934 3
14  2   10
...
```

Trigram File

A trigram file contains a list of $\langle w_i, w_j, w_k, c(w_i, w_j, w_k)\rangle$ 4-tuples. It may look like

```
...
10  25  342 7
10  34  1   2
11  7   43  4
...
```

Topic-dependent Unigram File

A topic-dependent unigram file contains a list of $\langle index, t, w_i, c(t, w_i)\rangle$, where t is a topic and $index$ is the index of the topic feature $g(t, w_i)$. This file must be sorted by t and w_i. It may look like

```
...
10  3   34  7
11  3   121 2
12  4   52  4
...
```

C.4.3 Discounting File

As we have already mentioned, target expectations of features should be smoothed from their empirical counts. To make the smoothing procedure easy, we assume that the target expectations of N-grams (from the same information source) with the same counts should be the same after discounting. The training programs will read the discounting file that indicates the value d for count c after discounting. Of course, users may set their own target expectations directly in their parameter files, as we will introduce in Section ??. The discounting file has the format of

```
M T
0 d1,1 d1,2 ... d1,T
...
0 dN,1 dN,2 ... dN,T
```

where M is the number of feature types, T is the maximum threshold for discounting, and di,j is the discounted value of the count j for the i^{th} kind of feature. An example of the discount file for trigram models is shown below.

```
3   5
0   0.7     1.7     2.7     3.7     4.7
0   0.5     1.5     2.6     3.6     4.7
0   0.3     1.4     2.5     3.6     4.7
```

C.4.4 History Equivalence Class File (By Information Source)

As we have described in preceding sections, the constraints of an ME model may come from M sources. We create history equivalence classes according to each information source. For example, in the syntactic model, we have three information sources: head-words, non-terminal labels and two preceding words. The corresponding history equivalence classes are pairs of h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, w_{i-2}, w_{i-1}, etc. The history equivalence class file contains tokens in the history equivalence class and the history equivalence class index. For example, the head word history equivalence class file contains tuples of $\langle h_{i-2}, h_{i-1}, index\rangle$. A real file looks like:

```
...
2   3   42
2   6   43
2   40  44
...
```

C.4.5 History File

Training data can be split into parts so that the model can be trained on different machines in parallel. Each part of training data is represented by a history file that will be introduced in this section, and a history-future tuple file that will be introduced in the following section. It is worth noting here that all files introduced in preceding sections are universal for the whole training data, whereas files introduced in this section and following sections are related to the parts of training data.

The history file contains history equivalence class indices (of different information sources) and its own index. This file is slightly different for trigram models and syntactic models. For trigram models, the history file has only two columns, which indicate the history equivalence class index for bigram w_{i-2}, w_{i-1} and the history index, respectively. For syntactic models, the file has four columns, which look like

```
...
102 422 13  59
103 423 12  60
103 425 14  61
...
```

The first three columns are the indices for history equivalence classes w_{i-2}, w_{i-1}, h_{i-2}, h_{i-1}, nt_{i-2}, nt_{i-1}, respectively, and the last one is the index of the history itself. Figure C.2 illustrates the relation between the history file and history equivalence class files.

C.4.6 Training Tuple File

The tuple file looks like a bigram file. It contains tuples of $\langle x, w_i, c(x, w_i)\rangle$. A tuple file may look like

```
...
59  93  7
59  323 4
60  26  1
...
```

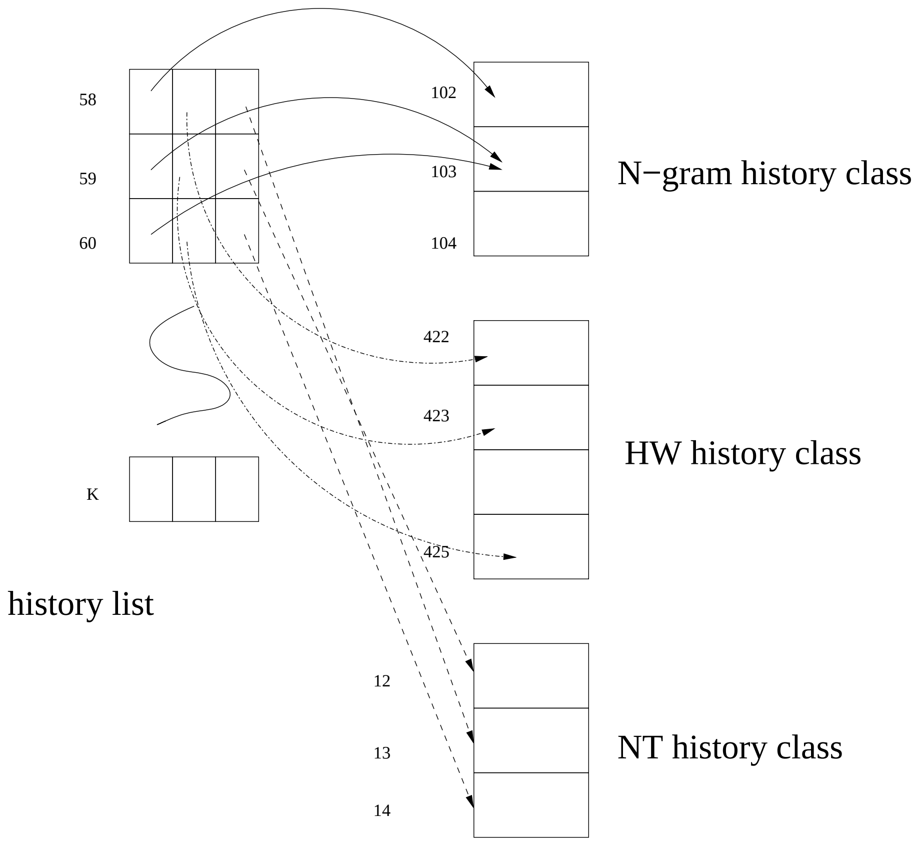

Figure C.2: Relation between the history file and history equivalence class files.

C.4.7 History Equivalence Class File (By Order)

To train ME models (*e.g.,* syntactic model) hierarchically, users need to create history equivalence classes $w_{i-1}, h_{i-1}, nt_{i-1}$ and words following them in the training data. The history equivalence class file (by order) is similar to the history file; it contains four columns, which indicate $w_{i-1}, h_{i-1}, nt_{i-1}$ and history equivalence class index, respectively. A real file for the syntactic/composite model may look like

```
...
9    13   4    101
9    20   7    102
10   6    1    103
...
```

C.4.8 Tuple File for History Equivalence Classes and Futures

This file contains tuples of history equivalence class (by order) index, future word and their count, whose format is similar to that of the training tuple file.

History and tuple files must be stored in one directory whose name is set in the argument file. The files must be named as follows:

```
history.part0 - history.partK, and
tuple.part0 - tuple.partK
```

if the training data has $K+1$ parts. Similarly, history equivalence class files and corresponding tuple files must be named

```
historyclass.part0 - historyclass.partK, and
tupleclass.part0 - tupleclass.partK
```

C.4.9 Test Tuple File

In the test tuple file, the first line indicates the number of test tuples. This number must be equal to or smaller than the real number of test tuples. If this number is less than the real size, then the evaluation program will read tuples up to this number. A test tuple file looks like

`number_of_tuples`

...

$0\ w_i\ w_{i-2}\ w_{i-1}\ 1$

...

for trigram models, and it looks like

`number_of_tuples`

...

$k\ w_i\ w_{i-2}\ w_{i-1}\ nt^1_{i-2}\ nt^1_{i-1}\ h^1_{i-2}\ h^1_{i-1}\ \rho(nt^1_{i-2}, nt^1_{i-1}, h^1_{i-2}, h^1_{i-1}) \cdots nt^k_{i-2}\ nt^k_{i-1}\ h^k_{i-2}\ h^k_{i-1}$
$\rho(nt^k_{i-2}, nt^k_{i-1}, h^k_{i-2}, h^k_{i-1})$

...

for syntactic models, where k is the number of candidate parses for w_i, and ρ is the likelihood of candidate parses.

C.4.10 Argument File

The argument files set environments for training and test programs, and for indicating the input and output.

Training Argument File

Here is a sample training argument file for trigram models.

```
10000 vocab_size
../data/train.txt.gt good_turing_discount
../data/train.txt.his N-gram_history_set
../data/train.txt.1gram unigram_file
../data/train.txt.2gram bigram_file
1               bi_cutoff
../data/train.txt.3gram trigram_file
2               tri_cutoff
topic_unigram_file *         topic-files
../data  tuple-directory
0               starting_part_of_training_data
9               ending_part_of_training_data
../data/train.txt.expect         expectation_file
../data/train.txt.model     new_model
10      number_of_iterations
```

Each line must have exactly two columns, even though the second one is simply for comments. If topic_unigram_file is valid, then the model is a topic-dependent model; otherwise, it is a topic-independent model.

In the example above, the initial model is saved in ../data/train.txt.model. As already explained, it should be renamed *old_para* before the first iteration of training.

Users can find more sample argument files

```
train_composite.argu, and
train_flat.argu
```

in the Release-0.2/sampledata/ directory.

Test Argument File

Here is a sample test argument file for trigram models.

```
60860 vocab_size
13 Unknown
../data/train.txt.his history_name
../data/train.txt.1gram unigram_file
../data/train.txt.2gram bigram_file
1              bi_cutoff
../data/train.txt.3gram trigram_file
2              tri_cutoff
bn.dev-test96.tuple testtuple
../data/train.txt.model     new_model
bn.dev-test96.ppl -logprob_for_test_tuples
```

Users can find more sample test argument files

```
test_composite.argu, and
test_flat.argu
```

in the Release-0.2/sampledata/ directory.

C.5 Advanced Topics

In this section, we describe for advanced users some hacks using the ME toolkits.

C.5.1 Reducing the Number of Iterations

N-gram models need only a few iterations to converge, and each iteration also takes a very short time. However, models with non-nested features, such as the syntactic model, need many iterations to converge, and each one takes a very long time. An efficient way of reducing the number of iterations for models with non-nested features is to train the N-gram model, the head-word N-gram model, *etc.*, first and then use the parameters of these models as the initial values of the target model with non-nested features. For example, the trigram models need about 20 iterations to converge, and the syntactic models need even more iterations if the process is started from the uniform model. However, the syntactic models can be converged in about 5 iterations if the process is started from the model whose parameters are set by the trained N-gram model, head-word N-gram model and non-terminal label N-gram model.

C.5.2 Add or Drop features

To add or drop features, users need to edit N-gram files, set correct target expectations and then run

```
train3gram train3gram.argu init
```

to create a new parameter file.

Users should NOT add or drop features directly from the parameter file.

C.5.3 Using Other Smoothing Methods

Users can change the last column of the initial model and set target expectations using their own smoothing methods.

C.5.4 High Order Features

The current ME toolkits do not directly support features of order 4 or above. However, the toolkits can be used to train and evaluate models with some kinds of order 4 features. For example, to train a model with features $g(nt_{i-3}, nt_{i-2}, nt_{i-1}, nt_i)$ and $g(nt_{i-2}, nt_{i-1}, nt_i)$, we can define a super non-terminal label set

$$NT^2 = NT \times NT + NT$$

where NT is the original non-terminal label set. Then both $g(nt_{i-3}, nt_{i-2}, nt_{i-1}, nt_i)$ and $g(nt_{i-2}, nt_{i-1}, nt_i)$ are order 3 features defined on NT^2. We reduce the order of features by this means. However, this trick does not apply if the size of the token set is large, *e.g.* a vocabulary.

C.6 Exercises

We provide some sample training and test data from *Jane Eyre* for users to learn how to use our toolkits. We recommend three exercises and provide answers for these exercises. All data are stored in directory sampledata/.

C.6.1 Building a Trigram Model

To build a trigram model, users can use the training data in je.training. Then, users can evaluate the ME model by computing the trigram probabilities of the test words in je.test.

The sample model (of 20 iterations) is je.model.3gram; the log probability file is je.ppl.3gram.

C.6.2 Building a Flat Model

Here we explain how to train an ME model to predict consonant / vowel of the next character (without looking at the character) using 5 preceding characters independently. The model looks like

$$p(y_i|c_{i-5},\cdots,c_{i-1}) = \frac{\alpha_{y_i}^{g(y_i)} \cdot \alpha_{c_{i-5},y_i}^{g(c_{i-5},y_i)} \cdots \alpha_{c_{i-1},y_i}^{g(c_{i-1},y_i)}}{z(c_{i-5},\cdots,c_{i-1})}$$

where y_i is either c or v. Users can use the same training and test data above. The sample model is je.model.cv, and the results are saved in je.cv.results.

C.6.3 Building a Skipped N-gram Model

Users can build and evaluate the skipped N-gram model

$$\begin{aligned} & p(w_i|w_{i-3},w_{i-2},w_{i-1}) \\ = \; & \frac{1}{z}\alpha_{w_i}^{g(w_i)} \cdot \alpha_{w_{i-1},w_i}^{g(w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_{i-1},w_i}^{g(w_{i-2},w_{i-1},w_i)} \cdot \alpha_{w_{i-2},w_i}^{g(w_{i-2},w_i)} \cdot \alpha_{w_{i-3},w_{i-2},w_i}^{g(w_{i-3},w_{i-2},w_i)}. \end{aligned}$$

using the same training and test data above. The sample model is je.model.skipped and the results are saved in je.ppl.skipped.

C.7 Troubleshooting

Most of the problems that users encounter in using the ME toolkits are related to invalid files. Here is the checklist for troubleshooting:

- Are files listed in the correct order in the argument file?

- Are files in correct formats?
- Are data in files sorted in the correct way?
- Are data files compatible? (Do indices agree with each other?)
- Contact junwu@clsp.jhu.edu

C.8 Update

Our toolkits are far from perfect right now. Users can check the update information for these toolkits from

`http://www.clsp.jhu.edu/junwu/METK.html`.

Bibliography

[Bellegarda, 1998] Bellegarda, J. R. (1998). Exploiting Both Local and Global Constraints for Multispan Statistical Language Modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* pp. 677–680, Seattle, USA.

[Berger & Printz, 1998] Berger, A. & Printz, H. (1998). Recognition Performance of a Large-Scale Dependency Grammar Language Model. *Proceedings of International Conference of Spoken Language Processing (ICSLP98)* pp.2083-2086. Sydney, Australia.

[Berger, Della Pietra & Della Pietra, 1996] Berger, A., Della Pietra, S. & Della Pietra, V. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics.* Vol 22, No.1, pp 39-72.

[Byrne, 2001] Byrne, W. (2001). The JHU March 2001 Hub-5 Conversational Speech Transcription System. In *Proceedings of the NIST LVCSR Workshop,* NIST.

[Charniak, 2000] Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. *Proceedings of NAACL-2000*, Seattle, USA.

[Chelba *et al.*, 1997] Chelba, C., Engle, D., Jelinek, F., Himenez, V., Khudanpur, S., Mangu, L., Printz, H., Ristad, E., Rosenfeld, R., Stolcke, A. & Wu, D. (1997). Structure and Performance of a Dependency Language Model. *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, pp. 2775–2778.

[Chelba & Jelinek, 1998] Chelba, C. & Jelinek, F. (1998). Exploiting Syntactic Structure for Language Modeling. *Proceedings of the COLING-ACL Meeting.* pp. 225–231, Montreal, Canada.

[Chelba & Jelinek, 1999] Chelba, C. & Jelinek, F. (1999). Recognition Performance of a Structured Language Model. *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, pp. 1567–1570. Rhodes, Greece.

[Chelba, 2000] Chelba, C., Advisor Jelinek, F. (2000). Exploiting Syntactic Structure for Natural Language Modeling. Ph.D. Dissertation, The Johns Hopkins University.

[Chen & Rosenfeld, 1998] Chen, S. & Rosenfeld, R. (1998). Topic Adaptation for Language Modeling Using Unnormalized Exponential Models. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* pp. 681–684, Seattle, WA.

[Chen & Goodman, 1999] Chen, S. & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling *Computer Speech and Language.* Vol. 13, No. 4, pp. 359-393.

[Chen & Rosenfeld, 2000] Chen, S. & Rosenfeld, R. (2000). A Guassian Prior for Smoothing Maximum Entropy Methods. *Technical Report CMU-CS-99-108*, Carnegie-Mellon University.

[Clarkson & Robinson, 1997] Clarkson, P. & Robinson, A. (1997). Language Model Adaptation Using Mixtures and an Exponentially Decaying Cache. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 799–802, Munich, Germany.

[Csiszár, 1975] Csiszár, I. (1975). I-Divergence Geometry of Probability Distributions and Minimization Problems. *The Annals of Statistics.* Vol. 3, No 1, pp.146-158.

[Csiszár, 1989] Csiszár, I. (1989). A Geometric Interpretation of Darroch and Ratcliff's Generalized Iterative Scaling. *The Annals of Statistics.* Vol. 17, No.3, pp.1409-1413.

[Csiszár, 1991] Csiszár, I. (1991). Why Least Squares and Maximum Entropy? An Axiomatic Approach to Inference for Linear Inverse Problems. *The Annals of Statistics.* Vol. 19, No. 4, pp. 2032-2056.

[Darroch & Ratcliff, 1972] Darroch, J. N. & Ratcliff, D. (1972). Generalized Iterative Scaling for Log-Linear Models. *The Annals of Mathematical Statistics*, vol. 43, No. 5, pp. 1470-1480.

[Della Pietra, Della Pietra & Lafferty, 1997] Della Pietra, S., Della Pietra, V. & Lafferty, J. (1997) Inducing Features of Random Fields, *IEEE Trans. on Pattern Analysis and Machine Intelligence.* vol.19, No.4, pp280-393.

[Florian & Yarowsky, 1999] Florian, R. & Yarowsky, D. (1999). Dynamic nonlocal language modeling via hierarchical topic-based adaptation. *Proceedings of ACL.* pp. 167-174, University of Maryland, USA.

[Godfrey *et al.*, 1992)] Godfrey, J., Holliman, E. & McDaniel, J. (1992). SWITCHBOARD: Telephone Speech Corpus for Research and Development. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* pp. 517-520. San Francisco, USA.

[Good, 1953] Good, J. & Turing, A. (1953). The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika.* vol. 40, pp. 237-264.

[Iyer & Ostendorf, 1996] Iyer, R. & Ostendorf, M. (1996). Modeling Long Range Dependence in Language, *Proceedings of ICSLP'96*, pp 236-239, Philadelphia, USA.

[Jaynes, 1982] Jaynes, E. T. (1982). On the rationale of Maximum Entropy Methods. *Proceedings of the IEEE*, Vol. 70, No.9 pp.939-952.

[Jelinek & Mercer, 1980] Jelinek, F. & Mercer, R. (1980). Interpolated estimation of Markov source parameters from sparse data. *Proceeding of Workshop on Pattern Recognition in Practice.* pp. 381-397, North Holland, Amsterdam.

[Jelinek, 1990] Jelinek, F. (1990). Self-organized Language Modeling for Speech Recognition. *Readings of Speech Recognition* edited by Waibel and Lee, pp.450-506. Morgan Kaufmann.

[Jelinek, 1997] Jelinek, F. (1997). Maximum Entropy Probability Estimate and Language Models. Ch 13, in *Statistical Methods For Speech Recognition.* MIT Press.

[Katz, 1987] Katz, S. (1987). Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing.* vol. 35, No.3, pp.400-401.

[Kneser & Ney, 1995] Kneser, R. & Ney, H. (1995). Improved smoothing for mgram language modeling, *Proceedings of ICASSP'95*, pp. 181-184, Detroit, USA.

[Kneser *et al.*, 1997] Kneser, R., Peters, H. & Klakow, D. (1997). Language Model Adaptation Using Dynamic Marginals. *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, pp. 1971-1974, Rhodos, Greece.

[Khudanpur & Wu, 1999] Khudanpur, S. & Wu, J. (1999). A Maximum Entropy Language Model to Integrate N-Grams and Topic Dependencies for Conversational Speech Recognition. *Proceedings of ICASSP'99*, pp. 553-556, Phoenix, USA.

[Khudanpur & Wu, 2000] Khudanpur, S. & Wu, J. (2000). Maximum Entropy Techniques for Exploiting Syntactic, Semantic and Collocational Dependencies in Language Modeling. *Computer Speech and Language* Vol.14, No.5, pp.355-372.

[Kuhn & De Mori, 1990] Kuhn, R., DeMori, R. (2990). A Cache-Based Natural Language Model for Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No.6, pp. 570-583.

[Lafferty & Suhm, 1996] Lafferty J. & Suhm B. (1996). Cluster Expansions and Iterative Scaling for Maximum Entropy Language Models. In *Maximum Entropy and Bayesian Methods*, Edited by Hanson & Silver, Kluwer Academic Publishers.

[Lafferty *et al.*, 1992] Lafferty, J., Sleator, D. & Temperly, D. (1992). Grammatical Trigrams: A Probabilistic Model of Link Grammar. *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, USA.

[Marcus, Santorini & Marcinkiewicz, 1993] Marcus, M., Santorini, B. & Marcinkiewicz, M. (1993). Building a Large Annotated Corpus of English: the Penn Treebank. *Computational linguistics* Vol 19, No.2, pp. 313-330.

[Martin, Liermann & Ney, 1997]Martin, S. C., Liermann, J. & Ney, H. (1997). Adaptive Topic-Dep. Language Modeling Using Word-Based Varigrams. *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, pp. 1447–1450, Rhodos, Greece.

[Mohri, Pereira & Riley, 2002] Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language.* vol. 16, No.1, pp 69-88.

[Ney, Martin & Wessel, 1997] Ney, H., Martin, S. & Wessel, F. (1997) Statistical Language Modeling Using Leaving-one-out. *Corpus-Based Methods in Speech and Language.* pp. 174-207, Kluwer.

[Numerical Recipes, 2002] Numerical Recipe Inc, Numerical Recipe available at "http://www.nr.com".

[Ratnaparkhi, 1996] Ratnaparkhi,A. (1996). A Maximum Entropy Part-Of-Speech Tagger. *Proceedings of Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, USA.

[Ristad, 1997] Ristad, E. (1997). Manual for Maximum Entropy Modeling Toolkit. "ftp://ftp.cs.princeton.edu/pub/package/memt".

[Roark, 2001] Roark, B. (2001). Probabilistic Top-down Parsing and Langauge Modeling. *Computational Linguistics*, vol 27, No. 2, pp 249-276.

[Rosenfeld, 1994] Rosenfeld, R. (1994). Adaptive Statistical Language Modeling: A Maximum Entropy Approach. *Computer Science Technical Report No. CMU-CS-94-138*, Carnegie Mellon University.

[Shore & Johnson, 1980] Shore, J. & Johnson, R. (1980). Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, vol 26, No.1, pp. 26-37.

[Sleator & Temperley, 1993] Sleator & Temperley(1993) Parsing English with a Link Grammar. In *Third International Workshop on Parsing Technologies*, Tilburg & Durbuy (Netherlands/ Belgium).

[Stolcke, *et al*, 1996] Stolcke, A., Chelba, C., Engle, D., Jimenez, V., Mangu, L., Printz, H., Ristad, E., Rosenfeld, R., & Wu, D. (1996). Dependency langaug modeling. *Technical Report in The Johns Hopkins Unversity CLSP Summer Workshop.*, Baltimore, USA.

[Stolcke, 1996] Stolcke, A. (1996). The SRI Language Modeling Toolkit, SRI International. Available at "http://www.speech.sri.com/projects/srilm"

[Wu & Khudanpur, 1999] Wu, J. & Khudanpur, S. (1999). Combining Nonlocal, Syntactic and N-Gram Dependencies in Language Modeling. In *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech).* pp. 2179-2182. Budapest, Hungary.

[Wu & Khudanpur, 2000a] Wu, J. & Khudanpur, S. (2000). Syntactic Heads in Statistical Language Modeling. *Proceedings of ICASSP2000.* pp. 1699-1702, Istanbul, Turkey.

[Wu & Khudanpur, 2000b] Wu, J. & Khudanpur, S. (2000). Efficient Training Methods for Maximum Entropy Language Modeling. *Proceedings of ICSLP2000.* pp. 114-117, Beijing, China.

[Wu & Khudanpur, 2002] Wu, J. & Khudanpur, S. (2002). Building A Topic-Dependent Maximum Entropy Language Model for Very Large Corpora. *Proceedings of ICASSP2002*, pp. 777-780. Orlando, USA.

[Young *et al.*, 1995] Young, S., Jansen, J., Odell, J., Ollasen, D. & Woodland, P. (1995). The HTK Book (Version 2.0). Entropic Cambridge Research Laboratory.

Curriculum Vitae

Education

Ph.D. candidate	1996 - Present	Dept. of Computer Science, Johns Hopkins University. Expected Date of Graduation: Summer 2002.
M.S.E	1998	Johns Hopkins University, Computer Science.
M.E	1993	Tsinghua University (China), Electronic Engineering.
B.E	1989	Tsinghua university (China), Computer Science.

Professional Experience

September 1996-Present	**Research Assistant** in Center for Speech and Language Processing, Johns Hopkins University. Participated in the NSF STIMULATE project. Exploiting non-local dependencies in language modeling for speech recognition. Proposed maximum entropy language models with syntactic and topic dependencies (with Sanjeev Khudanpur). Successfully applied this model to speech recognition and obtained significantly improvement in performance. Proposed fast training algorithms for maximum entropy models. Proposed a classifier combination method for part-of-speech tagging (with Eric Brill). **Teaching Assistant** in courses Natural Language Processing, Computer Architecture, Computer Systems and Computer Networks, Johns Hopkins University.
June - August 1997	**Intern**, AT&T Labs-Research, Florham Park, NJ. Participated in the development of a speech recognition system for telephone dialogues. (Mentors: Roberto Pieraccini and Esther Levin.)

July 1993 - August 1996	**Lecturer and Research Associate**, Department of Electronic Engineering, Tsinghua University, China. Principle investigator of the project on telephone speech recognition of Mandarin Chinese sponsored by Nortel Research. Individual investigator of the project on the dictation machine sponsored by National 863 Foundation of China.
May 1993 - May 1995	**Part time consultant**, Avante Electronics Corp., China. Worked as the project leader for the JANET system, a large vocabulary Mandarin Chinese speech recognition and handwriting recognition system.
July 1989 - August 1991	**Software Engineer** in Chinese Educational Electronics Corporation, China.

Honors and Awards

The ELSNET Prize for the Best Student Paper of Eurospeech99 (1999).

Honor for Excellent Young Faculty Members, by Tsinghua, China (1996).

Honor for Excellent Young Faculty Members, by Tsinghua, China (1995).

Best Paper Awards of the 3rd National Conference on Man-Machine Speech Communication, China (1994).

Excellent Graduate Awards, by Tsinghua (1993).

Chia-Chiao Lin Applied Mathematics Awards, Tsinghua University (1993).

Motorola fellowship, Tsinghua University, China (1992).

Research Interests

Speech Recognition,

Natural Language Processing,

Information Retrieval,

Machine Learning,

Statistical Modeling.

Affiliation

Member of IEEE, Society of Signal Processing.

Member of International Speech Communication Association (ISCA).

Selected Publications

1. **Jun Wu** and Sanjeev Khudanpur, ‘Building A Topic-Dependent Maximum Entropy Language Model for Very Large Corpora‘”, In Proceedings, ICASSP2002, Orlando, May, 2002.

2. Woosung Kim, Sanjeev Khudanpur and **Jun Wu**, “Smoothing Issues in the Structured Language Model,” Eurospeech 2001, September 2001, Denmark.

3. **Jun Wu** and Sanjeev Khudanpur, “Efficient Training Methods for Maximum Entropy Language Modeling”, Proceedings of ICSLP2000, Vol.3. pp 114-117, Oct. 2000, Beijing, China

4. Sanjeev Khudanpur and **Jun Wu** “Maximum Entropy Techniques for Exploiting Syntactic, Semantic and Collocational Dependencies in Language Modeling”. Computer Speech and Language, pp. 355-372, Oct. 2000.

5. **Jun Wu** and Sanjeev Khudanpur, “Syntactic Heads in Statistical Language Modeling”, Proceedings of ICASSP2000, pp. 1699-1702, June 4-9 2000, Istanbul, Turkey.

6. **Jun Wu** and Sanjeev Khudanpur, “Combining Nonlocal, Syntactic and N-Gram Dependencies in Language Modeling”. Proceedings of Eurospeech’99, vol 5, pp2179-2182, September 6-10, 1999, Budapest, Hungary. **Winner Eurospeech99 ELSNET Best Student Paper Award.**

7. Sanjeev Khudanpur and **Jun Wu**, “A Maximum Entropy Language Model to Integrate N-Grams and Topic Dependencies for Conversational Speech Recognition”. Proceedings of ICASSP’99, pp. 553-556, March 14-19, 1999, Phoenix.

8. Eric Brill and **Jun Wu**, "Classifier Combination for Improved Lexical Disambiguation", Processings of COLING-ACL'98, pp 191-195, August 10-14, 1998, Montreal Canada.

9. **Jun Wu** and Zuoying Wang, "Entropy of Chinese and the Perplexity of the Language Models", ACTA Electronica Sinica, v24 n10 Oct 1996, p69-71.

10. Zuoying Wang, **Jun Wu**, et al., "Methods Towards the Very Large Vocabulary Chinese Speech Recognition", Eurospeech' 95, pp.215-218, 1995.9, Madrid, Spain.

11. **Jun Wu**, Zuoying Wang, Jiasong Sun, Jin Guo, "Chinese Speech Understanding and Spelling-Word Translation Based on the Statistics of Corpus", ICSLP'94 , September 18-22, Yokohama, Japan.

12. **Jun Wu**, Jiasong Sun, Huizhong Yang and Yan Zhang, "IBM OS/2 Programming Guide" (Translation from English edition to Chinese edition), Tsinghua Press, 1994.

Other Publications

13. **Jun Wu**, Zuoying Wang and Jiasong Sun, "A Method of Speech Understanding for Chinese Speech Recognition", Chinese Software Journal, Special Issue, Oct 1996, pp188-193

14. **Jun Wu**, et al., "Entropy for Chinese Text", Chinese Information Journal, Vol. 6 N.2, pp. 17-23, 1996.

15. **Jun Wu**, Z. Wang, Y. Feng, "Automatic Classification of Chinese Texts", Chinese Information Journal, Vol. 5 N.4, pp. 34-41, 1995.

16. **Jun Wu**, Z. Wang, Y. Ren, "Stochastic Language Models for Chinese Speech Recognition Based on Chinese Spelling", 1994 International Symposium on Speech, Image Processing and Neural Networks, pp. 674-677., April 1994, HongKong.

17. **Jun Wu**, et al., "Large Vocabulary Telephone Speech Recognition System", The 4th National Conference on Man-Machine Speech Communication, 1996, 10, Beijing China.

18. **Jun Wu**, Z. Wang, et al., "A Strategy for Large Vocabulary Continuous Speech Recognition", Proc. of ICCC, Singapore, June, 1996.

19. **Jun Wu**, Z. Wang, J. Yu, "Research on the Perplexity of Language Models", The 2nd National Conference on Intelligent Computer Interface and Applications of Artificial Intelligence, pp. 169-174, July, 1995, Weihai, China.

20. **Jun Wu**, Z. Wang, J. Sun, J. Guo, "An Approach toward Speech Understanding and Spelling-word Translation", The 3rd National Conference on Man-Machine Speech Communication, pp. 214 -220, 1994.10, Chongqing, China.

21. Zuoying Wang, **Jun Wu**, et al., "JANET - A New Chinese Speech Recognition, Understanding, and Synthesis System", The 3rd National Conference on Man-Machine Speech Communication, pp.482-486, 1994.10, Chongqing, China.

22. **Jun Wu**, Z. Wang, Y. Ren, "A New Approach to Speech Understanding Based on Corpus", The 2nd National Conference of Computer Linguistics, pp. 96-101, Nov. 1993. Xiamen China.

23. **Jun Wu**, Z. Wang, "Stochastic Language Models for Eliminating Chinese Speech Recognition Error", The 1st National Conference of Intelligent Computer Interface and Application of Artificial Intelligence, pp. 233-238, July 22-24, 1993, Jinbo Lake, China.

Presentations/Seminars

Speechworks, Boston, July 2001.
AT&T Shannon Labs of Research, July 2001.
Speechworks, New York, June 2001.
IBM Almaden Research Labs, June 2001.

IBM T.J. Watson Research Labs, May 2001.

Johns Hopkins University, April 2001.

ICSLP, Beijing, October, 2000.

Nuance, August 2000.

SRI International, August 2000.

ICASSP, Istanbul, June, 2000.

IBM T.J. Watson Research Labs, April 1999.