AN EXPLORATION OF COMPOSITE LANGUAGE MODELING FOR SPEECH RECOGNITION

A Thesis
presented to
the Faculty of the Graduate School
at the University of Missouri-Columbia

In Partial Fulfillment
of the Requirements for the Degree
Master of Computer Science

by
Xiaolin Xie
Dr. Yunxin Zhao, Thesis Supervisor
May 2013
The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

AN EXPLORATION OF COMPOSITE LANGUAGE MODELING FOR SPEECH RECOGNITION

presented by Xiaolin Xie,

a candidate for the degree of master of computer science, and hereby certify that, in their opinion, it is worthy of acceptance.

____________________________________________
Dr. Yunxin Zhao

____________________________________________
Dr. Toni Kazic

____________________________________________
Dr. Jianlin Cheng
ACKNOWLEDGEMENTS

This work would not have been possible without support and help from many people. I want to express my deepest thanks to my advisor, co-workers, and friends.

First of all, I would like to express my sincerest appreciation to my advisor, Dr. Yunxin Zhao. Her exceptional mentoring and insightful guidance has been a tremendous asset in helping me complete this thesis. From the very first meeting, to the numerous meetings that followed, she spent a great deal of time and effort in advising my work by offering comments, ideas, and encouragement. This guidance and feedback kept me going and helped me overcome many challenges faced in completing this work.

In addition, I would like to thank Dr. Toni Kazic and Dr. Jianlin Cheng for their willingness to serve as my committee members and their invaluable suggestions and comments on my thesis.

This work built upon the research and the collaborative efforts of many individuals. My special thanks go to Ming Tan and Xiuzhen Huang for their development work on this project. Moreover, I would like to thank David Reid, for his kindly support and valuable feedback on my thesis.

Finally, my deepest gratitude goes to my wife, Yemin Huang, for her unconditional support on my work.
### TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................... ii  
TABLE OF CONTENTS ........................................................................................................ iii  
LIST OF TABLES .................................................................................................................... v  
LIST OF FIGURES .................................................................................................................. vi  
Abstract ................................................................................................................................ vii  

**Chapter 1**  Introduction ........................................................................................................ 1  
1.1. Speech Signal Processing .............................................................................................. 4  
1.2. Acoustic Modeling ........................................................................................................ 4  
1.2.1. HMMs in Speech Recognition .................................................................................. 5  
1.2.2. Pronunciation Dictionary ......................................................................................... 7  
1.3. Language Modeling ...................................................................................................... 8  
1.4. Decoding Search .......................................................................................................... 8  
1.5. Evaluation .................................................................................................................... 10  
1.6. Thesis Motivation and Organization ........................................................................... 11  

**Chapter 2**  Language Modeling .......................................................................................... 13  
2.1. Language Modeling in General ................................................................................... 15  
2.2. N-Gram Language Modeling ....................................................................................... 15  
2.3. Structured Language Modeling .................................................................................... 17  
2.4. Probabilistic Latent Semantic Analysis ....................................................................... 22  
2.5. A Syntactic, Semantic, and Lexical Language Model .................................................. 25  
2.6. Evaluating Language Models ....................................................................................... 27  

**Chapter 3**  Lattice Rescoring in Speech Recognition .......................................................... 29  
3.1. Two-pass Decoding ...................................................................................................... 29  
3.2. A* search ................................................................................................................... 31  
3.3. Lattice rescoring with A* search ................................................................................. 33  

**Chapter 4**  Confusion Network ............................................................................................ 36  
4.1. Introduction .................................................................................................................. 36  
4.2. Lattice to confusion network transformation ............................................................... 38  
4.3. Confusion network rescoring ....................................................................................... 43  

**Chapter 5**  Implementation, Experiments, and Results ......................................................... 47  
5.1. A distributed architecture ............................................................................................ 47  
5.2. Experimental Data Description .................................................................................... 49  
5.2.1. The ARPA WSJ Language Corpus ........................................................................ 49  
5.2.2. The Wall Street Journal lattices ............................................................................. 49  
5.2.3. Oracle WER .......................................................................................................... 50  
5.3. Experiments & Results ................................................................................................ 50  
5.3.1. Viterbi decoding ...................................................................................................... 50  
5.3.2. Lattice Grouping ..................................................................................................... 51  
5.3.3. Lattice rescoring with CLM and SLM .................................................................... 52  
5.3.4. Lattice to confusion network transformation ........................................................ 56  
5.3.5. Confusion network rescoring ................................................................................ 57  

**Chapter 6**  Discussion and Future studies ............................................................................ 62  
6.1. Discussion .................................................................................................................... 62
6.2. Future studies ................................................................. 63
Reference.................................................................................. 65
LIST OF TABLES

Table 5.1 Oracle WER of the WSJ lattices ................................................................. 50
Table 5.2 WER of decoding results of Viterbi decoder ............................................ 51
Table 5.3 WER of A* lattice rescoring ..................................................................... 53
Table 5.4 Details of WER (%) for lattice rescoring .................................................. 54
Table 5.5 Decoding results when CLM improves WER ........................................... 54
Table 5.6 Decoding results when CLM improves understanding ............................... 56
Table 5.7 WER of confusion networks ..................................................................... 57
Table 5.8 Oracle WER of pruned confusion networks ............................................. 58
Table 5.9 WER(%) of rescoring pruned confusion networks (equation 4.11) ......... 59
Table 5.10 WER(%) of rescoring pruned confusion networks (equation 4.12) ....... 59
Table 5.11 Oracle WER(%) of un-pruned confusion networks .................................. 60
Table 5.12 WER(%) of rescoring un-pruned confusion networks (equation 4.11) .. 60
Table 5.13 WER (%) of rescoring un-pruned confusion networks (equation 4.12) .. 61
LIST OF FIGURES

Figure 1.1 Communication theory view of speech recognition (Bahl et al., 1983) .... 2
Figure 1.2 Architecture of speech recognition systems (Jurafsky & Martin, 2008) .... 3
Figure 1.3 An example of HMM for a phone unit .................................................. 6
Figure 2.1 Noisy channel model (Brown et al., 1992) ............................................. 13
Figure 2.2 An example of a partial parse (Chelba & Jelinek, 2000) ...................... 17
Figure 2.3 A word-and-parse k-prefix (Chelba & Jelinek, 2000) ........................... 18
Figure 2.4 Structured language model operations (Chelba & Jelinek, 2000) ......... 19
Figure 2.5 Before an adjoin operation (Chelba & Jelinek, 2000) ......................... 20
Figure 2.6 Result of adjoin-left (Chelba & Jelinek, 2000) ................................... 20
Figure 2.7 Result of adjoin-right (Chelba & Jelinek, 2000) ................................... 20
Figure 2.8 The generative model of PLSA (Hofmann, 2001) ............................... 24
Figure 2.9 The composite language model (Tan et al., 2012) .............................. 26
Figure 3.1 An example of a word lattice .............................................................. 31
Figure 4.1 An example of a confusion network .................................................... 37
Figure 5.1 Architecture of the distributed application for lattice rescoring and
    confusion network rescoring ........................................................................... 48
An Exploration of Composite Language Modeling for Speech Recognition

Xiaolin Xie

Dr. Yunxin Zhao, Thesis Supervisor

Abstract

Language models are one of the most critical knowledge sources of automatic speech recognition (ASR) systems. In the past decades, many language models have been developed, and some have been proved useful and successful in speech recognition systems. However, almost all language models only capture one or two aspects of natural language. This study aims to investigate the effects of a syntactic, semantic, and lexical language model on speech recognition. In this study, we refer this language model as the composite language model (CLM). The parameters of the CLM in our study are distributed among hundreds of computer nodes in a supercomputer because they are too large to be stored in just one computer node. A distributed application has been developed to implement two speech rescoring techniques by using the CLM: lattice rescoring and confusion network rescoring. Experiments on a Wall Street Journal task have shown that using CLM to rescore word lattices and confusion networks have led to improvements in word accuracy over the commonly used trigram language model, with the latter offering a larger performance gain.
Chapter 1 Introduction

Speech is the most natural way that people communicate with each other. Years ago, due to computers’ inability of understanding human speech, the possible ways for people to control computer devices were limited to keyboards and mouse, which was inconvenient and inefficient in many cases. Automatic speech recognition (ASR) is the technology that transcribes speech into text. For the past decades, many studies have been put into this area, and a lot of progresses have been made. Nowadays, speech recognition is becoming more and more promising, and it is being used in a wide range of applications. For example, many customer telephone service systems use speech recognition to provide individual assistance to their customers. Global Positioning System (GPS) navigation systems take orders and send directions through speech. Siri, the intelligent personal assistant in iPhone, allows people to use voice to send messages, schedule meetings, place phone calls, and so on. Voice dictation, the transcription of monologue speech, is another typical application of speech recognition. For example, in order to assist people with disabilities, Zhao (2009) developed a speech recognition system to transcribe video teleconferencing speech conversation in telemedicine.

The communication theory view of speech recognition (Bahl, Jelinek, & Mercer, 1983) formulates speech recognition as a problem in telecommunication using the noisy channel model. A speaker transforms a word sequence into a speech waveform, and then the acoustic processor (AP) that is a signal processor converts the speech waveform into a sequence of feature vectors. After the word sequence
passes through the noisy channel, i.e., the speaker and the acoustic processor, some information is discarded or missing. The linguistic decoder takes a noisy string output from the noisy channel and recovers the underlying sentence to minimize the distortion in $\widehat{W}$ with respect to $W$.

The output of the acoustic processor is a sequence of vectors $O = o_1, o_2 \ldots o_t$. The goal of the linguistic decoder is to find a word sequence $W = w_1, w_2 \ldots w_n$ that matches the underlying sentence the best given the acoustic observation $O$. In order to minimize the error between the estimated word sequence and the underlying sentence, the word sequence $\widehat{W}$ is chosen so that

$$\widehat{W} = \operatorname{argmax}_W P(W|O) \quad (1.1)$$

According to the Bayes’ Rule,

$$P(W|O) = \frac{P(W) * P(O|W)}{P(O)} \quad (1.2)$$

Since the $P(O)$ does not depend on $W$, maximizing $P(W|O)$ is equivalent to maximizing $P(W) * P(O|W)$.

$$\widehat{W} = \operatorname{argmax}_W P(W) * P(O|W) \quad (1.3)$$

$P(W)$, the prior probability of the word sequence $W$, is estimated by a language model. $P(O|W)$, the observation likelihood of $O$ given a word sequence $W$, is computed using acoustic models.
The process of speech recognition primarily consists of three stages (Jurafsky & Martin, 2008). The first stage is speech signal processing and feature extraction. In this stage, speech waveform is sampled into frames, and then frames are transformed into feature vectors. In the second stage, likelihood scores of the vectors are computed by a set of well-trained acoustic models. In the third stage, the decoder takes the acoustic likelihood scores, combined with a language model and a pronunciation dictionary, to find the most likely sentence. Figure 1.2 (Jurafsky & Martin, 2008) represents the architecture of a speech recognition system.
1.1. **Speech Signal Processing**

The sound waves are produced by a series of changes in air pressure, which are caused by air passing the vocal organs of speakers. They are usually captured by microphone and converted to electrical signals. The analogy-to-digital conversion samples the sound waves at a certain clock rate and transforms the original sound waves into a digital speech waveform, which is the raw input of an ASR system.

The speech waveform has to be transformed into an appropriate feature representation before statistical decision methods can be applied to find the best matching word sequence. The signal processing and feature extraction stage in speech recognition converts the speech waveform into a sequence of vectors, and each vector represents the information in a small time window of the speech waveform. Commonly used features in speech recognition include: Linear Predictive Coefficient (LPC) (Atal & Schroeder 1967), Perceptual Linear Prediction (PLP) (Hermansky, 1990), and Mel Frequency Cepstral Coefficients (MFCC) (Mermelstein, 1976).

The above features are considered to be short-term stationary, and the temporal information of speech waveform is missing from those features. Thus, in practice, first-order and second-order time-derivatives are applied to these static features to capture the time dynamic information. In addition, the extracted features can be further transformed to produce discriminative and robust features for ASR systems.

1.2. **Acoustic Modeling**

Acoustic models describe the acoustic-phonetic characteristics of speech pronunciations. Hidden Markov model (HMM) has been widely used to model time
sequential data, and it is also the tool that is used to model the distribution of the acoustic vectors in speech recognition. In HMM, the speech waveform is considered as generated by a set of states with their evolutions governed by a Markov process.

### 1.2.1. HMMs in Speech Recognition

A HMM model is characterized by the following components (Rabiner & Juang, 1993).

1. A set of $N$ hidden states $Q = \{q_1, q_2 \ldots q_N\}$.

2. A set of distinct observation symbols that are drawn from some space $V$. The observation symbols correspond to the outputs of the Markov process. The distribution of observation symbols is determined by the hidden state.

3. The state transition probability matrix $A = \{a_{ij} \mid 1 \leq i, j \leq N\}$, where each $a_{ij} = P(q_{t+1} = j \mid q_t = i)$ represents the probability of moving from state $i$ to state $j$. For some pairs of $(i, j)$, $a_{ij} = 0$ which means that state $i$ cannot move to state $j$. The transition matrix denotes how states can be reached from each other.

4. A set of observation probability distributions $B = \{b_j(o_t) \mid 1 \leq j \leq N\}$ where $b_j(o_t)$ represents the distribution of observation symbols given the hidden state $q_j$.

5. The initial state distribution, $\pi = \{\pi_i \mid 1 \leq i \leq N\}$, where $\pi_i = P(q_1 = i)$. It denotes the prior probability distribution of the beginning hidden states.
The pronunciations of words are described by individual units called phones. In large-vocabulary continuous speech recognition (LVCSR), each phone is modeled by one HMM. The commonly used HMM model in acoustic modeling uses 3 emitting states and 2 non-emitting states to model one phone. The 3 emitting states generate the speech observation in a Markov process. The 2 non-emitting states are an entry state and an exit state that do not generate any observation. They are reached before and after the observation generation process, respectively.

The transitions between the five states are very constrained. It only allows a state to transit to itself (self-loop) or to the succeeding state where the self-loop transition allows a single state to last for a number of frames. The entry state and exit state are reached just once. The pronunciation of a phoneme varies hugely with the context, the speaker, and the acoustic environment.

![Figure 1.3 An example of HMM for a phone unit](image)

The acoustic observation probability distribution is modeled by a Gaussian Mixture Model (GMM). A multivariate Gaussian distribution $\mathcal{N}(u_j, \Sigma_j)$ for acoustic vectors given the $j$-th state can be represented as:

$$b_j(o) = \frac{1}{D} \frac{1}{(2\pi)^{D/2} |\Sigma_j|^{1/2}} \exp \left[-(o - u_j)^t \Sigma_j^{-1}(o - u_j) \right]$$

(1.4)

where $D$ is the number of dimensions of an acoustic vector and $u_j$ and $\Sigma_j$ are the mean vector and covariance matrix of the multivariate distribution. Usually, it is
assumed that dimensions in acoustic vectors are independent from each other for each mixture component.

The GMM for a sub-phone given the state \( q_j \) can be represented as:

\[
b_j(o) = \sum_{m=1}^{M} c_{jm} \frac{1}{(2\pi)^{D/2} |\Sigma_{jm}|^{1/2}} \exp \left[ -\frac{1}{2} (o - u_{jm})^{t} \Sigma_{jm}^{-1} (o - u_{jm}) \right]
\]

where \( M \) represents the number of mixture components, and \( c_{jm} \) represents the weights for the \( m \)-th Gaussian distribution, thus

\[
\sum_{m=1}^{M} c_{jm} = 1 \text{ and } c_{jm} \geq 0
\]

To summarize, a HMM model of each phoneme in speech recognition is parameterized by the following components (Jurafsky & Martin, 2008):

- \( Q = \{q_0, q_1, q_2, q_3, q_4\} \): Two non-emitting states and three emitting states
- \( A = \{a_{01}, a_{11}, a_{12}, a_{22}, a_{23}, a_{33}, a_{34}\} \): States transition probabilities where states can only transit to itself or the next succeeding state, and the two-non emitting states can be reached only once.
- \( B = \{b_j(o_t) | 1 \leq j \leq N\} \): The acoustic observation probability distribution given a hidden state is modeled by a GMM.

**1.2.2. Pronunciation Dictionary**

The phonetic alphabet is a set of symbols used to describe phoneme in words’ pronunciations. The International Phonetic Alphabet (IPA) developed by the International Phonetic Association is one of the most widely used phonetic alphabet, and the ARPAbet (Shoup, 1980) is another popular phonetic alphabet that uses ASCII symbols to represents phonemes in word pronunciations.
A pronunciation dictionary defines each word pronunciation with a sequence of phoneme symbols. The common used dictionary includes CELEX dictionary (Baayen et al., 1995), CMU Pronouncing Dictionary (CMU, 1993), and PRONLEX dictionary (LDC 1995). With the help of pronunciation dictionary, HMM models for phoneme can be concatenated to describe the HMM for a word.

1.3. Language Modeling

The prior probability $P(W)$ of a word sequence is estimated by a language model. Language models are also important to other natural language related fields, such as machine translation. There are different approaches to language modeling, and different language models have been built in the past decades. Chapter 2 discusses language modeling in details.

1.4. Decoding Search

If the way that a noisy channel distorts the source is known, one can find the correct sentence by running every possible sentence through the noisy channel and then selecting the best matching one (Jurafsky & Martin, 2008). Basically, this is the idea of how speech recognition systems search for the best hypothesis sentence. Since it is impossible to run through every sentence because of the infinite search space, a lot of candidates with low probabilities have to be pruned in the search process, which leaves a few candidates that have a good chance of matching the input speech waveform. The decoding stage combines the knowledge sources from both the acoustic model and the language model to prune less probable candidates and find the $\hat{W}$ such that
\[ \hat{W} = \arg \max_{\text{w}} P(W)P(O|W) \quad (1.7) \]

\( P(W) \) is a probability that ranges from 0 to 1, while \( P(O|W) \) is a probability density function that has a very large dynamic range. Because of the different scales of \( P(W) \) and \( P(O|W) \), we reweight the two factors by applying a language model weight to \( P(W) \) to adjust the relative effects of the language model and acoustic model in the search process:

\[ \hat{W} = \arg \max_{\text{w}} P(O|W)P(W)^{LM\_WEIGHT} \quad (1.8) \]

In practice, we use log probabilities because of the underflow problem of floating point numbers in computers.

\[ \hat{W} = \arg \max_{\text{w}} [\log(P(O|W)) + LM\_WEIGHT \times \log(P(W))] \quad (1.9) \]

In addition, the pronunciation of a long time-span word is often broken into several short time-span words, and the later case often has a higher probability because of its flexibility. In order to compensate for the long time-span word, an insertion penalty to each word is added to favor long time-span word:

\[ \hat{W} = \arg \max_{\text{w}} [\log(P(O|W)) + LM\_WEIGHT \times \log(P(W)) - N \times WP] \quad (1.9) \]

where \( N \) is the length of the word sequence \( W \), and \( WP \) is the penalty score for each word in the sentence.

In HMM, Viterbi algorithm (Viterbi, 1967) is used to find the most probable sequence of hidden states given a sequence of observations, which is also the algorithm used in speech recognition for decoding. In essence, Viterbi is a practice of dynamic programming (Ney & Ortmans, 1999). Dynamic programming that solves
problems in a bottom up manner works well for optimization problems in which sub problems share common problems.

In speech recognition, the decoding stage primarily consists of two steps. The first step is forward-extension in which possible paths are extended from time 0 to $T - 1$ where $T$ is the number of window frames of a speech waveform. During the extension, path scores are calculated by linearly combining the log acoustic scores and log language model scores up to the time $t$. Each word in the path will record its best previous word that leads to it. Different heuristic pruning methods are necessary for narrowing down the search space and speeding up the decoding. The second step is backtracking in which the path with the highest ending score is selected and its trace is found by recursively finding the best previous word for the current word.

1.5. Evaluation

The standard evaluation metric for speech recognition is word error rate (WER). The WER is calculated based on the minimum edit distance (Wagner & Fischer, 1974) between the hypothesis sentence and the reference sentence. A sequence of words can be transformed to another word sequence by a set of insertion, deletion, and substitution operations. The minimum edit distance between two sequences of words is the minimum number of operations needed for this transformation. Usually, one uses dynamic programming to find the best alignment between the hypothesis sentence and the reference sentence for the minimum edit distance. Once this best alignment is found, the WER is computed as follows:
\[
\text{WER} = \frac{\text{Insertions} + \text{substitutions} + \text{Deletions}}{\text{Total words in the reference sentence}} \times 100\% \tag{1.10}
\]

The sentence error rate (SER) is calculated as the percentage of sentences with at least one error

\[
\text{SER} = \frac{\text{number of sentences with at least one error}}{\text{total number of sentences}} \times 100\% \tag{1.11}
\]

1.6. Thesis Motivation and Organization

Although speech recognition is widely used in a lot of applications and there are many successful cases, its performance is still not perfect, and its accuracy needs to be improved. A language model is important to the performance of speech recognition. Although many language models have been built during the past decades and some of them have been proved useful and successful in some cases, most language models capture only one or two aspects of natural language. In this study, we focus on exploring innovative and effective language models for speech recognition. Specifically, we study a syntactic, semantic, and lexical language model, apply it to two rescoring techniques in speech recognition: lattice rescoring and confusion network rescoring, and examine its effects on the performance of speech recognition.

The rest of this thesis is organized as follows:

Chapter 2 discusses different language models. Firstly, two paradigms of language modeling are introduced, and the basic idea of language modeling is discussed. Next, different language models are presented, including n-gram language model, structured language model, probabilistic latent semantic analysis, and a syntactic, semantic, and lexical language model.
Chapter 3 discusses the rescoring technique of lattice rescoring in speech recognition. Firstly, a limitation of the Viterbi decoder is discussed, and two-pass decoding, a way to remedy the limitation of the Viterbi decoder, is introduced. Then, the two-pass decoding technique of lattice rescoring is introduced. The details of the A* algorithm for lattice rescoring are also presented.

Chapter 4 talks about another rescoring technique based on confusion network in speech recognition. Firstly, the discrepancy between the traditional MAP decoding approach and the standard performance metric WER in speech recognition is discussed. Next, confusion network as an approach to address this inconsistency is introduced. Finally, chapter 4 presents our attempts on confusion network rescoring.

Chapter 5 describes our implementation of lattice rescoring and confusion network rescoring, and presents the corresponding experiments and results on lattice rescoring and confusion network rescoring. Firstly, the distributed architecture of our application for lattice rescoring and confusion network rescoring is presented. Next, our experiments and results on lattice rescoring and confusion network rescoring are presented.

Chapter 6 briefly discusses the results in our experiments and outlines some directions for future studies.
Chapter 2 Language Modeling

The goal of language modeling is to build models to predict and estimate the probability of a word sequence. Language models are very important in speech recognition and other natural language related areas, such as machine translation (MT) and automatic spelling correction. Figure 2.1 (Brown, deSouza, Mercer, Pietra, & Lai, 1992) presents a model that has been used in ASR, machine translation, and automatic spelling correction. In this model, Y represents the observation, such as an acoustic observation O in ASR, a sentence or paragraph in foreign language in machine translation, or a sequence of characters from a typist in automatic spelling correction. Given an observation Y, all these tasks aim to find a word sequence W that maximizes the posteriori probability of W given Y.

\[
\hat{W} = \arg\max_W P(W|Y) = \arg\max_W \frac{P(W) \cdot P(Y|W)}{P(Y)}
\]  

(2.1)

Since the \(P(Y)\) does not depend on \(W\), maximizing \(P(W|Y)\) is equivalent to maximizing \(P(W) \cdot P(Y|W)\). The estimation of \(P(W)\) is computed by a language model.

\[
\hat{W} = \arg\max_W P(W) \cdot P(Y|W)
\]  

(2.2)

Figure 2.1 Noisy channel model (Brown et al., 1992)
Natural language contains rich information in many aspects, so the modeling of natural language requires knowledge sources from many disciplines. Generally speaking, three kinds of knowledge are critical to language modeling. Lexical knowledge concerns word definition and word orders in natural language; syntactic knowledge describes grammatical rules of natural language and structure of sentences; semantic knowledge deals with semantic meaning of documents (Zhang, 2005). How well a language model reflects knowledge in these aspects determines its quality. However, most language models capture only one or two aspects of natural language.

There are two primary paradigms towards language modeling. The first one approaches language modeling from the perspective of linguistics. It uses a set of linguistic rules to model natural language, such as Context-Free-Grammar (CFG) (Tomita, 1987). The second paradigm models natural language with statistical models. In practice, rule-based models do not work well due to two major reasons. Firstly, the natural language is too complex to be captured by a set of predefined rules. Secondly, it is common that grammatically incorrect or incomplete sentences exist in spoken language, which causes difficulties for rule-based language models to make predictions.

In the past decades, many statistical language models have been developed, and some of them have been proved useful and successful in a certain degree. This chapter discusses the following statistical language models: n-gram language model, structured language model, probabilistic latent semantic analysis, and a syntactic, semantic, and lexical language model.
2.1. Language Modeling in General

The primary goal of a language model is to estimate the probability of a word sequence \( P(W) \). For a given sequence of string \( W = w_1 w_2 \ldots w_n \), using the chain rule of probability, one can get

\[
P(W) = \prod_{k=1}^{n} P(w_k | w_1, w_2, \ldots, w_{k-1})
\]

(2.3)

The parameter space of \( P(w_k | w_1, w_2, \ldots, w_{k-1}) \) is too large for large \( k \) values. Thus, language models usually simplify the history \( W_{k-1} = w_1 w_2 \ldots w_{k-1} \) into an equivalence class determined by a function \( \varphi(W_{k-1}) \) (Chelba, 2000). By this simplification,

\[
P(W) \cong \prod_{k=1}^{n} P(w_k | \varphi(W_{k-1}))
\]

(2.4)

The studies in statistical language modeling primarily aim to find appropriate equivalence classifiers \( \varphi(W_{k-1}) \) and effective methods to estimate \( P(w_k | \varphi(W_{k-1})) \) (Chelba, 2000).

In addition, there are many words in a language. In order to further control the parameter space, language models also set the word vocabulary into a certain number of commonly used words the task corpora. Words out of the vocabulary are usually mapped into a special string, such as <unknown>.

2.2. N-Gram Language Modeling

N-gram models natural language according to the lexical order of words in corpora. In n-gram language modeling, the current word history \( W_{k-1} \) is simplified into the closest \( n-1 \) words, assuming that \( k \) is larger than \( n \). Thus, a n-gram language model can be presented by \( P(w_k | w_{k-n+1}, \ldots, w_{k-1}) \) where \( w_{k-n+1}, \ldots, w_{k-1} \) represents the closest \( n-1 \) words before the current word. For
example, in the trigram language model, the $W_{k-1}$ is simplified into a two words sequence $w_{k-2} \ w_{k-1}$, and two histories are considered as equivalent if they have the same last two words.

The parameters of a $n$-gram language model can be estimated by examining the relative frequency counts of the $n$-word sequence $w^k_{k-n+1}$ and the $n-1$ word sequence $w^{k-1}_{k-n+1}$ in training corpora, which is computed as:

$$P(w_k | w^{k-1}_{k-n+1}) = \frac{C(w^{k-1}_{k-n+1}w_k)}{C(w^{k-1}_{k-n+1})} \quad (2.5)$$

where $C(w^k_{k-n+1})$ represents the count of the word sequence $w^k_{k-n+1}$ in the training corpora. This method is an example of maximum likelihood estimation (MLE), and the resulting parameters training from this method maximizes the likelihood of the training corpora (Jurafsky & Martin, 2008).

Due to the limited amount of data in training corpora, many grammatically correct and semantically meaningful word sequences are not observed in the training corpora. The above maximum likelihood estimation method assigns zero probability to the word sequences that contain these unobserved word sequences, which is a problem when we apply the language model to some new data sets. Furthermore, the MLE method also produces poor estimates when the counts are non-zero but still small (Jurafsky & Martin, 2008).

In order to reconcile this problem, some smoothing methods are proposed to reassign probabilities among high and low count word sequences. For example, Laplace smoothing achieves this purpose by adding one count to each possible the $n$-word sequence.
2.3. Structured Language Modeling

The Structured Language Model (SLM) (Chelba & Jelinek, 2000) exploits the syntactic structure of sentences for word prediction. The basic idea of SLM is that syntactic structure filters out irrelevant words and leaves important ones for word prediction. Chelba (2000) used the sentence "the contract ended with a loss of 7 cents after trading as low as 9 cents" to illustrate the idea of SLM. The partial parse of the word history before predicting “after” is shown in Figure 2.2. The word “ended” rooted in the sub tree is called a headword that represents a constituent in the sentence. The phrase “with a loss of 7 cents” is a preposition phrase that complements the word “ended”. A trigram language model would predict word “after” using its two closest words “7” and “cents”, while the headwords “contract” and “ended” are a better predictor for “after”.

![Figure 2.2 An example of a partial parse (Chelba & Jelinek, 2000)](image)

SLM incrementally parses a sentence and then extracts the exposed headwords for word prediction. The parse of a sentence is represented by a binary tree that consists of parts-of-speech (POS) tags, binary branching parses, non-
terminal labels, and headwords. Headwords are words on the non-leaf nodes of the binary trees, and non-terminal labels represent the POS tags of headwords.

Let \( W \) be a sentence of \( n \) words where \( w_0 = <s> \) and \( w_{n+1} = </s> \), \( T_k \) represents the partial parse for \( W_k \) which is a series of sub-trees of a binary tree, and \( W_k T_k \) is called the word and parse \( k \)-prefix. Figure 2.3 (Chelba, 2000) shows a word and parse \( k \)-prefix. \( h(0) \ldots h(-m) \) are the roots of the current \( m \) sub-trees, which are called heads. Each head consists of a headword and a non-terminal label, or a word and its POS tag in the case of root-only trees.

\[
\begin{align*}
  h(-m) &= (<s>, SB) \\
  h(-2) &= (h(-2).word, h(-2).tag) \\
  h(-1) &= (h(-1).word, h(-1).tag) \\
  h(0) &= (h(0).word, h(0).tag)
\end{align*}
\]

\[
\begin{align*}
  \ldots &
  \\
  (wp, tp) &
  \\
  \ldots &
\end{align*}
\]

\[
\begin{align*}
  \langle<s>, SB \rangle & \\
  \langle<wr, tr> \rangle & \\
  \langle<wp-1, tp-1> \rangle & \\
  \langle<wp, tp> \rangle & \\
  \langle<wp+1, tp+1> \rangle & \\
  \langle<wk, tk> \rangle & \\
  \langle<wk+1, ... \rangle & \\
  \langle</s> \rangle
\end{align*}
\]

**Figure 2.3 A word-and-parse \( k \)-prefix (Chelba & Jelinek, 2000)**

A partial parse is built by three modules: WORD-PREDICTOR, TAGGER, and CONSTRUCTOR. Figure 2.4 (Chelba, 2000) illustrates how the three modules work together. At first, the WORD-PREDICTOR predicts the next word \( w_{k+1} \) given the word-and-parse \( k \)-prefix \( W_k T_k \) and then passes control to the TAGGER. Next, TAGGER predicts the POS tag \( t_{k+1} \) for the word \( w_{k+1} \) given \( W_k T_k \) and the word \( w_{k+1} \) and then passes control to the CONSTRUCTOR. Finally, CONSTRUCTOR builds the new partial parse \( T_{k+1} \) from \( T_k \) by taking a series of actions. There are four different kinds of actions: (unary, NT-label), (adjoin-left, NT-label), (adjoin-right, NT-label), NULL. Once the CONSTRUCTOR hits NULL, it moves to the next step of predicting a new word.
Figure 2.4 Structured language model operations (Chelba & Jelinek, 2000)

The (unary, NT-label) operation generates a root-only sub-tree from the current word $w_{k+1}$ and its tag $t_{k+1}$ without adjoin operation. The operations (adjoin-left, NT-label) and (adjoin-right, NT-label) combine two sub-trees or a sub-tree and a word into a new sub-tree, and the left and right specify where the new headword is inherited. NT-label represents the non-terminal label assigned to the headword. The operations for (adjoin-left, NT-label) and (adjoin-right, NT-label) are illustrated in Figure 2.5-2.7 (Chelba, 2000). Figure 2.5 represents the parse before the adjoin action. Figure 2.6 and Figure 2.7 represent the parse after (adjoin-left, NT-label) and (adjoin-right, NT-label) being taken, respectively. In Figure 2.6, the headword of the new head comes from the left head, while in Figure 2.7 it comes from the right head.
SLM incrementally builds the parse while traversing a sentence from left to right. In this process, SLM assigns a probability to each word and parse $k$-prefix $W_kT_k$. Since a given word and parse $k$-prefix is generated by a unique sequence of operations, and SLM assigns a probability to each action, for a given $(W, T)$ pair, its
probability $P(W, T)$ can be calculated by multiplying the probabilities of all of its actions, which is calculated as

$$P(W, T) = \prod_{k=1}^{n}[P(w_k|W_{k-1}T_{k-1}) \times P(t_k|W_{k-1}T_{k-1}, w_k) \times P(T^k_{k-1}|W_{k-1}T_{k-1}, w_k, t_k)]$$

(2.6)

$$P(T^k_{k-1}|W_{k-1}T_{k-1}, w_k, t_k) = \prod_{i=1}^{N_k} P(p^k_i|W_{k-1}T_{k-1}, w_k, t_k, p^k_1 \ldots p^k_{i-1})$$

(2.7)

where $T^k_{k-1}$ is the incremental parse structure from $T_{k-1}$ to $T_k$ and the $p^k_i$ denotes the $i$-th action taken in the CONSTRUCTOR when constructing $T_k$ from $T_{k-1}$.

The probabilities $P(w_k|W_{k-1}T_{k-1})$, $P(t_k|W_{k-1}T_{k-1}, w_k)$, and $P(T^k_{k-1}|W_{k-1}T_{k-1}, w_k, t_k)$ represent the statistical models for WORD-PREDICTOR, TAGGER, and CONSTRUCTOR in SLM, respectively. In order to estimate the parameters of these models, SLM makes the following simplifications for the three models. The WORD-PREDICTOR uses the previous $m$ exposed heads as the equivalence classification for the history, and the TAGGER uses the non-terminal labels of the $m$ exposed heads and the current word as its equivalence classification, and the CONSTRUCTOR uses the tags of the previous $m$ exposed heads as its equivalence classification.

$$P(w_k|W_{k-1}T_{k-1}) = P(w_k|h_0 \ldots h_{-(m-1)})$$

(2.8)

$$P(t_k|W_{k-1}T_{k-1}, w_k) = P(t_k|h_0, tag \ldots h_{-(m-1)}, tag, w_k)$$

(2.9)

$$P(p^k_i|W_{k-1}T_{k-1}) = P(p^k_i|h_0, tag \ldots h_{-(m-1)}, tag)$$

(2.10)

For a given $W_k$, its possible parse $T_k$ is not unique. Thus, SLM predicts next words based on a set of parses weighted by their probabilities $P(T_k|W_k)$. Actually, for a normal length word sequences $W_k$, there are too many possible partial parses
In order to save the computational cost, SLM prunes parses with low probabilities and leaves the parses with high probabilities for word prediction. SLM keeps multiple stacks (multi-stacks), each of which contains partial parses that have been constructed by the same number of CONSTRUCTOR actions. There are two parameters controlling the number of parses in each stack: the stack depth \textit{maximum_stack_depth} representing the maximum number of partial parses allowed in each stack, and log-probability threshold \textit{LnP_threshold} representing the maximum difference allowed between the highest log probability of the partial parse and the lowest one in each stack. Given the multi-stacks of $W_k$, $P(w_{k+1}|W_k)$ is calculated as

$$P(w_{k+1}|W_k) = \sum_{T_k \in S_k} P(w_{k+1}|W_k T_k) \ast \rho(W_k, T_k)$$  \hspace{1cm} (2.11)

where $S_k$ is the set of all parses in the multi-stacks for $W_k$, and \(\rho(W_k, T_k)\) represents the normalized conditional probability $P(T_k|W_k)$, which is calculated as:

$$\rho(W_k, T_k) = P(W_k T_k) / \sum_{T_k \in S_k} P(W_k T_k)$$  \hspace{1cm} (2.12)

### 2.4. **Probabilistic Latent Semantic Analysis**

Probabilistic Latent Semantic Analysis (PLSA) (Hofmann, 2001) predicts words based on the semantics of the document, and it is closely related to the technique of Latent Semantic Analysis (LSA). LSA is the application of Singular Value Decomposition (SVD) to a term-by-document matrix. Given a $t \times d$ term-by-document matrix $X$ of rank $r$, SVD can decompose the matrix $X$ into the products of three matrices.

$$X_{t \times d} = U_{t \times r} \ast S_{r \times r} \ast V_{d \times r}^{'}$$  \hspace{1cm} (2.13)
where \( S \) is a diagonal matrix in which values in the diagonal are the square root of the eigenvalues of the matrix \( X'X \), and the columns of \( U \) and \( V \) are the eigenvectors of the matrix \( XX' \) and \( X'X \), respectively. One takes the first \( k \) columns of the matrix \( U \) to form a \( t \times k \) matrix \( T' \), and \( T'X \) projects the original term-by-document vectors into a \( k \) dimensional space.

LSA assumes that semantically related terms occur in the same documents more often than chance. With SVD, co-occurring terms are projected into the same dimensions, while non-co-occurring terms are projected into different dimensions. Thus, documents can have high similarity in the latent semantic space even if they do not share any terms as long as their terms are semantically similar (Manning & Schütze, 1999).

PLSA is the statistical view of LSA, and it defines a generative model for word-document co-occurrences generation using the bag-of-words assumption. Each observation \((w_j, d_i)\) represents an occurrence of the word \( w_j \) in a document \( d_i \). PLSA assumes that the distribution of the word \( w_j \) in a document \( d_i \) is associated with a latent semantic class variable \( z \). Specifically, PLSA assumes that a latent semantic class variable \( z \) determines the distribution of the next word and that the distribution of the semantic class \( z \) is determined by the current document. The generative model of PLSA is presented in Figure 2.8 (Hofmann, 2001) and can be described as follows:

1. Select a document \( d_i \) with probability \( P(d_i) \).
2. Pick a latent semantic class \( z_k \) with probability \( P(z_k | d_i) \).
3. Generate a word \( w_j \) with probability \( P(w_j | z_k) \).
The latent class size $K$ is much smaller than the word vocabulary size $M$ and the number of documents $N$. Thus, the hidden variable size $K$ acts as a bottleneck in word prediction (Hofmann, 2001). For a given observation pair $(d_i, w_j)$, its probability can be calculated as:

$$P(d_i, w_j) = P(d_i)P(w_j | d_i)$$  \hspace{1cm} (2.14) 

$$P(w_j | d_i) = \sum_{k=1}^{K} P(w_j, z_k | d_i) = \sum_{k=1}^{K} P(w_j | z_k, d_i) * P(z_k | d_i)$$  \hspace{1cm} (2.15)

Since the document variable $d_i$ and the word variable $w_j$ are conditionally independent on the latent variable $z_k$, one can get

$$P(w_j | d_i) = \sum_{k=1}^{K} P(w_j | z_k) * P(z_k | d_i)$$  \hspace{1cm} (2.16)

Given a training corpus, a PLSA model is trained with the expectation maximization (EM) algorithm. EM iteratively goes through the E-step and M-step to maximize the log likelihood of the observed data, which can be calculated as:

$$L = \sum_{i=1}^{N} \sum_{j=1}^{M} n(d_i, w_j) \log P(d_i, w_j)$$

$$= \sum_{i=1}^{N} n(d_i) \left[ \log P(d_i) + \sum_{j=1}^{M} \frac{n(d_i, w_j)}{n(d_i)} \log \sum_{k=1}^{K} P(w_j | z_k) * P(z_k | d_i) \right]$$  \hspace{1cm} (2.17)

where $n(d_i, w_j)$ refers to the number of occurrences of $w_j$ in document $d_i$, and $n(d_i) = \sum_{j=1}^{M} n(d_i, w_j)$ denotes the length of the document $d_i$.

In the E-step, one estimates the posterior probabilities of hidden variables $z_k$ given the observation:

$$P(z_k | d_i, w_j) = \frac{P(w_j | z_k) * P(z_k | d_i)}{\sum_{h=1}^{K} P(w_j | z_h) * P(z_h | d_i)}$$ \hspace{1cm} (2.18)
In the M-step, one updates the conditional probability of $w$ given the hidden variable $z$:

\[
P(w_j | z_k) = \frac{\sum_{i=1}^{N} n(d_i, w_j) P(z_k | d_i, w_j)}{\sum_{m=1}^{M} n(d_i, w_m) P(z_k | d_i, w_m)} \tag{2.19}
\]

\[
P(z_k | d_i) = \frac{\sum_{j=1}^{M} n(d_i, w_j) P(z_k | d_i, w_j)}{n(d_i)} \tag{2.20}
\]

2.5. A Syntactic, Semantic, and Lexical Language Model

The above n-gram, SLM, and PLSA language models capture only one or two aspects of the natural language. n-gram reflects the lexical information in natural language, and SLM exploits the syntactic structure of sentences for word prediction, while PLSA predicts word document occurrences based semantics of the current document.

Tan et al. (2012) proposed a language model that integrates the three language models above: n-gram, SLM, and PLSA. In this study, we refer this language model as the composite language model (CLM). CLM is a generative model that has four components: WORD-PREDICTOR, TAGGER, CONSTRUCTOR, and SEMANTIZER. The TAGGER and CONSTRUCTOR in SLM and the SEMANTIZER in PLSA remain unchanged. However, the WORD-PREDICTOR in CLM is an integration of word predictors in n-gram, SLM, and PLSA. The word predictors in n-gram, SLM, and PLSA use the closest $n-1$ word $w_{k-(n-2)}^k$, the previous $m$ exposed heads $h_{(m-1)}^0$, and the current semantic class variable $z$ respectively to predict the next word $w_{k+1}$. The composite language model combines all these parameters in n-gram, SLM, and PLSA,
which results in a more complex and powerful word predictor

\[ P(w_{k+1}|w_k^{(n-2)}, h_{m-1}^0, z_k). \]

\[ P(w_{k+1}|w_k) = \sum_{T_k \in S_k} \rho(Z_k | W_k) * P(T_k | z_k, W_k) * P(z_k) \]

(2.21)

When CLM is used in the decoding search of speech recognition, the document keeps changing as the decoding process moves forward from one sentence to next. CLM has to keep updating the distributions of semantic classes according to the content of the current document. The topic distribution of the

\[ \text{Figure 2.9 The composite language model (Tan et al., 2012)} \]

CLM has to parse the sentence while traversing it from left to right as SLM does. Similar to SLM, CLM also prunes the parses with low probabilities and leaves the high probability ones for word prediction. For the word history \( W_k \), given the multi-stacks \( S_k \) that stores its parses and the distribution of semantic class variable \( z_k \) of the current document, the probability of next word \( w_{k+1} \) is calculated as:

\[ P(w_{k+1}|w_k) = \sum_{T_k \in S_k} \rho(Z_k | W_k) * P(T_k | z_k, W_k) * P(z_k) \]
current document is re-estimated by maximizing the probability of word sequences
in the document so far while holding the other parameters fixed. Tan et al (2012)
used three different ways, one-step online EM, online EM with fixed learning rate,
and batch EM, to re-estimate the distribution of semantic class variable \( z_k \). The
online EM algorithm is stated as follows:

\[
P(z_k|d_k) = \gamma \frac{\sum_{k \in Z} \sum_{h_{(m-1)} \in T_{k-1}; T_{k-1} \in S_{k-1}} P(w_k|W_{k-n+1}^{k-1}h_{(m-1)}^{(m-1)}z_k)P(z_k|d_{k-1})P(T_{k-1}|W_{k-1})}{\sum_{k \in Z} \sum_{h_{(m-1)} \in T_{k-1}; T_{k-1} \in S_{k-1}} P(w_k|W_{k-n+1}^{k-1}h_{(m-1)}^{(m-1)}z_k)P(z_k|d_{k-1})P(T_{k-1}|W_{k-1})} + (1 - \gamma) \cdot P(z_k|d_{k-1})
\]

(2.22)

where \( S_{k-1} \) represents the set of parses in the multi-stacks of \( W_{k-1} \), \( \gamma \) is a value
between 0 and 1, \( h_{(m-1)}^{(m-1)} \) represents the recent \( m \) exposed headwords of \( T_{k-1} \), and
\( d_{k-1} \) represents the document that consists of previous sentences and the word
history \( W_{k-1} \) in the current sentence.

2.6. Evaluating Language Models

Given two language models \( LM1 \) and \( LM2 \), one wants to evaluate the
qualities of the two language models and choose the better one. The most reliable
way to evaluate language models is to apply each of them in an application system
and use the performance of the system to measure their qualities. This is called
extrinsic evaluation (Jurafsky & Martin, 2008). Language models interplay with
other components in systems. Therefore, it is difficult to draw a conclusion even if
one language model has a better performance than the other one in a system
because it is possible that the other model outperforms this one in some other
systems or cases.
Moreover, it is difficult and time-consuming to carry out an extrinsic evaluation of language models. Intrinsic evaluation uses a metric independent of any application to measure the quality of language models. Given a test word sequence \( W \) and two language models \( LM_1 \) and \( LM_2 \), the better language model has a tighter fit to the test data and thereby assigns higher probability to the test data \( W \) (Jurafsky & Martin, 2008). In other words, \( LM_1 \) is considered as a better language model than \( LM_2 \) if \( p_{LM_1}(W) > p_{LM_2}(W) \).

Perplexity is such an intrinsic evaluation metric for measuring the quality of language models. It is defined as:

\[
PP(W) = P(w_1w_2 \ldots w_N)^{-1/N} = \sqrt[N]{1 \over P(w_1w_2 \ldots w_N)}
\]

From the definition, one can find that the higher the probability of the word sequence \( P(w_1w_2 \ldots w_N) \) is, the lower the perplexity is. Although an intrinsic improvement in perplexity does not guarantee an extrinsic improvement in speech recognition, it often correlates with such an improvement.

CLM captures three most important aspects of natural language: lexical order in corpora, syntactic structures of sentences, and semantics of documents. Tan et al.’s (2012) experiments also show that CLM gives drastic perplexity reduction over \( n \)-gram, PLSA, and SLM, which makes CLM a promising language model for speech recognition. Thus, we aim to apply CLM in the decoding stage of speech recognition and investigate its effects on the performance of speech recognition in our study.
Chapter 3 Lattice Rescoring in Speech Recognition

3.1. Two-pass Decoding

Although the Viterbi decoder is able to find a word sequence hypothesis for a given acoustic observation \( O \), it has some major constraints. One of the major constraints is that it is infeasible to apply sophisticated language models or acoustic models in the Viterbi decoder for speech recognition because of the computational cost arising from a large search space. When complex language models or acoustic models are applied, the Viterbi decoder has a high order of time complexity and space complexity. However, online speech recognition systems like the telemedicine automatic captioning system usually require the overall processing time, which includes front-end processing, feature analysis, and search and decoding, to be less than \( 1.0 \times \) real time (Xue, 2007). The most commonly used \( n \)-gram language models in the Viterbi decoder are bigram and trigram. For a large vocabulary task, when \( n \) equals to 4 or larger value, the decoding cannot be done in real time.

Simple language models and acoustic models are not powerful enough to deliver accurate results. A solution to fix this problem is to modify the Viterbi decoder to return multiple potential hypotheses, instead of just the single best, and then use other sophisticated language models or acoustic models to re-rank these hypotheses and output the new best one. This is called two-pass decoding in speech recognition.

The two-pass decoding consists of two stages. The first stage uses an unsophisticated but efficient models like a bigram language model or simplified acoustic models to generate a set of hypotheses. The second stage rescores the
hypotheses in a reduced search space with a more sophisticated language model or acoustic model and then re-ranks them to find the best one. The interface between the two passes varies. It could be a N-best list or a word lattice.

A N-best list is a list of N hypotheses, each of which contains a word sequence, and the acoustic score and the language model score of the word sequence. The first pass returns the top N hypotheses according to their acoustic score and language model score. In the second pass, a more sophisticated language model or acoustic model is applied to each of the N-best hypotheses, and the original language model score or acoustic score of each hypothesis is replaced by the new language model score or acoustic model score. The N-best hypotheses will be rescored and re-ranked, and the new best one is output.

Usually, N ranges from 100-1000. When N is large, it is inefficient to generate N-best list from the Viterbi decoder. In addition, the second pass decoder has to go through each of the hypothesis to find the new best, which is also time-consuming. Because of these disadvantages, the first pass decoder usually returns a more sophisticated representation called a word lattice (Murveit et al., 1993; Aubert and Ney, 1995). A word lattice is a more compact representation of the alternative hypotheses. A normal sized word lattice stores orders of magnitude more hypotheses than a typical N-best list.

A word lattice is a directed graph that has exactly one starting node and one ending node. Usually, each node represents a time frame, and the starting node and the ending node of the lattice represent the first and the last frame of the utterance, respectively. Each link represents a hypothetical word and contains the following
information: a word symbol, an acoustic model score, and a language model score for that word symbol. A path in the lattice is an ordered sequence of links connected by nodes. A complete path that presents a hypothesis is the path whose starting and ending nodes are starting and ending nodes of the lattice respectively. The score of the hypothesis is calculated by accumulating the acoustic score and language model score in all links of the path. The words of the links in the complete path form the sentence hypothesis.

**Figure 3.1 An example of a word lattice**

### 3.2. A* search

The second pass decoder on a word lattice aims to find the complete path with the maximum score using a new model, such as a new language model. In essence, it is a graph search problem. Many algorithms can be used to address the graph search problem, such as breath-first search (BFS) and depth-first search (DFS). BFS and DFS expand the nodes in a graph in a pre-defined order. For graphs with thousands of nodes, which is common in a word lattice, BFS and DFS algorithm are inefficient. A* search (Russell, Norvig, Davis, Russell, & Russell, 2010) is another widely used search algorithm for graphs. If an appropriate heuristic function is chosen, A* search is extremely efficient and is able to find the optimal path in the graph by expanding a few nodes.
A* search uses a priority queue to store the set of paths that it is exploring. At the beginning, an empty path that just contains the root node is inserted into the priority queue. In each step, the first path in the priority queue is popped out and examined to see if its end node matches the goal. If it does not, the path is expanded, and its successors are put in priority queue. The paths in the priority queue are sorted by a rank function $f(x)$. $f(x)$ is calculated by adding two components $g(x)$ and $h(x)$, where $g(x)$ represents the score of the path $x$, and $h(x)$ is a heuristic function that estimates the score of the optimal path that starts from the end node of $x$ to the goal node. Thus, the function $f(x)$ estimates a total score of an optimal complete path containing the prefix $x$.

If $h(x)$ satisfies certain conditions, A* search is regarded as both complete and optimal. Completeness means that A* is able to find a solution if it exists without trapping in some loops in the graph. Optimality means that the solution found by A* is optimal. If $h(x)$ never overestimates the score for the optimal path in the case of finding the path with minimum score, and never underestimates the score in the case of finding the path with maximum score, the function $h(x)$ is called an admissible heuristic function. In that case, A* search is both complete and optimal.

The function $h(x)$ is critical to A* search. If the value estimated by $h(x)$ is close to the actual score of the optimal path that starts from the end node of $x$ to the goal node, A* search will be very effective, which means it can find the optimal path by expanding fewer nodes in the graph. Otherwise, it may not be more efficient than BFS and DFS.
3.3. Lattice rescoring with A* search

The original language model scores in the links of lattices are calculated from a simple language model in the first pass decoder. The second pass decoder needs to recalculate the language model scores with a more sophisticated language model, rescore paths in the lattice, and find the new best path with maximum score. In this study, we adopt the rescoring algorithm that is used by Chelba(2000) for lattice rescoring using SLM. The algorithm is described as follows.

A path $p$ in the lattice is an ordered set of links $l_0 ... l_n$ in which any two consecutive links are connected by a node in the lattice.

$$ p = \{ l_0 ... l_n \mid \text{ending Node}(l_i) = \text{starting Node}(l_{i+1}), 0 \leq i < n \} \quad (3.1) $$

Each link $l_i$ contains a word symbol $w(l_i)$, the log acoustic model score $\log P_{AC}(l_i)$, the original log language model score $\log P_{NG}(l_i)$ calculated from an n-gram language model in the first pass decoder. In the second pass decoder, the original log language model score $\log P_{NG}(l_i)$ will be replaced by a new language score $\log P_{LM}(l_i)$ computed from a more sophisticated language model. The lattice rescoring aims to find a complete path $p = l_0 ... l_n$ that maximize the score function

$$ g = \sum_{i=0}^{n}[\log P_{AC}(l_i) + \text{LM\_WEIGHT} \cdot \log P_{LM}(w(l_i)|w(l_0)...w(l_{i-1})) - WP] \quad (3.2) $$

where $WP$ is the word insertion penalty score, and $P_{LM}(w(l_i)|w(l_0)...w(l_{i-1}))$ is the language model score computed from a sophisticated language model in the second pass decoder, and $\text{LM\_WEIGHT}$ represents the weight of language model score.
The hypotheses in a lattice are organized as a prefix tree. A hypothesis $h$ that is represented by a complete path can be considered as a concatenation of a prefix $x$, a path that starts from the start node of the lattice to an intermediate node, and a suffix $y$, a path that starts from intermediate node to the end node of the lattice. As the $A^*$ search is exploring the lattice, a set of prefixes are examined until a complete path is found. As we mentioned earlier, prefixes in the priority queue are sorted according to a rank function: $f(x) = g(x) + h(x)$, where $g(x)$ represents the score of the prefix $x$, which can be calculated by equation 3.2, and $h(x)$ estimates the score for the optimal suffix that starts from the end node of the prefix $x$. In order to be optimal, $h(x)$ should not underestimate the score in this case. For a given prefix $x$, let $S(x)$ be the set of suffix whose prefix is $x$. $h(x)$ is set as:

$$h(x) = \max_{y \in S(x)} h(y|x)$$  \hspace{1cm} (3.3)

where $h(y|x)$ estimates the score of the suffix $y$. If $h(y|x)$ does not underestimate the score for each suffix $y$, $h(x)$ will be an admissible function. When $x$ is a complete path, it means the suffix $y$ is empty, $h(x) = 0$. In that case, one has $f(x) = g(x)$. For a given suffix $y = l_k \ldots l_n, h(y|x)$ is calculated as:

$$h(y|x) = \sum_{i=0}^{n} [\log P_{AC}(l_i) + LM_{WEIGHT} \ast (\log P_{NG}(l_i) + \log P_{COMP}) - WP]$$  \hspace{1cm} (3.4)

where $\log P_{COMP}$ compensates the difference between the original log language model score $\log P_{NG}(l_i)$ and the new log language model score $\log P_{LM}(l_i)$. Theoretically, $\log P_{COMP}$ should satisfy the condition

$$\log P_{COMP} \geq \max_{l_i} (\log P_{LM}(l_i) - \log P_{NG}(l_i))$$  \hspace{1cm} (3.5)

However, in practice, $\log P_{COMP}$ is set it by trial and error. For each node, one needs to calculate $h(y|x)$ for all suffixes starting from this node and picks up the
maximum score. This can be easily done with a backward dynamic programming algorithm.

The A* search uses a priority queue to store the prefixes that are being examined. Prefixes in the priority queue are ordered according to the rank function $f(x)$. In each step, the top prefix is popped out and expanded. Usually, a prefix has multiple branches. Thus, several new prefixes are added back into the priority queue. A* search repeats this procedures until the prefix popped out from the priority queue is a complete path. In practice, in order to speed up the processing time, the size of the priority queue is controlled by two parameters to restrict the number of paths that will be examined. The priority queue depth $queue\_depth\_threshold$ represents the maximum number of prefixes allowed in the priority queue, and the log threshold $queue\_logP\_threshold$ represents the maximum allowed score difference between the top prefix and the bottom one.
Chapter 4  Confusion Network

4.1. Introduction

In the standard MAP approach to speech recognition, the decoder searches for the sentence $W$ that maximizes the posterior probability of $W$ given an acoustic observation $O$ (Bahl, Jelinek & Mercer, 1983). From the perspective of Bayesian decision theory, maximizing sentence posterior probability corresponds to minimizing SER. However, minimizing SER does not necessarily lead to minimizing WER, the commonly used standard performance metric for speech recognition (Mangu, Brill, & Stolcke, 2000).

Minimizing WER corresponds to maximizing the word posterior probability in each position because it calculates error rate for each hypothetical word. Given an acoustic observation $O$, if one can find the words whose posterior probabilities are maximum at each position of the reference sentence, the concatenation of these words would lead to a sentence whose expected WER is minimum. The expected error for deletion and insertion can be resolved by introducing an “empty” word.

Although SER and WER tend to highly correlate with each other, they are not exactly the same. Empirical results show that there is a clear difference between decoding results from minimizing SER and WER in some cases (Andreas, Konig, & Weintraub, 1997). This happens when the concatenation of words with maximum word posterior probabilities does not generate the sentence with the maximum sentence posterior probability.

Considering that minimizing WER is the commonly agreed objective metric of speech recognition, the intuition to improve the traditional MAP approach is to
directly search for the hypothesis that minimizes the expected WER instead of SER. Although WER minimization has been implemented on N-best lists (Stolcke, Konig, & Weintraub, 1997), word lattices are a better option for WER minimization because of the following reason: 1) word lattices contains orders of magnitude more hypotheses than typical N-best lists, which makes the estimation of word posterior probabilities more accurate, 2) it also has a larger search space for finding the hypothesis that minimizes the expected WER (Mangu, Brill, & Stolcke, 2000).

Confusion network is an approach proposed by Mangu et al. (2000) to directly minimize the expected WER based on word lattices. A confusion network is a special linear graph that is transformed from a word lattice, and an example of confusion network is presented in Figure 4.1. In a confusion network, nodes are arranged in a linear order, and links only exist between two adjacent nodes. Each pair of two adjacent nodes forms a bin that contains many links. Each link has a word symbol and the posterior probability of the word symbol. The posterior probabilities of all links in a bin sum to 1. Any complete path that represents a hypothesis in the graph crosses all nodes.

![Confusion network example](image)

**Figure 4.1 An example of a confusion network**

For a given hypothesis $L = l_1 \ldots l_n$ in a confusion network, its expected WER can be calculated as:

$$E(WER) = \sum_{i=1}^{n}(1 - p_i)$$

(4.1)
where $l_i$ is a link in the $i$-th bin of the confusion network, and $p_i$ represents the posterior probability of the link $l_i$. For a given confusion network, one can choose the links with the highest posterior probability in each bin to form the sentence hypothesis that minimizes the expected WER.

### 4.2. Lattice to confusion network transformation

For a given confusion network transformed from a lattice, each node corresponds to a set of nodes in the lattice, and each link corresponds to a set of links that have the same word in the lattice. The posterior probability of a link in the confusion network is the sum of the posterior probabilities of the set of corresponding links in the lattice. The first step of lattice to confusion network transformation is to compute the posterior probabilities of the links in a lattice. The posterior probability of a link $l$ given a lattice is the sum of the probabilities of all paths passing through the link $l$ normalized by the sum of probabilities of all paths in the lattice, which can be computed by the forward-backward algorithm (Wessel, Schluter, Macherey, & Ney, 2001). The algorithm is described in detail as follows.

For a given complete path $p$ crossing the node $n_i$ in a lattice, it can be considered as a concatenation of a prefix $x$, the partial path from starting node of the lattice to node $n_i$, and a suffix $y$, the partial path from node $n_i$ to the ending node of the lattice. Given a node $n_i$ in a lattice, let $I(n_i)$ be the set of incoming prefixes of node $n_i$ and $O(n_i)$ be the set of outgoing suffixes of node $n_i$. Given a link $l$, $starting\_node(l)$ and $ending\_node(l)$ represent the starting and ending nodes of $l$ respectively, and $P(l)$ represents the probability of the link $l$. 
The forward algorithm calculates $\alpha (n_i)$ for all nodes in the lattice, which is the sum of the probabilities of all prefixes ending at node $n_i$ specifically. $\alpha (n_i)$ is calculated as:

$$\alpha (n_i) = \sum_{l \in l(n_i)} \alpha (\text{starting\_node}(l)) * P(l) \quad (4.2)$$

The forward algorithm starts from the starting node of the lattice whose $\alpha$ value is 1.

The backward algorithm calculates $\beta (n_i)$ for each node $n_i$ in the lattice, which is the sum of the probabilities of all suffixes starting from node $n_i$ specifically. $\beta (n_i)$ is calculated as:

$$\beta (n_i) = \sum_{l \in o(n_i)} \beta (\text{ending\_node}(l)) * P(l) \quad (4.3)$$

The backward algorithm starts from the ending node of the lattice whose $\beta$ value is 1.

For a link $l$, the sum of probabilities of all paths that pass through $l$ can be calculated as:

$$P = \alpha (\text{starting\_node}(l)) * P(l) * \beta (\text{ending\_node}(l)) \quad (4.4)$$

The sum of probabilities of all paths in the lattice can be obtained from the $\alpha$ value of the ending node of the lattice or the $\beta$ value of the starting node of the lattice. Thus, the posterior probability of the link $l$ given the lattice $G$ can be calculated as:

$$P(l|G) = \frac{\alpha (\text{starting\_node}(l)) * P(l) * \beta (\text{ending\_node}(l))}{\alpha (\text{ending\_node}(G))} \quad (4.5)$$

or

$$P(l|G) = \frac{\alpha (\text{starting\_node}(l)) * P(l) * \beta (\text{ending\_node}(l))}{\beta (\text{starting\_node}(G))} \quad (4.6)$$
Because the language model scores $P(W)$ and acoustic model scores $P(O|W)$ in a lattice are in different scales, in practice, a scaling factor $\beta$ whose value is usually less than 1 is used to flatten the acoustic model scores when calculating the posterior probabilities of links. Thus, given an observation $O$, the posterior probability of the word sequence $W$ is calculated as:

$$P(W|O) = \frac{P(W)P(O|W)^\beta}{P(O)} \quad (4.7)$$

This scaling factor $\beta$ is very important to the computation of the posterior probabilities of links in a lattice. Because of the large dynamic range of the acoustic scores, a few hypotheses will dominate the posterior probabilities if the scaling factor $\beta$ is not set appropriately (Wessel, Schluter, Macherey, & Ney, 2001). A posterior probability computed from a lattice is only an approximation to the actual word posterior probability because the word lattice is only a limited representation of the infinite space of solutions (Wessel, Schluter, Macherey, & Ney, 2001).

After the posterior probabilities of the links in a lattice are computed, the next step is to find an appropriate lattice to confusion network transformation algorithm that maps the nodes and links in a word lattice to the nodes and links in a confusion network. The original algorithm proposed by (Mangu, Brill, & Stolcke, 2000) for transforming a word lattice into a confusion has a time complexity of $O(T^3)$ where $T$ is the number of links in a lattice. The high order of time complexity of this algorithm limits its application in real time speech recognition. Xue and Zhao (2005) proposed a linear time complexity algorithm for transforming word lattices into confusion networks, which is adopted by our study. The algorithm is described as follows.
Given a lattice $G, N = \{n_0, n_1 ... n_k\}$ is the set of nodes in the lattice that are sorted in the ascending order of time frames, and $E = \{l_0, l_1 ... l_m\}$ is the set of links in the lattice. Let $t(n_i)$ represent the time frame represented by the node $n_i$. Thus, $t(n_i)$ is greater than or equal to $t(n_j)$ if $i$ is greater than or equal to $j$.

Input: a lattice $G$ with $(k + 1)$ nodes $\{n_0, n_1, ..., n_k\}$ that has been sorted in the ascending order of time frames

Output: a confusion network $CN = \{N_0, N_1, ..., N_f\}$ in which $B_{s \rightarrow s+1}$ represents the bin between the confusion network nodes $N_s$ and $N_{s+1}$

Initialization: create a confusion network node $N_0$, put it in $CN$, and assign $n_0$ to $N_0$.

For $i = 1$ to $k$

Suppose $n_{i-1}$ is assigned to $N_x$

If there are no links between node $n_i$ and any other nodes that has been assigned to $N_x$

Assign $n_i$ to $N_x$

Else

Create a new node $N_{x+1}$, put it in $CN$, and assign $n_i$ to $N_{x+1}$

End

For each incoming link $l$ of node $n_i$

Suppose the starting node and the ending node of $l$ has been assigned to $N_s$ and $N_t$ respectively

If $t = s + 1$ then

the link $l$ is assigned to the bin $B_{s \rightarrow t}$
Else

the link is assigned to a bin $B_{r\rightarrow r+1}$ where $s \leq r \leq t - 1$

according to a similarity function between the lattice link $l$ and a

bin $B_{c\rightarrow c+1}$, and $r = \arg\max_{S \leq r \leq t-1} SIM(l, B_{c\rightarrow c+1})$

End

End

End For

For each bin $B_{r\rightarrow r+1}$ in the confusion network

If the sum of the posterior probabilities of all links in the bin is less than 1

A special link with word symbol $<EPS>$ is added to the bin to make the

sum of the posterior probabilities of all links in the bin to 1, where

$<EPS>$ represents an “empty” word.

End if

End for

When a lattice link is assigned to a bin in the confusion network, the
algorithm needs to check whether there is a confusion network link in the bin that
has the same word with the lattice link. If there is none, the algorithm will create a
confusion network link and put in the bin, and assign the word symbol and posterior
probability of the lattice link to the confusion network link. If there is a confusion
network link in the bin that has the same word with the lattice link, the algorithm
will add the posterior probability of the lattice link to the posterior probability of the confusion network link.

The similarity between a lattice link and a confusion network bin is calculated as:

$$\text{SIM}(l, B_{c \rightarrow c+1}) = \frac{1}{|l|} \cdot \sum_{l' \in B_{c \rightarrow c+1}} \text{sim}(w(l), w(l')) \cdot \text{overlap}(l, B_{c \rightarrow c+1}) \quad (4.8)$$

where $l'$ represents any link that has been assigned to the bin $B_{c \rightarrow c+1}$. $w(l)$ and $w(l')$ are words on the lattice link $l'$ and $l$, respectively. $\text{sim}(w(l), w(l'))$ computes the phonetic similarity between two words, which is calculated as follows in our study:

$$\text{sim}(w(l), w(l')) = 1 - \frac{\text{minimum edit distance}(\text{pron}(w(l)), \text{pron}(w(l')))}{\text{length}(\text{pron}(w(l'))) + \text{length}(\text{pron}(w(l')))} \quad (4.9)$$

where $\text{pron}(w(l))$ is the phone sequence of $w(l)$'s pronunciation. $\text{overlap}(l, B_{c \rightarrow c+1})$ represents the time overlap between the lattice link $l$ and the bin. The length of $B_{c \rightarrow c+1}$ is $T_{\text{min}}(N_{c+1}) - T_{\text{max}}(N_c)$ where $T_{\text{max}}(N_i) = \max_{n_j \in N_i} (t(n_j))$ and $T_{\text{min}}(N_i) = \min_{n_j \in N_i} (t(n_j))$. Thus, $\text{overlap}(l, B_{c \rightarrow c+1})$ is calculated as:

$$\text{overlap}(l, B_{c \rightarrow c+1}) = \min(T_{\text{min}}(N_{c+1}), t(\text{ending node}(l))) - \max(T_{\text{max}}(N_c), t(\text{starting node}(l))) \quad (4.10)$$

### 4.3. Confusion network rescoring

After a confusion network is transformed from a lattice, one can take the link with the highest posterior probability in each bin of the confusion network to form the sentence hypothesis with the minimum expected WER. However, this method loses the chance of taking advantage of sophisticated language models because lattice is usually generated from a simple language model.
Confusion network rescoring uses a new language model to rescore the links in each bin of the confusion network with a rescoring function. After the links in each bin are re-ranked, the hypothesis that is formed by taking the top links in each bin optimizes the rescoring function. The purpose is to take advantage of sophisticated language models to further improve the decoding results. In this study, we follow the basic idea of a rescoring framework for confusion networks by Deoras and Jelinek (2009). The algorithm of the method is stated as follows:

Input: a confusion network CN that consists of $n$ confusion bins $\{B_1, B_2, \ldots, B_n\}$ where links in each bin are stored in descending order according to their posterior probabilities. $|B_i|$ is the size of the $i$-th bin, and $w_i(j)$ is the word on the $j$-th link in the $i$-th bin $B_i$.

Output: a word sequence $W = \{w_1(1), \ldots, w_n(1)\}$ where the word $w_i(1)$ is the word that comes from the top link in the bin $B_i$.

Initialization: $W_p = \text{null}, W_c = \{w_1(1), \ldots, w_n(1)\}, \text{iter} = 0$

While $W_p \neq W_c$ and $\text{iter} < \text{max\_iter}$

For $i = 1$ to $n$

For each link $w_i(j)$ in $B_i$

Rescore $w_i(j)$ by a scoring function given the context $\{w_1(1), \ldots, w_{i-1}(1), w_{i+1}(1), \ldots, w_n(1)\}$

End

Sort the links in $B_i$ according to their scores

End

$W_p = W_c$
$$W_c = \{w_1(1), ..., w_1(1), ..., w_n(1)\}$$

End

Basically, this algorithm is an iterative decoding of hill climbing. In each iteration of the loop, links with higher scores are moved to the top of bins. The word sequence $$W_c = \{w_1(1), ..., w_n(1)\}$$ resulting from each iteration is not worse than the previous one according to the scoring function. With a few iterations, the links on the top do not change, and the loop stops. A critical part of the framework is to find an appropriate scoring function to rescore the links in each bin. In this study, we test the following two rescoring functions. The first scoring function is defined as follows:

$$f(w_i(j)) = \log p_{i,j} + \lambda \cdot \log P(w_i(1) | w_i(1), ..., w_{i-1}(1)) \quad \text{for } w_i(j) \neq <\text{EPS}> \quad (4.11a)$$
$$f(w_i(j)) = \log p_{i,j} + \lambda \cdot \log P(<\text{EPS}>) \quad \text{for } w_i(j) = <\text{EPS}> \quad (4.11b)$$

where $$\log p_{i,j}$$ is the log posterior probability of the $$j$$-th link in $$B_i$$, and $$\log P(w_i(j) | w_i(1) ..., w_{i-1}(1))$$ is the log language model score of the word $$w_i(j)$$ given the words in the top links of the bins before $$B_i$$, which is calculated using a new language model. $$\lambda$$ is a scaling factor for the language model score, which is set to a fixed value. $$\log P(<\text{EPS}>)$$ represents a log probability of the empty word $$<\text{EPS}>$$ and is set to a fixed value.

The second scoring function is defined as follows:

$$f(w_i(j)) = \frac{\log P_{i,j}}{n} + \lambda \cdot \frac{\log P(w_i(1) ..., w_{i-1}(1), w_i(j), w_{i+1}(1), w_n(1))}{\text{length} \left( P(w_i(1) ..., w_{i-1}(1), w_i(j), w_{i+1}(1), w_n(1)) \right)} \quad (4.12)$$

where $$\log P_{i,j}$$ is the sum of the log posterior probability of the $$j$$-th link in $$B_i$$ and the log posterior probabilities of the top links in other bins.
\[ \log P(w_1(1) \ldots w_{i-1}(1), w_i(j), w_{i+1}(1), w_n(1)) \] is the log language model score of the word sequence \((w_1(1) \ldots w_{i-1}(1), w_i(j), w_{i+1}(1), w_n(1))\), which is calculated from a new language model. \( \text{length}(P(w_1(1) \ldots w_{i-1}(1), w_i(j), w_{i+1}(1), w_n(1))) \) is the length of the word sequence, which equals to the number of bins minus the number of \(<\text{EPS}>\) in the word sequence because \(<\text{EPS}>\) does not contribute to the length of the word sequence. \( \lambda \) is a scaling factor for the new language model score, which is set to a fixed value.

Before confusion network rescoring, confusion networks can be pruned in order to speed up the processing. For the links in each bin, if their posterior probabilities are too small in comparison with the top link, it is unlikely that they will be re-ranked to the top of the bins. We use a parameter \( \text{Log\_threshold} \) to control the number of links in each bin. If the difference between the maximum log posterior probability in the bin and the log posterior probability of the link is greater than \( \text{Log\_threshold} \), the link will be pruned from that bin.
Chapter 5 Implementation, Experiments, and Results

5.1. A distributed architecture

The word predictor in CLM uses previous $m$ exposed heads, previous $n - 1$ closest words, and the latent semantic class variable $z$ to predict next word. The parameters of a CLM trained from some normal sized corpora with a normal sized word vocabulary are too large to be stored in just one computer (Tan et al., 2012). In our study, the parameters of the composite language model are stored among hundreds of computer nodes in a supercomputer, and the parameters are distributed by hashing the $m$ exposed heads, previous $n - 1$ closest words, and the latent semantic class $z$. Considering that SLM is also a computational expensive language model, and its parameters are also distributed in the supercomputer using a similar way.

Since the parameters of CLM and SLM are distributed among different computer nodes in our study, a distributed application is necessary for implementing the lattice rescoring and confusion network rescoring with CLM in our study. The distributed application of lattice rescoring and confusion network rescoring is implemented with C++ and the Message Passing Interface (MPI). MPI is a standard message-passing library for parallel computing. Basically, it defines a set of protocols and libraries for message passing between processes running on different CPU nodes (Snir, et al., 1998).

Our application is a typical client-server architecture system. The lattice rescoring and confusion network rescoring are running on the clients, and the parameters of CLM and SLM are loaded to the servers. Figure 5.1 represents the
architecture of our application for lattice rescoring and confusion network rescoring using CLM or SLM. During the process of lattice rescoring and confusion network rescoring, many operations, such as calculating the probability of the current word given its history using SLM or CLM, request the probability of a CLM or SLM word prediction event. Whenever a client needs the probability of a CLM or SLM prediction event, it uses the hash algorithm to find the appropriate server, and sends the request to it. The servers are listening to the requests from the clients. Once a server gets a request, it finds the probability and sends it back to the client.

![Figure 5.1 Architecture of the distributed application for lattice rescoring and confusion network rescoring](image)
5.2.  Experimental Data Description

5.2.1. The ARPA WSJ Language Corpus

The CLM and the SLM used in this study are trained from the ARPA WSJ Language Corpus. The ARPA language corpus consists of articles published in the Wall Street Journal from December 1986 to November 1989. The articles are processed to reflect the way they are likely to be said when read aloud, such as transforming numbers and dollar marks into the corresponding words. After the transformation, the WSJ corpus consists of about 41.5 million words (Rosenfeld, 1996). The vocabulary for language model training is set to the 19997 most common words in the corpus, which combines <unk> representing words out of the vocabulary, <s> representing the sentence beginning, and </s> representing the sentence end, into a vocabulary of 20k words. Our partners at Wright State University trained a distributed SLM and a distributed CLM from the ARPA WSJ Language Corpus using the 20k words vocabulary. The parameters of the SLM and CLM are stored in 90 servers and 160 servers, respectively, in Ohio Supercomputer Center.

5.2.2. The Wall Street Journal lattices

The Wall Street Journal (WSJ) lattices used in this study are generated by Chelba (2000) from DARPA’93 HUB1 test speech set using the standard DARPA’93 HUB1 language model (LAT3-gram), which is trained from the ARPA WSJ Language Corpus using the DARPA’93 HUB1 standard open vocabulary of 20k words (Chelba, 2000). The test set consists of 213 lattices. The lattices vary in size with small
lattices containing dozens of nodes and dozens of links and large lattices containing
tens of thousands nodes and hundreds of thousands links.

5.2.3. Oracle WER

Oracle WER is the WER of the hypotheses that have the lowest WER. Oracle WER represents the best results that a decoder could achieve. After a confusion network is transformed from a lattice, it induces many paths that do not exist in the lattice. Thus, the oracle WER of confusion networks is usually less than that of the corresponding lattices. In practice, it has been observed that the oracle WER of confusion networks is lower than the corresponding lattices’ oracle WER by 1% absolute (Deoras and Jelinek, 2009).

5.3. Experiments & Results

5.3.1. Viterbi decoding

The oracle WER of the WSJ lattices is presented in Table 5.1. The substitution, insertion, and deletion error are 2.46%, 0.32%, and 0.58% respectively. The SER is 33.3% where 71 lattices do not contain the hypothesis that completely matches the reference sentence.

<table>
<thead>
<tr>
<th>Substitution</th>
<th>Insertion</th>
<th>Deletion</th>
<th>WER</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.46%</td>
<td>0.32%</td>
<td>0.58%</td>
<td>3.36%</td>
<td>33.3%</td>
</tr>
</tbody>
</table>

The complete path in a lattice with the maximum score before rescoring is the decoding result of the Viterbi decoder in the first pass. Given a lattice, this path can be easily obtained with a backward or forward algorithm. The WER of the
decoding results of the Viterbi decoder are presented in Table 5.2 when the language model weight is set to 16, and word insertion penalty is set 0.

**Table 5.2 WER of decoding results of Viterbi decoder**

<table>
<thead>
<tr>
<th>Substitution</th>
<th>Deletion</th>
<th>Insertion</th>
<th>WER</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.7%</td>
<td>2.9%</td>
<td>1.0%</td>
<td>13.7%</td>
<td>69.0%</td>
</tr>
</tbody>
</table>

5.3.2. Lattice Grouping

The WSJ test set in our study consists of 213 lattices, which are generated from the speech of reading texts drawn from the Wall Street Journal publications by different speakers. Each group of several lattices is generated from some selected reading texts in the same document by one speaker.

The latent class variable $z$ plays its role when the topic distribution converges to several topics, which usually requests updating the initial topic distribution with a few sentences of a document. The original lattices do not contain information about how lattices should be grouped into documents in order to take advantage of the prediction power of the latent class variable. To overcome this shortcoming, after we get the decoding results of the Viterbi decoder, we use a text segmentation tool by Utiyama and Isahara (2001) to assign the Viterbi decoding results into different documents. According to the segmentation results, the 213 sentences from the Viterbi decoder are grouped into 65 documents in our study. In the experiments of lattice rescoring with CLM and confusion network rescoring with CLM, lattices are grouped in the same way according to the Viterbi decoding results. The perplexity of the CLM on the segmented decoding results of the Viterbi decoder is 102.97.
5.3.3. Lattice rescoring with CLM and SLM

Chelba (2000) has developed a program for lattice rescoring with SLM on the WSJ lattices. In our study, a distributed application for lattice rescoring with SLM or CLM is developed based on Chelba’s program. The 213 WSJ lattices are processed by 10 clients, and each client takes care of about 20 lattices. A set of experiments is conducted on lattice rescoring with CLM or SLM to evaluate the effects of CLM on lattice rescoring. Especially, we are interested in comparing the relative effectiveness of SLM and CLM on lattice rescoring with the WSJ lattices.

Instead of simply replacing the original trigram language model score with the CLM score or SLM score for each link in lattice rescoring, we linearly interpolate the original trigram language model score with the language model score form CLM or SLM as follows:

\[ P(l) = (1 - \lambda) \cdot P_{CLM/SLM}(l) + \lambda \cdot P_{LAT3-gram}(l) \]  \hspace{1cm} (5.1)

where \( \lambda \) is set to different values in a set of experiments. When \( \lambda \) is set to 0, the original trigram language model score is replaced by the CLM score or the SLM score. When \( \lambda \) is set to 1, there is no replacement, and A* lattice rescoring gets the decoding results of the Viterbi decoder.

The parameters in our study are set to the same values as Chelba’s experiments. The two parameters that control the multi-stacks that store the parses \( T_k \) for the word history \( W_k \) are set to: \textit{maximum_stack_depth} = 10 and \textit{LnP_threshold} = 6.91 (see section 2.3). \( \log P_{COMP} \) compensating for the difference in log language model scores is set to 0.5 (see equation 3.4). The language model weight \textit{LM_WEIGHT} is set to 16, and the word insertion penalty score is set to 0.
The two parameters that control the number of prefixes in the search stack are set to: \textit{queue\_logP\_threshold} = 100 and \textit{queue\_depth\_threshold} = 25 (see section 3.3).

The WER of the decoding results of lattice rescoring with CLM or SLM are presented in Table 5.3. The linear interpolation parameter \( \lambda \) is set to 0.0, 0.2, 0.4, 0.6, and 0.8, respectively. When \( \lambda \) is set to 0.0 for lattice rescoring with SLM, the WER is increased from 13.7 to 14.1, which means SLM alone cannot decrease the WER in our study. When \( \lambda \) is set to 0.0 for lattice rescoring with CLM, CLM alone is able to decrease the WER from 13.7 to 13.3. When \( \lambda \) is set to 0.2 in lattice rescoring with CLM, the WER of the decoding results is 13.2, which is the best results we get in this set of experiments.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>A*</th>
<th>Viterbi</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>WER (%) (SLM)</td>
<td>14.1</td>
<td>13.5</td>
</tr>
<tr>
<td>WER (%) (CLM)</td>
<td>13.3</td>
<td>13.2</td>
</tr>
</tbody>
</table>

Table 5.4 provides the details of WER of the Viterbi decoder, and lattice rescoring with CLM and SLM when \( \lambda \) is set 0. The results show that the primary improvement of lattice rescoring with CLM comes from the decrease on substitution error. Compared with the Viterbi decoder and lattice rescoring with SLM, lattice rescoring with CLM decrease the substitution error by 0.7\% and 0.9\%, respectively.
Table 5.4 Details of WER (%) for lattice rescoring

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Substitution</th>
<th>Deletion</th>
<th>Insertion</th>
<th>WER</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viterbi</td>
<td>9.7</td>
<td>2.9</td>
<td>1.0</td>
<td>13.7</td>
<td>69.0</td>
</tr>
<tr>
<td>SLM</td>
<td>9.9</td>
<td>2.9</td>
<td>1.3</td>
<td>14.1</td>
<td>70.9</td>
</tr>
<tr>
<td>CLM</td>
<td>9.0</td>
<td>3.2</td>
<td>1.0</td>
<td>13.3</td>
<td>66.2</td>
</tr>
</tbody>
</table>

Table 5.5 presents some decoding results when lattice rescoring with CLM gets better results. We find that CLM combines the advantages of SLM and the trigram language model in many cases. In some cases, CLM and SLM get the correct results, while trigram language model does not. In some other cases, CLM and trigram language model get the correct decoding results, while SLM does not.

Table 5.5 Decoding results when CLM improves WER

<table>
<thead>
<tr>
<th>Lattice Id</th>
<th>Decoding Results</th>
</tr>
</thead>
</table>
| 4oac020e   | CLM: he said suez came to bring better *management* to the company to increase productivity and profitability  
            SLM: he said suez came to bring better *management’s* to the company to increase productivity and profitability  
            Viterbi: he said suez came to bring better *management* to the company to increase productivity and profitability  
            Reference: he said suez aimed to bring better *management* to the company to increase productivity and profitability |
| 4oac020g   | CLM: in the newly created position he heads the new public finance *department*  
            SLM: in the newly created position he heads the new public finance *departments* |
Viterbi: in the newly created position he heads the new public finance department

Reference: in the newly created position he heads the new public finance department

4oac020k  CLM: purchasers also named one hundred eighty nine commodities that rose in price last month while only three dropped in price

SLM: purchasers also named one hundred eighty nine commodities that rose in price last month when only three dropped in price

Viterbi: purchasers also named one hundred eighty nine commodities that rose in price last month by only three dropped in price

Reference: purchasers also named one hundred and eighty nine commodities that rose in price last month while only three dropped in price

(4obc0201)  CLM: only three of the nine banks saw foreign exchange profits declined in the latest quarter

SLM: only three of the nine bank saw foreign exchange profits declined in the latest quarter

Viterbi: only three of the nine bank saw foreign exchange profits declined in the latest quarter

Reference: only three of the nine banks saw foreign exchange profits decline in the latest quarter (4obc0201)

4obc0203  CLM: a spokeswoman blames the decline on market volatility and says the swing is within a reasonable range for us

SLM: a spokeswoman blames the decline on market volatility and says the swing is within a reasonable range for us

Viterbi: a spokeswoman blames the decline on market volatility and says this swing is within a reasonable range for us

Reverence: a spokeswoman blames the decline on market volatility and says the swing is within a reasonable range for us

4ojc020h  CLM: yesterday many corn futures prices plunged by the permissible daily limits
In addition, we also find that the CLM improves the quality of the decoding results although it does not decrease the WER of the decoding results in some cases. Table 5.6 presents some of these decoding results.

**Table 5.6 Decoding results when CLM improves understanding**

<table>
<thead>
<tr>
<th>Lattice Id</th>
<th>Decoding results</th>
</tr>
</thead>
<tbody>
<tr>
<td>4obc0204</td>
<td>CLM: law enforcement officials <em>sensible</em> a measure their success by the price of drugs on the street (4obc0204)</td>
</tr>
<tr>
<td></td>
<td>SLM: law enforcement officials <em>sent simply</em> measure their success by the price of drugs on the street (4obc0204)</td>
</tr>
<tr>
<td></td>
<td>Viterbi: law enforcement officials <em>said simply</em> measure their success by the price of drugs on the street (4obc0204)</td>
</tr>
<tr>
<td></td>
<td>Reference: law enforcement officials <em>sensibly</em> measure their success by the price of drugs on the street (4obc0204)</td>
</tr>
<tr>
<td>4odc020k</td>
<td>CLM: Monday’s decline <em>shake</em> the confidence of every investor big and small</td>
</tr>
<tr>
<td></td>
<td>SLM: Monday’s decline <em>shipped</em> the confidence of every investor big and small</td>
</tr>
<tr>
<td></td>
<td>Viterbi: Monday’s decline <em>ship</em> the confidence of every investor big and small</td>
</tr>
<tr>
<td></td>
<td>reference: Monday’s decline <em>shook</em> the confidence of every investor big and small</td>
</tr>
</tbody>
</table>

### 5.3.4. Lattice to confusion network transformation

The algorithm of lattice to confusion network transformation (Xue & Zhao, 2005) adopted in our study has a linear order of time complexity. We implement it with the Java programming language. After a confusion network is transformed
from a lattice, and links in each bin of the confusion network are sorted in descending order of posterior probability, one can take the top link in each bin to find the decoding results of the confusion network. A set of experiments is conducted to evaluate the performance of lattice to confusion network transformation on the WSJ lattices. The only parameter needed to be set is the scaling factor for acoustic scores, which is set 1/10, 1/12, 1/14, 1/16, 1/18, and 1/20 respectively in our study. The WER of the confusion networks is presented in the following table when different scaling factor is set. From the results, we find that confusion networks can decrease the WER from 13.7% to 13.3% when the scaling factor for acoustic model scores is set to 12 or 16.

<table>
<thead>
<tr>
<th>Scaling factor</th>
<th>1/10</th>
<th>1/12</th>
<th>1/14</th>
<th>1/16</th>
<th>1/18</th>
<th>1/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>WER (%)</td>
<td>14.0</td>
<td>13.3</td>
<td>13.4</td>
<td>13.3</td>
<td>13.6</td>
<td>14.4</td>
</tr>
</tbody>
</table>

5.3.5. Confusion network rescoring

The goal of confusion network rescoring is to take advantage of a sophisticated language model to rescore links in each bin to get a better decoding result. In confusion network rescoring, we use the CLM to rescore the links in each bin and find a new hypothesis that optimizes a scoring function. An appropriate scoring function is critical to confusion network rescoring. In our study, the two scoring functions defined by equation 4.11 and equation 4.12 are tested. The confusion networks for rescoring are transformed from the WSJ lattices when the scaling factor for acoustic score is set to 12. The distributed application for lattice rescoring with CLM is also developed using C++ and MPI. The 213 confusion
networks are processed by 10 clients with each client taking care of about 20 confusion networks.

For the scoring function equation 4.11a and 4.11b, there are two parameters needed to be set: $\lambda$, the scaling factor for the language model score, and $\log P(<$ EPS $>)$, the log probability score of the empty word $<$ EPS $>$. For the scoring function equation 4.12, only $\lambda$ needs to be set. In our study, different values of those parameters are tested for the two scoring functions.

1. Pruned confusion network rescoring

Firstly, a set of experiments is conducted on the pruned confusion networks. The log threshold is set to 5. For each bin, if the difference between the maximum log posterior probability and the log posterior probability of a link in the bin is greater than 5, then the link is pruned from that bin, which leaves us a more concise confusion network for rescoring.

The oracle WER of the pruned confusion network is presented in the following table. The error in substitution, insertion, and deletion are 2.83%, 0.03%, and 1.89%. The oracle WER has been increased from 3.36 % in lattices to 5.04%, and the SER has been increased from 33.3% to 43.7%(93 confusion networks do not contain correct hypotheses versus 71 lattices do not contain correct hypotheses)

Table 5.8 Oracle WER of pruned confusion networks

<table>
<thead>
<tr>
<th>Substitution</th>
<th>Insertion</th>
<th>Deletion</th>
<th>WER</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.83%</td>
<td>0.32%</td>
<td>1.89%</td>
<td>5.04%</td>
<td>43.7%</td>
</tr>
</tbody>
</table>

Table 5.9 presents the WER of the decoding results when the pruned confusion network is rescored using the scoring function equation 4.11a and
equation 4.11b. \( \log P(< \text{EPS} >) \), the log language model score for the empty word \(<\text{EPS}>\), is set to -4 and -8, respectively, and the scaling factor for language model score is set to 0.02, 0.04, 0.08, 0.16, 0.25, 0.50, and 1.00, respectively. When appropriate values for the two parameters are set, the confusion network rescoring can decrease WER. For example, when \( \lambda \) and \( \log P(< \text{EPS} >) \) is set to 0.16 and -8 respectively, the WER of the confusion network rescoring is further decreased from 13.3% to 12.8%.

<table>
<thead>
<tr>
<th></th>
<th>0.02</th>
<th>0.04</th>
<th>0.08</th>
<th>0.16</th>
<th>0.25</th>
<th>0.50</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-4)</td>
<td>13.1</td>
<td>13.2</td>
<td>13.1</td>
<td>13.2</td>
<td>13.4</td>
<td>13.7</td>
<td>15.2</td>
</tr>
<tr>
<td>(-8)</td>
<td>13.1</td>
<td>13.1</td>
<td>13.1</td>
<td>12.8</td>
<td>13.2</td>
<td>14.0</td>
<td>16.5</td>
</tr>
</tbody>
</table>

Table 5.10 presents WER of the decoding results when the pruned confusion networks are rescored using the rescoring function equation 4.11, and the language model scaling factor is set respectively to 0.02, 0.04, 0.08, 0.16, 0.25, 0.50, and 1.00. When an appropriate value for \( \lambda \) is set, the confusion network rescoring can decrease WER. When \( \lambda \) is set to 0.16, the WER is decreased from 13.3% to 12.7%.

<table>
<thead>
<tr>
<th></th>
<th>0.02</th>
<th>0.04</th>
<th>0.08</th>
<th>0.16</th>
<th>0.25</th>
<th>0.50</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>WER</td>
<td>13.3</td>
<td>13.2</td>
<td>13.1</td>
<td>12.7</td>
<td>12.8</td>
<td>13.3</td>
<td>14.2</td>
</tr>
</tbody>
</table>

2 Un-pruned confusion network rescoring

Another set of experiments is conducted on the confusion networks without pruning. The oracle WER of the un-pruned confusion networks is presented in the
following table. The error in substitution, insertion, and deletion are 0.75%, 0.03%, and 1.86%, respectively. The oracle WER has been dropped from 3.36% in lattices to 2.64%, and the SER has been dropped from 33.3% in lattices to 30.5% (65 confusion networks do not contain the correct hypotheses versus 71 lattices do not contain the correct hypotheses).

Table 5.11 Oracle WER(%) of un-pruned confusion networks

<table>
<thead>
<tr>
<th>Substitution</th>
<th>Insertion</th>
<th>Deletion</th>
<th>WER</th>
<th>SER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.75%</td>
<td>0.03%</td>
<td>1.86%</td>
<td>2.64%</td>
<td>30.5%</td>
</tr>
</tbody>
</table>

Table 5.12 presents the WER of the decoding results when the un-pruned confusion networks are rescored using the scoring function 4.11a and 4.11b. where \( \log P(<\text{EPS}>), \) the log language model score for the empty word \(<\text{EPS}>, \) and \( \lambda, \) the scaling factor for the language model score, are set to the same values as the experiments with the pruned confusion networks using the scoring function equation 4.11. From the decoding results, we find that, compared with the pruned confusion networks, resoring on un-pruned confusion networks does not get a better result when equation 4.11a and 4.11b are used as the scoring function.

Table 5.12 WER(%) of resoring un-pruned confusion networks (equation 4.11)

<table>
<thead>
<tr>
<th></th>
<th>0.02</th>
<th>0.04</th>
<th>0.08</th>
<th>0.16</th>
<th>0.25</th>
<th>0.50</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>13.1</td>
<td>13.2</td>
<td>13.1</td>
<td>13.2</td>
<td>13.4</td>
<td>13.7</td>
<td>15.8</td>
</tr>
<tr>
<td>-8</td>
<td>13.1</td>
<td>13.1</td>
<td>13.1</td>
<td>12.8</td>
<td>13.2</td>
<td>14.0</td>
<td>17.0</td>
</tr>
</tbody>
</table>
Table 5.13 presents the WER of the decoding results when the un-pruned confusion networks are rescored using the scoring function equation 4.12. The scaling factor for the language model score $\lambda$ is set to the same values as the experiments with the pruned confusion networks using the scoring function equation 4.12. From the results, we find that, compared to pruned confusion networks, resoring un-pruned confusion networks using the scoring function 4.12 does not get a better result.

Table 5.13 WER (%) of rescoring un-pruned confusion networks (equation 4.12)

<table>
<thead>
<tr>
<th>0.02</th>
<th>0.04</th>
<th>0.08</th>
<th>0.16</th>
<th>0.25</th>
<th>0.50</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>WER</td>
<td>13.3</td>
<td>13.2</td>
<td>13.1</td>
<td>12.7</td>
<td>12.7</td>
<td>13.4</td>
</tr>
</tbody>
</table>
Chapter 6 Discussion and Future studies

6.1. Discussion

From the decoding results of A* based lattice rescoring on the WSJ lattices, the SLM alone cannot improve the decoding results, and the linear interpolation of SLM with the trigram language model can improve the decoding results when the linear interpolation coefficient \( \lambda \) is set to 0.2. This result is consistent with the findings in Chelba’s (2000) lattice rescoring with SLM on the same data set. On the other hand, CLM alone is able to decrease the WER from 13.7% to 13.3%, and the linear interpolation of CLM with the trigram language model score at \( \lambda = 0.2 \) achieves the best WER 13.2%. The positive experimental results suggest that CLM is worthy of further studies.

In the experiments of lattice to confusion network transformation, we find that the lattice to confusion network transformation can improve the WER results if appropriate scaling factor for acoustic scores is used. For example, the WER of decoding results is decreased from 13.7% to 13.3% when the scaling factor is set to 12 or 16. In addition, the oracle WER of un-pruned confusion networks is also decreased by 0.7% when the scaling factor is set to 12. This improvement on the decoding results and oracle WER is consistent with existing studies.

In the confusion network rescoring experiments, the confusion network rescoring can further improve the decoding results when the appropriate parameters are set for the two scoring functions. When equation 4.11a and 4.11b are used as the scoring function, the WER of confusion network rescoring on the pruned confusion networks can be decreased to 12.8% when the language model
scaling factor $\lambda$ is set to 0.16 and $\log P(<\text{EPS}>)$ is set to -8. When equation 4.12 is used as the scoring function, the WER of confusion network rescoring on pruned confusion networks can be decreased to 12.7% when the language model scaling factor $\lambda$ is set to 0.16. Compared to pruned confusion networks, the rescoring on un-pruned confusion networks does not get a better result.

6.2. Future studies

Our work is far away from being complete, and there are a lot of future studies that can be done to further improve the performance of speech recognition. We outline some possible directions for future studies.

In practice, in order to speed up the processing, the number of prefixes in the priority queue of $A^*$ search for lattice rescoring is controlled by some parameters, such as the parameters $\text{queue\_depth\_threshold}$ and $\text{queue\_logP\_threshold}$ (see section 3.3) in our study. The prefixes in the priority queue are sorted according to the function $f(x)$ that has two components $g(x)$ and $h(x)$. Prefixes may be pruned if their scores are not big enough. The function $h(x)$ estimates the score for the optimal suffix that starts from the end node of the prefix $x$. $\text{logCOMP}$ that compensates for the difference between the two language model scores is set to a fixed value in the function $h(x)$. If the value is not set appropriately, some nodes may get too much compensation because they have much longer suffixes. Therefore, it is possible that the prefix of the correct hypothesis is pruned because many other prefixes in the priority queue get too much compensation from their ending nodes. It gives rises to two possible directions for futures studies. The first one is to find an appropriate way to estimate the forward score for suffixes, such as estimating an
appropriate value for $logCOMP$ from a held-out dataset or finding a certain dynamic way to compensate for the language model score differences so that it does not overestimate the scores too much. The second one is to find some other search algorithms that can bypass this compensation problem for lattice rescoring.

Another potential direction for further studies is the scoring function in confusion network rescoring. Usually, rescoring is conducted on lattices, and there are not many existing studies about confusion network rescoring. However, confusion network rescoring is a promising direction for future studies because it has smaller oracle WER. The rescoring function is critical to confusion network rescoring. In our study, our scoring function just takes the following two features of confusion networks into consideration: the posterior probabilities of the confusion network links and the language model scores from a sophisticated language model. Actually, there are more features that can be kept when lattices are transformed into confusion networks, such as the acoustic score, the time points of starting and ending nodes of each bin, the pronunciation of words in each link, and the decoding results of the Viterbi decoder. We think all these features may contribute to confusion network rescoring. A key question is to find a way to integrate these features into an appropriate scoring function. Along this direction, Maximum entropy appears to be a promising framework for integrating these features.
Reference


*Computational linguistics, 13*(1-2), 31-46.


*Journal of the ACM (JACM), 21*(1), 168-173.


