# LANGUAGE MODELING FOR TURKISH CONTINUOUS SPEECH RECOGNITION

**A THESIS SUBMITTED TO**

**THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

**OF**

**MIDDLE EAST TECHNICAL UNIVERSITY**

**BY**

**SERKAN ŞAHİN**

**IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE**

**DEGREE OF**

**MASTER OF SCIENCE**

**IN THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS**

**ENGINEERING**

**DECEMBER 2003**

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

_____

Prof. Dr. Mübeccel Demirekler

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Tolga Çiloğlu

Supervisor

Examining Committee Members:

Prof. Dr.Mübeccel Demirekler _____

Assoc. Prof. Dr. Engin Tuncer _____

Assoc. Prof. Dr. Tolga Çiloğlu _____

Assoc. Prof. Dr. Buyurman Baykal _____

M.Sc. Onur Çilingir _____

# ABSTRACT

LANGUAGE MODELING FOR TURKISH CONTINUOUS SPEECH RECOGNITION

**ŞAHİN, Serkan**

**M.S., The Department of Electrical and Electronics Engineering**

**Supervisor: Assoc. Prof. Dr. Tolga Çiloğlu**

**December 2003, 93 pages**

This study aims to build a new language model for Turkish continuous speech recognition. Turkish is very productive language in terms of word forms because of its agglutinative nature. For such languages like Turkish, the vocabulary size is far from being acceptable from only one simple stem, thousands of new words can be generated using inflectional and derivational suffixes. In this work, word are parsed into their stem and endings. First of all, we consider endings as words and we obtained bigram probabilities using stem and endings. Then, bigram probabilities are obtained using only the stems. Single pass recognition was performed by using bigram probabilities. As a

second job, two pass recognition was performed. Firstly, previous bigram probabilities were used to create word lattices. Secondly, trigram probabilities were obtained from a larger text. Finally, one-best results were obtained by using word lattices and trigram probabilities. All work is done in Hidden Markov Model Toolkit (HTK) environment, except parsing and network transforming.

**Keywords:** Speech Recognition, continuous speech, language model, bigrams, trigrams, two pass recognition, stem, ending, parsing, Turkish morphology.

# ÖZ

TÜRKÇE AKAN KONUŞMA TANIMA İÇİN DİL MODELLEMESİ

**ŞAHİN, Serkan**

**Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü**

**Tez Yöneticisi: Assoc. Prof. Dr. Tolga Çiloğlu**

**Aralık 2003, 93 sayfa**

Bu çalışmada, Türkçe için akan konuşma tanıma sisteminde kullanılacak yeni bir dil modeli geliştirilmesi amaçlanmıştır. Türkçe, sondan eklemeli özelliğinden dolayı kelime biçimleri açısından çok üretken bir dildir. Böylesi diller için dağarcık boyutu kabul edilebilir olmaktan uzaktır. Basit bir gövde kullanarak, yapım ve çekim ekleri sayesinde binlerce yeni biçimli kelime üretilebilir.Bu çalışmada sözcükler gövde ve eklerine ayrılmışlardır. İlk olarak, ekler kelime gibi kabul edilip ikili kelime olasılıkları elde edilmiştir. Daha sonra, ikili kelime olasılıkları yalnızca gövdeler üzerinden elde edilmiştir. Bu ikili kelime olasılıkları kullanılarak tek geçişte tanıma

işlemi gerçekleştirilmiştir. İkinci çalışma olarak, iki geçişli tanıma işlemi gerçekleştirilmiştir. Bunun için öncelikle daha önce elde edilen ikili kelime olasılıkları kullanılarak kelime latisleri oluşturulmuştur. İkincil olarak daha geniş bir metin dosyasından üçlü kelime olasılıkları elde edilmiştir. Son olarak bu kelime latisleri ve üçlü kelime olasılıkları kullanılarak en iyi kelime dizisi ortaya çıkarılmaya çalışılmıştır. Gövde-ek ayırma ve ağ dönüştürme işlemleri dışında tüm işlemler Hidden Markov Model Toolkit (HTK) ile gerçekleştirilmiştir.

**Anahtar Sözcükler:** Konuşma tanıma, akan konuşma, dil modeli, kelime ikilisi, kelime üçlüsü, iki geçişli tanıma, gövde, ek, parçalama, Türkçe biçimbilim.

## ACKNOWLEDGMENTS

I would like to thank my advisor, Assoc. Prof. Dr. Tolga Çiloğlu, for his guidance, leadership and support. My colleagues; Dinç Acar, who help me about HTK and M. A. Çömez, who help me about every subject.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

FIGURE

# CHAPTER 1

# INTRODUCTION

## 1.1 Speech Recognition Basics

Speech recognition is the process performed by a computer (or other type of machine) which identifies the spoken words. This process is sometimes referred to as speech-to-text. Some terms must be known for understanding speech recognition technology. These terms are: utterance, phoneme, speaker dependent/independent systems, vocabulary, accuracy, and adaptation.

The smallest unit in speech is called *phoneme*, which does not have any meaning alone. An *utterance* is any stream of speech between two silences. Utterances can be a single word, a couple of words, a sentence or even multiple sentences. Silence is almost as important as spoken words in speech recognition. Because it takes part at the beginning and at end of an utterance.

*Speaker dependent* systems are designed according to a specific speaker. These systems are more accurate for that specific speaker, but less accurate for other speakers. They assume that speaker will speak in a consistent voice and tempo.

*Speaker independent* systems are designed for variety of speakers. These systems are most difficult to develop, most expensive and lower accuracy compared to speaker dependent systems.

*Vocabularies* (or dictionaries) are list of words or utterances that can be recognized by the speech recognition system. Generally, smaller vocabularies are easier to be recognized by a computer. The size of vocabulary of a speech recognition system affects the complexity, processing requirements and the accuracy of the system. Some applications, which needs to recognize only telephone numbers, require a few words, others like dictation machines require very large vocabularies. Even though there are no established definitions, we can say

- Small vocabulary – tens of words
- Medium vocabulary – hundreds of words
- Large vocabulary – thousands of words
- Very-large vocabulary – tens of thousands of words.

The performance of a speech recognition system is measurable. *Accuracy* is one of the most widely used criterion, it is the measurement how well a recognizer recognizes the spoken utterances. This is typically a quantitative measurement which can be calculated in several ways. It also includes identifying if the spoken words is not in its vocabulary. A good ASR system has an accuracy of 98% or more.

In order to improve accuracy, some speech recognizer systems have ability to *adapt* to speaker's voice and speaking style.

## 1.2 Types of Speech Recognition

Speech recognition system can be categorized to different classes according to the ability that they have for recognizing. The classes are based on the fact that the ability to determine when a speaker starts and finishes an utterance, which is one of the difficulties of ASR. These classes can be categorized as follows.

**Isolated Word Recognition**: Isolated recognition in general means recognition of speech based on any kind of isolated speech unit, which can be a word or a sub word or even a concatenation of words. Isolated word recognizers usually require each utterance to have silence (lack of an audio signal) on both sides. An isolated-word system operates on single words at a time - requiring a pause between saying each word. This is the simplest form of recognition to perform because the end points are easier to find and the pronunciation of a word tends not affect others. Thus, because the occurrences of words are more consistent they are easier to recognize.

**Connected Word Recognition:** Connected words recognition can be considered as similar to isolated words recognition. The difference is connected words recognition requires minimal pauses between utterances, but isolated words recognition allows separate utterances. In connected words recognition, longer

phrases are possible to recognize, and the necessary computation increases as a result.

**Continuous Speech Recognition:** Recognizers with continuous speech must utilize special methods to determine utterance boundaries, thus this type is the most difficult one. A continuous speech system operates on speech in which words are connected together, i.e. not separated by pauses. Continuous speech is more difficult to handle because of a variety of effects. First, it is difficult to find the start and end points of words. Another problem is co-articulation. The production of each phoneme is affected by the production of surrounding phonemes, and similarly the start and end of words are affected by the preceding and following words. The recognition of continuous speech is also affected by the rate of speech, i.e. fast speech tends to be harder. Continuous speech recognizers allow users to speak almost naturally.

## 1.3 Outline of the Thesis

In Chapter 2, fundamentals of speech recognition will be given. Feature extraction, acoustic modeling and Hidden Markov Modeling (HMM) will be explained in detail.

In Chapter 3, Language modeling will be explained in general and then Turkish morphology and language modeling for agglutinative languages will be discussed.

In Chapter 4, basic search (decoding) techniques will be given and multipass search will be explained in detail.

In Chapter 5, language model used in this thesis, which is core of this thesis, will be given and statistics of training corpus and the experimental results are given in tables.

Chapter 6 concludes this thesis and describes ideas for future work.

# CHAPTER 2

# FUNDAMENTALS OF SPEECH RECOGNITION

## 2.1 Overview of Speech Recognition

Approaches to speech recognition can be divided into different categories [1]. In general there are three approaches: 1) *the acoustic-phonetic approach*,2) *the statistical recognition or pattern recognition* and 3) *the artificial intelligence (AI) approach*. The acoustic-phonetic method is the oldest approach to speech recognition and AI is the youngest and the least known. Statistical methods are by far most appropriate to model speech. Especially, Hidden Markov models (HMMs) are most exclusively used in ASR. The statistical methods consist of following stages: 1) *acoustic processing (feature extraction),* 2) *acoustic modeling,* 3) *language modeling* and *4) hypothesis search (or decoding).*

### 2.1.1 Definition of the Problem

In ASR, first procedure to be done is to convert an unknown speech waveform into a sequence of acoustic vectors, i.e. $O = o_1, o_2, ..., o_T$ , by a front-end signal processor. Each of these vectors is a compact representation of the short-time speech spectrum. It is assumed that the utterance consists of a sequence of words, $W = w_1, w_2, ..., w_n$ . The duty of an ASR system is to find the most probable string of words $W$ corresponding to an acoustic signal $O$. We can write this as an equation:

$$\hat{W} = \arg\max_{W} P(W/O) \tag{2.1}$$

This means that the probability of every sequence of words $W$ need to be computed given the acoustics $O$ and the most probable one will be chosen as the recognized output. To achieve this, we use Bayes' rule to decompose the Equation(2.1) into two components;

$$\hat{W} = \arg\max_{W} \frac{P(W)P(O/W)}{P(O)} \tag{2.2}$$

Since $P(O)$ is the probability of the data independent of the word string $(W)$, we can simplify this further:

$$P(W/O) \propto P(O/W) \quad P(W) \tag{2.3}$$

The first term, $P(O/W)$, denote the probability of observing the vector sequence $O$ given some specified word sequence $W$ and this probability is determined by an

*acoustic model* which is estimated from a speech corpus. As stated earlier, an HMM system is used to model the acoustic probability.

The second term, $P(W)$, represents the prior probability of a sequence of words. Since this probability does not depend on the acoustic data $O$, it can be estimated from a large text corpus. This probability is known as *language model* probability.

Figure 2.1 shows the process described above.

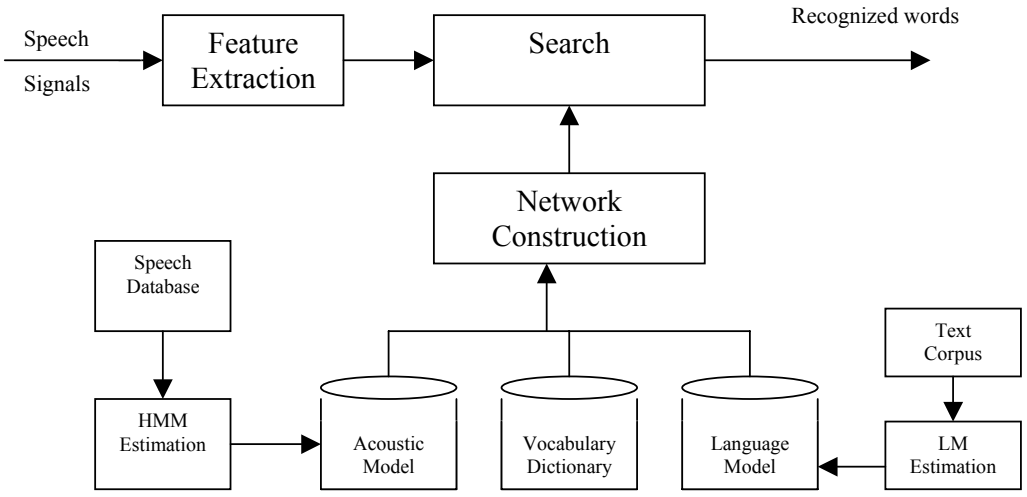



Figure 2.1: Speech recognition system block diagram. The vocabulary, language model, and acoustic model components construct a hypothesis for interpreting a speech sample.

- the vocabulary defines the possible words that the search can hypothesize, representing each word as a linear sequence of phonemes;
- the language model models the linguistic structure but does not contain any knowledge about the relationship between the feature vectors and the words
- the acoustic model models the relationship between the feature vectors and the phonemes

### 2.1.2 Speech Signal Processing and Feature Extraction

Parametric representation of the speech waveform must be obtained for further analysis and processing [2]. For doing this it is assumed that the speech signal can be regarded as stationary (i.e. spectral characteristics are relatively constant) over a short period of time. Therefore, when the speech signal is divided into frames (i.e. block of samples) it can be considered stationary and spectral analysis can be performed with this frame methodology. The spacing between blocks is typically 10ms. Blocks are normally overlapped to give a longer analysis window, typically 25ms. It is usual to multiply each block by a tapered window function (e.g. Hamming window) so as to minimize the adverse effects of chopping the speech signal as blocks. Before all, the raw signal is pre-emphasized by applying high frequency amplification to compensate for the attenuation caused by the radiation from the lips.

Linear predictive (LP) modeling or Fourier analysis or Cepstrum analysis can be used to compute the spectral estimates and the coefficients and the final acoustic vectors can be obtained using a number of transformations [1]. Linear predictive modeling can be considered as one of the most important analysis techniques. It is based on the fact that a speech sample can be approximated as a linear combination of past samples and exploits the sample-to-sample correlations among speech samples. Cepstral coefficients is another short-time representation of speech and computed either directly from FFT spectra or from LP coefficients. Cepstrum coefficients calculation using DTFT is shown below.

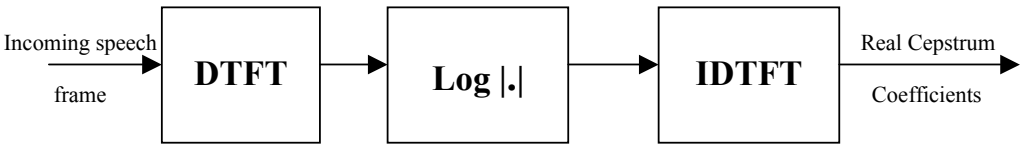



Figure 2.2: Computation of real cepstrum with DTFT

#### 2.1.2.1 Mel Frequency Cepstrum Coefficients

Over the past 40 years of speech recognition research, many feature extraction techniques has been tried to find the reliable representation of speech waveform. Human auditory system simulates the logarithmic frequency scale not the linear frequency scale. Cepstrum analysis does not consider the structure of human auditory system so that it can not be a good parameter set. Recognition performance can be improved by incorporating this knowledge into the parameter set. One of the popular scale which consider human auditory system is Mel scale.

HTK provides a simple Fourier transform based filterbank which is designed to produce a solution similar to a mel-scale [3]. Figure 2.3 shows the general form of this filterbank.

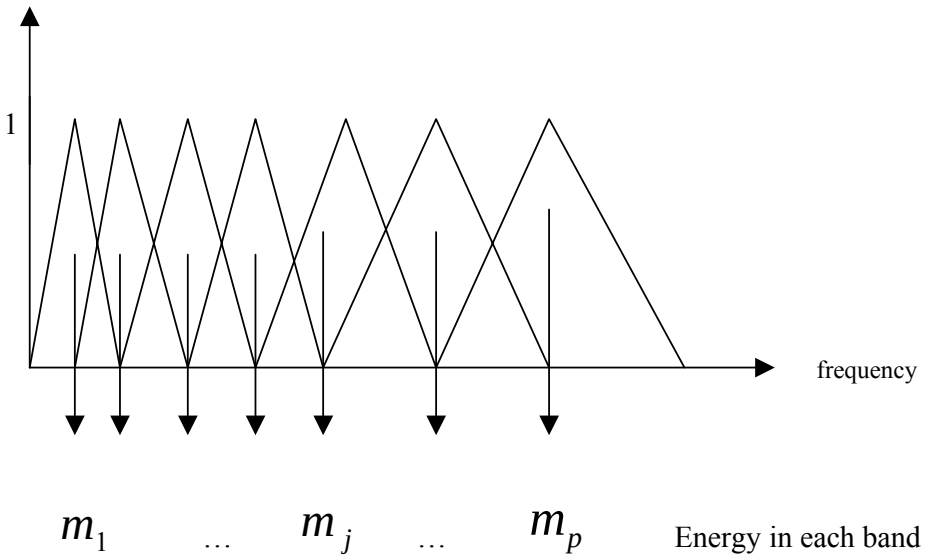



Figure 2.3: Mel-Scale filterbank

An approximate conversion to Mel scale is given in (2.4).

$$F_{mel} = 2595 \log_{10}(1 + \frac{F_{Hz}}{700}) \tag{2.4}$$

$F_{Hz}$ = Frequency in Hertz,

$F_{mel}$ =Corresponding Mel frequency.

In computation of the coefficients, each of the filter produces a log energy coefficient that represents the energy in that band. Than inverse DFT is performed to get back to cepstral domain and obtain mel-cepstral coefficients. This procedure summarized in Figure 2.4.

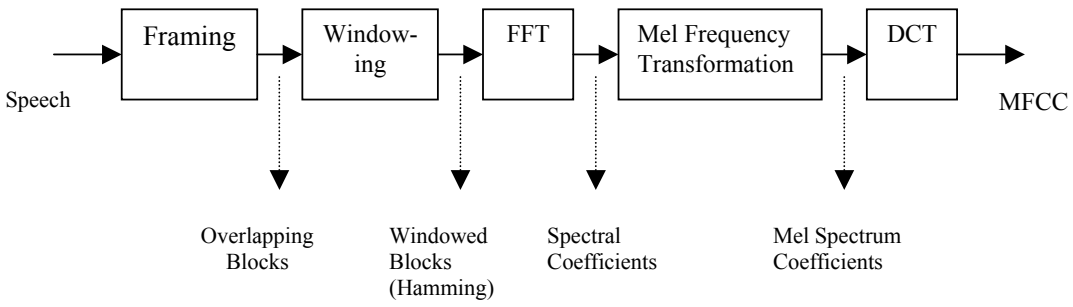



Figure 2.4: Mel-Cepstrum Block Diagram

The feature vector used for speech recognition is typically a combination of 12 cepstral and one energy coefficient plus the delta coefficients (+13) plus the acceleration coefficients (+13).

### 2.1.3 Acoustic Modeling

Acoustic modeling plays a critical role in improving accuracy and is the central part of any speech recognition system. Acoustic modeling quality is affected various causes of contextual variation. Some of these variations are due to the environment (is it noisy? are there multiple speakers?), the channel (telephone? wide band microphone?), the speaker (male or female? age?), or the speaking style (dictated or spontaneous?). In order to minimize these effect acoustic modeling corpus must me chosen quite large.

After obtaining the feature vectors belonging to speech waveform, Hidden Markov modeling (HMM) can be used to model the subword units.

### 2.1.3.1 Hidden Markov Modeling

As stated earlier, the most popular approach to acoustic modeling is by use of HMMs. The HMM is a doubly stochastic state machine that has a Markov distribution associated with the transitions across various states and a probability density function that models the output for every state. Depending on the complexity of the recognition problem, this distribution can be modeled as a discrete-valued or continuous-valued process [4].

Each HMM represents a unit of sound, such as a word or a sub-word unit (e.g. phoneme) that is to be recognized. Each model is constructed using statistics calculated from examples of the speech unit that the model represents. This process is referred to as ‘ training’ the models. The generation process of an HMM can be depicted as shown in Figure 2.5.

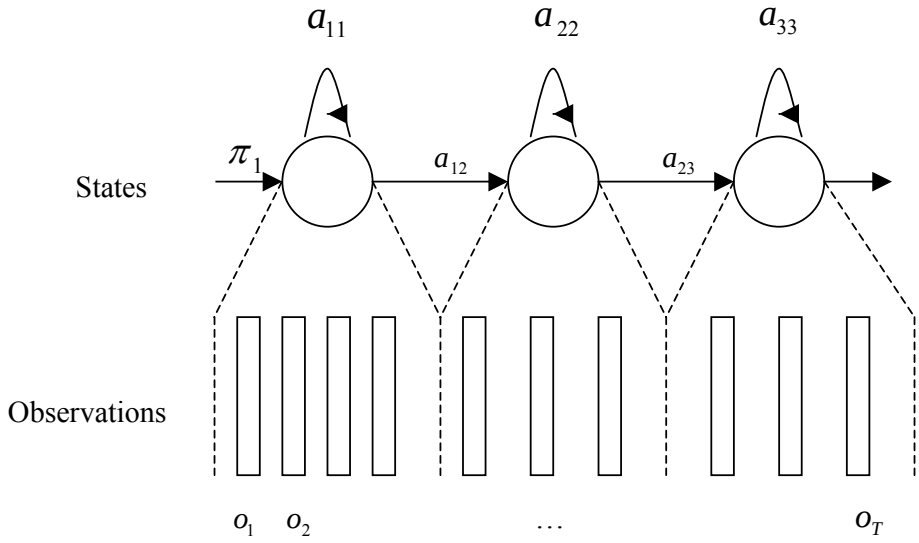



Figure 2.5: Representation of a three state HMM and a possible sequence of generated observations.

We can define HMM parameters as follows [5]:

$O = \{o_1, o_2, ..., o_T\}$ : The observation sequence correspond to the output of the system being modeled.

$S = \{s_1, s_2, ..., s_T\}$ : A sequence of states representing the state space. Here $s_t$ represents the state at time *t*.

$a_{ij} = P(s_t = j \mid s_{t-1} = i)$ : The probability of taking a transition from state *i* to state *j*. $1 \le i, j \le N$ .

$A = \{a_{ij}\}$ : Probability transition matrix.

$b_i(o_t) = P(o_t \mid s_t = i)$ : The probability of observing the symbol $o_t$ when being in state *i* at time *t.*

$B = \{b_i(o_t)\}$ : An output probability matrix (diagonal matrix with diagonal entries).

$\pi = \{\pi_i\}$ : A initial state distribution.

$\Phi = (A, B, \pi)$ indicates the whole parameter set of an HMM.

Given the definition of HMM above, three basic problems of it must be stated before its application.

**1. The Evaluation Problem:** Given an HMM $\Phi$ and a sequence of observations $O = \{o_1, o_2, ..., o_T\}$ , what is the probability that the observations are generated by the model, $P(O \mid \Phi)$ ?

**2. The Decoding Problem:** Given a model $\Phi$ and a sequence of observations $O = \{o_1, o_2, ..., o_T\}$, what is the most likely state sequence $S = \{s_1, s_2, ..., s_T\}$ in the model that produces the observations?

**3. The Learning Problem:** Given a model $\Phi$ and a sequence of observations $O = \{o_1, o_2, ..., o_T\}$, how should we adjust the model parameters $(A, B, \pi)$ in order to maximize $P(O|\Phi)$?

#### 2.1.3.1.1 How to Evaluate an HMM – The Forward Algorithm

We have a model $\Phi = (A, B, \pi)$ and a sequence of observations $O = \{o_1, o_2, ..., o_T\}$, and $P(O|\Phi)$ must be found. In order to do this, the probability for observing the sequence up to a given feature vector is computed over all possible state sequences (Figure 2.6).

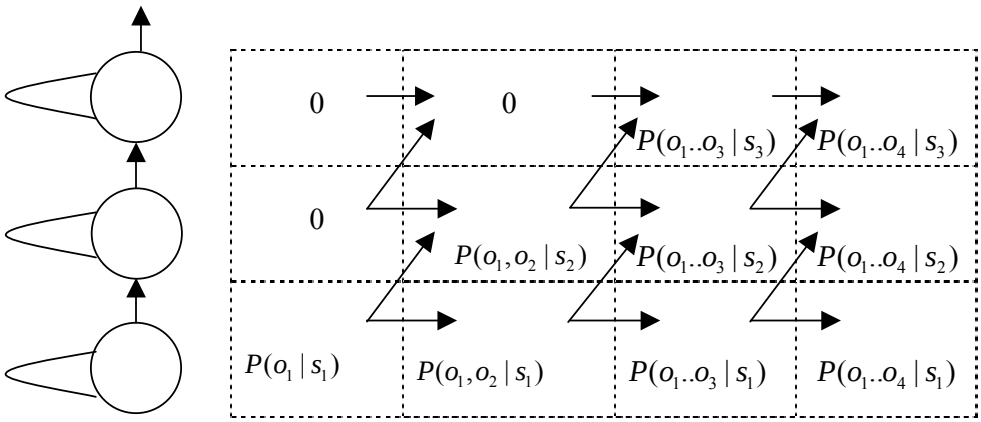



Figure 2.6: Example of the *forward algorithm.*

When processing the first feature vector, the probability for being in the first state

of the chain must be one and the probability for being in any other state must be zero, since it is not allowed to start the sentence in the middle of a word.

Summing over all possible state sequences requires summing over $N^T$ possible sequences. Instead, we can consider calculating the *forward variable* $\alpha_t(i)$ defined as the probability of the partial observation sequence $o_1, o_2, ..., o_t$ ending in state $s_i$ at time *t*.

$$\alpha_t(i) = P(o_1 o_2 ... o_t, s_t = i \mid \Phi)$$

We can solve $\alpha_t(i)$ recursively as follows

**Step 1: Initialization**

$$\alpha_t(i) = \pi_i b_i(o_1) \qquad 1 \le i \le N$$

**Step 2: Induction**

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(o_{t+1}) \qquad 1 \le t \le T-1$$

$$1 \le j \le N$$

**Step 3: Termination**

$$P(O \mid \Phi) = \sum_{i=1}^{N} \alpha_T(i)$$

In a similar way we can define *backward variable* $\beta_t(i)$ defined as the probability of the partial observation $o_{t+1}, o_{t+2}, ..., o_T$ after starting in state $s_i$ at time *t* until the end of the observation sequence:

$$\beta_t(i) = P(o_{t+1} o_{t+2} ... o_T \mid s_t = i, \Phi)$$

Backward procedure is similar to forward procedure:

**1.Initialization:**

$$\beta_T(i) = 1 \qquad 1 \le i \le N$$

**2.Induction:**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \qquad 1 \le t \le T-1$$

$$1 \le i \le N$$

Both the forward and the backward procedure can be used to solve the evaluation problem. Besides, the product of these probabilities gives another way to calculate $P(O|\Phi)$.

### 2.1.3.1.2 How to Decode an HMM – The Viterbi Algorithm

The forward algorithm computes the probability that an HMM generates an observation sequence by summing up the all possible path probabilities, so it does not provide the best path (or state sequence). As a matter of fact, finding the best state sequence is the cornerstone for continuous speech recognition. This problem is similar to to find the optimal path in dynamic programming. Viterbi algorithm which is based on dynamic programming can be used to solve this problem.

The Viterbi algorithm can be regarded as modified forward algorithm or as a dynamic programming algorithm applied to HMM. It stores and then remembers the optimal state sequence instead of summing up all possible path probabilities coming to the same destination state. Viterbi algorithm is said to be time synchronous

because it completely processes at time *t* before going into the time *t*+1. Finally a backtracking pass gives the required state sequences. This procedure is shown in Figure 2.7.

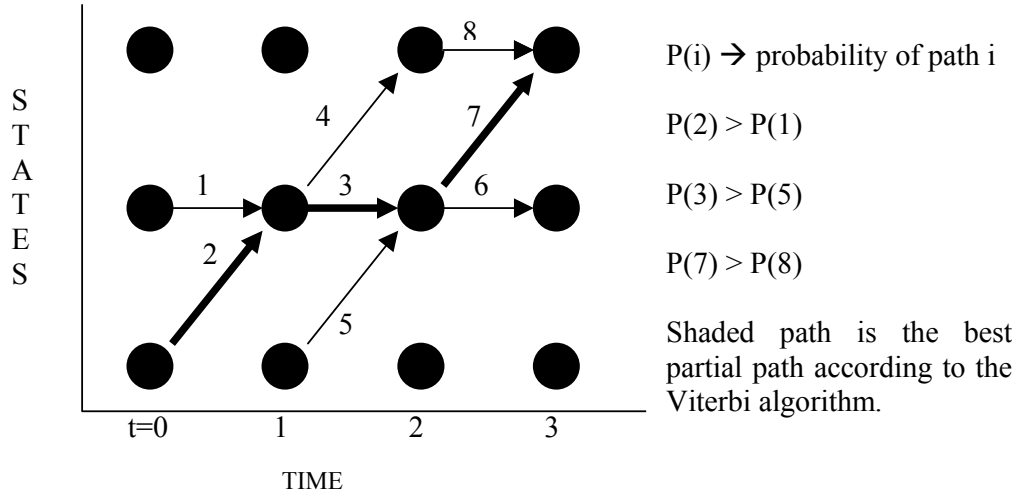



Figure 2.7: Example of the Viterbi algorithm.

Before giving the complete algorithm, we must define an auxiliary variable,

$$\delta_t(i) = \max_{s_1, s_2, \ldots s_{t-1}} (P[s_1, s_2, \ldots, s_t = i, O \mid \Phi])$$

which gives the highest probability that partial observation sequence and state sequence up to time *t* can have, when the current state is *i*. Also need to store the state argument at time t-1, from which the best transition is made to the current state. This will be kept in the vector $\psi_t(j)$. The complete Viterbi algorithm is as follows:

**1.Initialization:**

$$\delta_1(i) = \pi_i b_i(o_1) \qquad 1 \le i \le N$$
$$\psi_1(i) = 0$$

**2.Recursion:**

$$\delta_t(j) = \max_{1 \le i \le N}(\delta_{t-1}(i)a_{ij})b_j(o_t)$$
$$\psi_t(j) = \arg\max_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}] \qquad 2 \le t \le T, \quad 1 \le j \le N$$

**3.Termination:**

$$P^* = \max_{1 \le i \le N}[\delta_T(i)]$$
$$q_T^* = \arg\max_{1 \le i \le N}(\delta_T(i))$$

**4.State sequence backtracking recovery:**

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, ..., 1.$$

Using max function instead of sum function is the big advantage of the Viterbi algorithm over the forward algorithm [6]. Therefore it is possible to work with logarithm of all probabilities. The resulting logarithmic score range stays within the limitations of most computers.

### 2.1.3.1.3 The Learning Problem and Baum-Welch Algorithm

This the most difficult of the three problems. The problem can be solved by *Baum-Welch* algorithm, also known as the *forward-backward* algorithm. To describe the Baum-Welch algorithm, we need to define two more auxiliary variables, in addition to the forward and backward variables defined in a previous section. These variables can however be expressed in terms of the forward and backward variables.

First one of those variables is defined as the probability of being in state *i* at time *t* and in state *j* at time *t*+1. Formally,

$$\xi_t(i,j) = P(s_t = i, s_{t+1} = j \mid O, \Phi)$$

This is the same as,

$$\xi_t(i,j) = \frac{P(s_t = i, s_{t+1} = j, O \mid \Phi)}{P(O \mid \Phi)}$$

Using forward and backward variables this can be expressed as,

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})}$$

The second variable is the a posteriori probability,

$$\gamma_t(i) = P(s_t = i \mid O, \Phi)$$

that is the probability of being in state *i* at time *t*, given the observation sequence and the model. In forward and backward variables this can be expressed by,

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

One can see that the relationship between $\gamma_t(i)$ and $\xi_t(i)$ is given by,

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

The expected number of transition from state $i$ is then, $\sum_{t=1}^{T-1} \gamma_t(i)$

The expected number of transitions from state *i* to state *j* is then,

$$\sum_{t=1}^{T-1} \xi_t(i, j)$$

We can estimate new values of the parameters given the observation as:

$\overline{\pi}$ = expected frequency of being in state *i* at time =1. This is equal to $\gamma_1(i)$.

$$\overline{a_{ij}} = \frac{\text{Expected number of transitions from state } i \text{ to state } j}{\text{Expected number of transitions from state } i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\overline{b_j(k)} = \frac{\text{Expected number of times in state } j \text{ observing } v_k}{\text{Expected number of times in state } j}$$

$$= \frac{\sum_{\substack{t=1 \\ o_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Hidden Markov models have some assumptions about the structure of the process that they represent however these are not necessarily true for speech signals [7]. These are;

1) The observations accurately represent the signal.
2) The observations are independent of each other.
3) The between state transition probabilities are constant.

### 2.1.3.2 Modeling Context

Since speech is produced by articulatory movements, the production of a phone is strongly affected by both the preceding and the following phones. When speaking there are not discrete jumps between phones, there is a motion when passing one phone to another [8]. This phenomenon is called as *coarticulation*. Coarticulation means that the acoustics of a phone may be reasonably consistent when surrounding phonetic context is taken into account, compared with when a single model is used for the phone for all context. Therefore, in speech recognition it is taken into account to improve accuracy.

*Context-dependent* phone models are the most usual way to model the coarticulation effects. It involves using several basic subword units for each phone instead of one unit. Contexts used in speech recognition can be defined as follows:

**Monophone context**: This does not consider the surrounding context. In Turkish this means 32 phones (plus silence) are required. The word "kelime" can be represented using monophones as "k e l i m e".

**Allophone context:** this also involves no context, but there are multiple possible units for each phone. Using same example: Kelime = k(2) e(1) l(2) i(1) m(1) e(1).

**Biphone context:** In these case each phone is represented with a particular left or right context. There are possible 32 x 33 = 1056 left or right biphone context phone models (plus silence). Kelime = "$-k k-e e-l l-i i-m m-e" (left biphones) and "k+e e+l l+i i+m m+e e+$" (right biphones). (Note that $: word boundary, -: a left context, +: a right context.)

**Triphone context:** Both left and right context are used in order to represent each phone. There are possible 32 x 33 x 33 = 34848 triphone context dependent models (plus silence). Kelime = "$-k+e k-e+l e-l+i l-i+m i-m+e m-e+$".

In modern continuous speech recognition systems, the most popular context dependent subword unit employed is the triphone model. In this thesis triphone modeling is used. We also consider what to do at word boundaries. There are two basic possibility for this:

**Cross word models:** In this case phonetic context is taken into account across word boundaries. For example:

Kelime tanıma: $-k+e e-l+i l-i+m i-m+e m-e+t e-t+a t-a+n a-n+ı n-ı+m ı-m+a m-a+$.

**Word internal models:** phonetic context across word boundaries is not used. For example:

Kelime tanıma: $-k+e e-l+i l-i+m i-m+e m-e+$ $-t+a t-a+n a-n+ı n-ı+m ı-m+a m-a+$.

In continuous speech there aren't silences between words, therefore cross word models is important. On the other hand, there are some advantages of the word internal case: The number of the contexts in cross word is more than word internal case since new context are introduced. The words in word internal case are considered to be in isolation. This property makes the decoding (search) process easier compared to cross word case.

Even though the training data is quite large, it is impossible that it contains all contexts in a language. For a good speech recognition system it is not acceptible not to model the contexts that do not occur in the training data. One of the solution of this problem is *backing-off*. In this case, if there is not enough example to train a triphone context dependent model then a less specific biphone context dependent model is used instead. In the same way, the context dependent monophone model is used for biphone context dependent model that does not have enough data to train.

However, relatively few data are trained by using full triphone context when data is sparse (especially cross word case) and this causes loss of some advantage of the cross word property. *Sharing* is another solution to data sparseness problem. Rather than backing-off to less specific models, it is based on sharing models between different context [9]. This approach means that context dependency is maintained while each model has enough training data.

#### 2.1.3.3 Shared Context Dependent Models

Sharing scheme can be divided into two approaches:

**Bottom-up approach:** This begins by assigning a triphone context dependent model to all models. Since not all of these models will have enough training data, a merging process is used to find the closest models by using some distance measure. The main drawback of this approach is that training examples are required for all initial models. So the unseen context in the training data can not be modeled. Therefore, backing-off procedure is used when encountered unseen context during recognition process.

**Top-down approach:** This initially assumes that all the context are the same and are grouped. Then a splitting procedure is used to get more accurate and specific models. One way of realizing this is *decision tree*. The root of the tree is a context independent model. Each node of the tree contains a binary question about the context to get more specific models by splitting the current model (e.g. “is the left context a nasal?” or “is the right context /l/?”). The leaves of the tree correspond to

resultant context dependent models. That the every context is modelled by a context dependent model is an advantage of this approach. The key constraints for constructing a decision tree are:

- Each leaf (context dependent model) must have a minimum number of training examples.
- A finite set of questions must be chosen to split each node.
- The resultant leaves must be able to be well modelled by HMMs.

# CHAPTER 3

# LANGUAGE MODELING

## 3.1 Languge Modeling

Language modeling is required for large vocabulary continuous speech recognition to select the most likely word sequence from a large number of alternative word hypotheses produced during the decoding (search) process. Since the word boundaries in the continuous speech are not known, several word hypotheses are produced in addition to intended or correct ones.

Statistical language modeling is concerned with the probability of naturally occuring word sequences in a language. It is essential to reduce the complexity of the recognition task. Its job is simply to put high probability on word sequences that are more likely to appear and low probability on word sequences that are less likely to appear. Given a word sequence $w_1 w_2 ... w_N$ to be used as a test corpus, perplexity and entropy scores can be used to measure the quality of language model.

$$Perplexity = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 ... w_{i-1})}}$$

The perplexity of a language model is defined as the reciprocal of the (geometric) average probability assigned by the model to each word in the test set [5]. Entropy score is related to the perplexity as follows:

$$Entropy = \log_2 Perplexity$$

If a language has a higher perplexity, then this means that the number of words branching from a previous word is larger on average. Therefore, we can consider the perplexity as an indication of the complexity of a language. The perplexity of a particular language can affect vocabulary size, grammar rules and estimated probabilities. The goal is to obtain small values of these measures.

### 3.1.1 N-gram Language Models

The most commonly used, simplest and successful technique for constructing a language model for a language is the N-gram model. We can write the probability of any sequence of words, $w_1 w_2 \ldots w_N$, (or the probability of a sentence) as follows:

$$\begin{aligned} P(w_1 w_2 \ldots w_N) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)P(w_4 \mid w_1 w_2 w_3)\ldots P(w_N \mid w_1 \ldots w_{N-1}) \\ &= \prod_{i=1}^{N} P(w_i \mid w_1 \ldots w_{i-1}) \end{aligned}$$

The word sequence, $w_1 w_2 \ldots w_{i-1}$ ,is said to be the *word history* or simply *history* for the word $w_i$. An N-gram model approximates the probability of a word occurrence depends only on the previous n-1 words:

$$P(w_i \mid w_1 \ldots w_{i-1}) = P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$

Unigram, bigram and trigram are the most commonly used N-grams. These are defined respectively as follows:

$P(w_i)$ = probability of word $w_i$ ,

$P(w_i \mid w_{i-1})$ = probability of $w_i$ given a one word history $w_{i-1}$ ,

$P(w_i \mid w_{i-1}w_{i-2})$ = probability of $w_i$ given a two word history $w_{i-1}$, $w_{i-2}$ .

Maximum likelihood estimate of N-grams can be calculated from simple frequency counts. General likelihood formula for an N-gram model is

$$\hat{P}(w_i \mid w_{i-n+1}...w_{i-1}) = \frac{\#(w_{i-n+1}...w_i)}{\#(w_{i-n+1}...w_{i-1})}$$

For a trigram case (N=3) the likelihood

$$\hat{P}(w_i \mid w_{i-1}w_{i-2}) = \frac{\#(w_{i-2}w_{i-1}w_i)}{\#(w_{i-2}w_{i-1})}$$

where $\#(xyz)$ indicates the number of occurrences of trigram $xyz$ in the training corpus and $\#(xy)$ is the number of occurrences of bigram $xy$. For a vocabulary that has $V$ words, there are possible $V^3$ trigram. Many of them will not appear in the training corpus or appear several times. This means estimation using above equation will be a poor estimate. We can say there is sparse data problem.

*Discounting* and *back-off* methods can be used to alleviate the sparse data problem [10]. In discounting methods, the trigram counts of more frequently occurring trigrams are reduced and the resulting excess probability is distributed amongst the less frequently occurring trigrams. In back-off method, the probability of trigrams (or bigrams) that occur few times are replaced by a scaled bigram (or

unigram) probability and same procedure is applied for bigrams which appear few times in the training corpus.

### 3.1.2 Discounting Methods

This section describes some discounting methods for removing the data sparsity problem[29]. The term $n_r$ denotes the number of words occurring $r$ times.

#### 3.1.2.1 Absolute Discounting

In this method, the frequency of a word is reduced by a constant *c*. Where *c* is often defined as the upper bound:

$$c = \frac{n_1}{n_1 + 2n_2}$$

So the probability of word $w_i$ given the history $w_{i-n+1}...w_{i-1}$ is calculated as:

$$\hat{P}(w_i \mid w_{i-n+1}...w_{i-1}) = \frac{\#(w_{i-n+1}...w_i) - c}{\#(w_{i-n+1}...w_{i-1})}$$

#### 3.1.2.2 Linear Discounting

In this method, the probability of $w_i$ given $w_{i-n+1}...w_{i-1}$ is calculated as:

$$\hat{P}(w_i \mid w_{i-n+1}...w_{i-1}) = \alpha \frac{\#(w_{i-n+1}...w_i)}{\#(w_{i-n+1}...w_{i-1})}$$

where scaling factor $\alpha$ is defined as:

$$\alpha = 1 - \frac{n_1}{C}$$

where *C* is the number of events (unigrams).

### 3.1.2.3 Good-Turing Discounting

In this method, the count of an event occurring $r$ times is discounted with

$$GT_r = (r+1)\frac{n_{r+1}}{n_r}$$

The probability of $w_i$ given $w_{i-n+1}...w_{i-1}$ is calculated as:

$$\hat{P}(w_i \mid w_{i-n+1}...w_{i-1}) = \frac{GT_{\#(w_{i-n+1}...w_i)}}{\#(w_{i-n+1}...w_{i-1})}$$

### 3.1.2.4 Witten-Bell Discounting

This is similar to the linear discounting in which the probability is calculated as:

$$\hat{P}(w_i \mid w_{i-n+1}...w_{i-1}) = \alpha \frac{\#(w_{i-n+1}...w_i)}{\#(w_{i-n+1}...w_{i-1})}$$

where α is defined differently as:

$$\alpha = 1 - \frac{N}{\#(w_{i-n+1}...w_{i-1}) + N}$$

where N is the number of distinct words that can follow $w_{i-n+1}...w_{i-1}$ in the training data.

### 3.1.3 Back-off Smoothing

The above discounting methods of redistributing probability from observed events to unseen events. Additionally, if events are infrequently observed then they can be smoothed with more frequently observed events.

In any text corpora the number of observable bigrams and trigrams are limited. In this case, the probabilities of bigrams and trigrams that are seen few times in the training corpus are calculated by back-off method. This means that the trigram probability is replaced by a scaled bigram probability and the bigram probability is replaced by a scaled unigram probability,

$$P(w_i \mid w_{i-1}) = \alpha(w_{i-1})P(w_i)$$
$$P(w_i \mid w_{i-1}w_{i-2}) = \alpha(w_{i-1}, w_{i-2})P(w_i \mid w_{i-1})$$

where $\alpha$ is a back-off function that ensures the corresponding bigram and trigram probabilities is properly normalized.

Calculation of back-off bigram probability is as follows [3]:

Let the total number of occurrences of label *i* be $N(i) = \sum_{j=1}^{L} N(i, j)$ where *L* is the number of distinct labels and *N(i, j)* is the total number of occurrences of adjacent pair of $i$ and $j$ the bigram probability *P(i, j)* is given by

$$P(i,j)=\begin{cases}\alpha N(i,j)/N(i) & if \quad N(i)>0\\ 1/L & if \quad N(i)=0\\ f & otherwise\end{cases}$$

where $f$ is a floor probability and α is chosen to ensure that $\sum_{j=1}^{L} P(i,j)=1$.

For back-off bigrams, the unigram probabilities are given by

$$P(i)=\begin{cases}N(i)/N & if \quad N(i)>u\\ u/N & otherwise\end{cases}$$

where $u$ is unigram floor count and $N=\sum_{i=1}^{L}\max[N(i),u]$.

The backed-off bigram probabilities are given by

$$P(i,j)=\begin{cases}(N(i,j)-D)/N(i) & if \quad N(i,j)>t\\ \alpha(i)P(j) & otherwise\end{cases}$$

where $D$ is a discount and $t$ is a bigram count threshold.. *Discounting* is the process in which a small part of the available probability mass is deducted from the higher bigram counts and distributed amongst the infrequent bigrams. When a bigram count falls below the threshold *t,* the bigram is backed-off to the unigram probability suitably scaled by a back-off weight in order to ensure that $\sum_{j=1}^{L} P(i,j)=1$, i.e.

$$\alpha(i)=\frac{1-\sum_{j\varepsilon B}P(i,j)}{1-\sum_{j\varepsilon B}P(j)}$$

where $B$ is the set of all words for which $P(i, j)$ has a bigram.

In Figure 3.1 illustration of a back-off node is shown. Only observed bigrams are connected by direct word transitions with correspondent bigram probabilities. For back-off bigrams, the last state of previous word (i.e. $w_3$) is first connected to the back-off node with transition probability equal to back-off weight ($\alpha(w_3)$). The back-off node is then connected to the beginning of each word $w_m$ with transition probability equal to corresponding unigram probability $P(w_m)$.

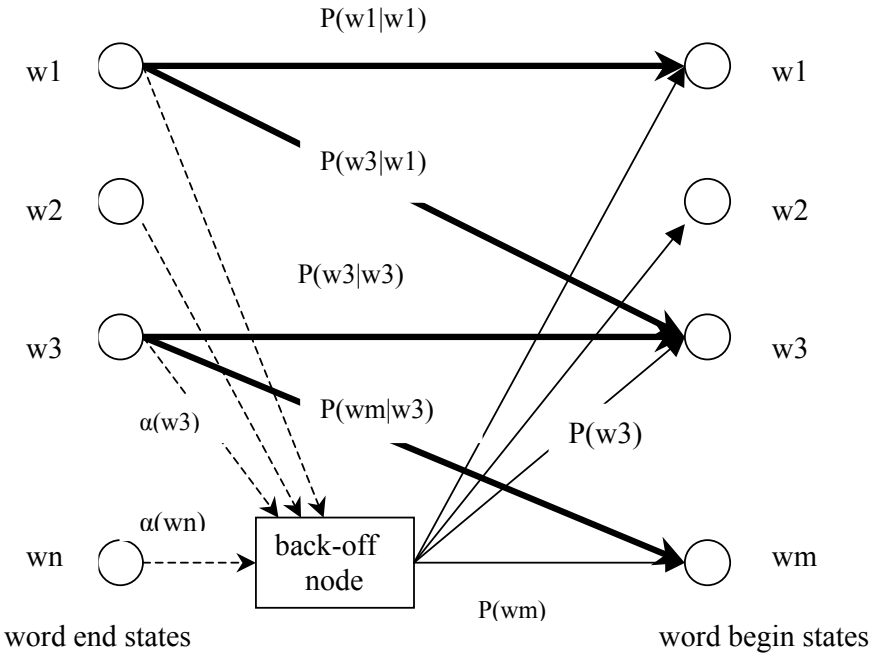



Figure 3.1: Illustration of back-off node.

## 3.2 Turkish Morphology

Turkish has an agglutinative morphology with productive inflectional and derivational suffixations. Since every productive suffix produces a new word, the number of words in Turkish is very high. There can be greater than 400.000 distinct words in a text corpus with 10 million words [11]. Since there are so many words in the language, it is unlikely that the lexicon for large vocabulary continuous speech recognition contains all the words. As a result of this, out of vocabulary words will be large. Some words that have very rare usage will not be appear in a very large text corpus. An example for this case is [12]:

OSMANLILAŞTIRAMAYABİLECEKLERİMİZDENMİŞSİNİZCESİNE

which can be broken down into morphemes as follows:

OSMAN+LI+LAŞ+TIR+AMA+YABİL+ECEK+LER+İMİZ+DEN+MİŞ+SİNİZ +CESİNE

On the other hand, Turkish has a free constituent order, however the order of constituents may be changed to emphasize certain constituents of the sentence. The word which will be emphasized is put before the predicate of the sentence. This is called as the *placement of constituents* rule. For example, all of the following sentences have the same meaning, but emphasized word in each, which is written in bold, is different:

- Murat yarın trenle **İzmir'e** gidecek. (Usual order)
- Murat yarın İzmir'e **trenle** gidecek. (Instrumental is emphasized)
- Murat trenle İzmir'e **yarın** gidecek. (Time is emphasized)
- Yarın trenle İzmir'e **Murat** gidecek. (Subject is emphasized)

Turkish morphology exhibits vowel harmony. Suffixation is subject to vowel harmony. The last vowel of the stem affects the first vowel of the suffix. Stem is the word without suffix. It is different than the root while stems may include suffixes. A stem ending with a back vowel (a, ı, o, u) takes a suffix starting with a back vowel, a stem ending with a front vowel (e, i, ö, ü) takes a suffix starting with a front vowel.

### 3.2.1 Language Modeling for Agglutinative Languages

Language modeling for an agglutinative language is different than language modeling for a language like English. An algorithm for an agglutinative language is given below [13]. In this algorithm, the roots and endings are considered as language model entries:

1. All possible endings are identified by means of a vocabulary.
2. The endings are extracted from all dictionary words. This can be done either by using a dictionary in which endings and stems are already defined or by processing the text to find endings and stems.
3. Using a sufficiently large text and the method in step 2, a text composed of stems and endings separated by white spaces is created.
4. Using the text obtained in step 3, the vocabulary that will be used for language modeling is constructed.

5. For each stem, a set of probability for the endings is calculated.
6. For any combination of stem and endings, an n-gram language model is generated.

# CHAPTER 4

# SEARCH

## 4.1 Introduction

The search (decoding) problem in large vocabulary speech recognition can be stated as finding the most likely word sequence given the acoustic models and the language model constraints, and the spoken utterance (or acoustic data). This is a demanding problem since word boundary information is not available in continuous speech, thus each of the words in the vocabulary may be hypothesized to start at each frame of acoustic data. Decoding strategy combines the scores obtained on the acoustic and language models and generates the possible word sequences (or hypotheses) from which we need to find the most likely for recognition. The task for the search algorithm is to evaluate the following equation,

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(W \mid O) = \underset{W}{\operatorname{argmax}} \frac{P(W)P(O \mid W)}{P(O)} = \underset{W}{\operatorname{argmax}} P(W)P(O \mid W) \qquad (4.1)$$

i.e., determine $\hat{W}$ given the various models and the acoustic data. Direct evaluation of all the possible word sequences is impossible (given the large vocabulary). An efficient search algorithm will consider only a very small subset of all possible sequence of words. Since word boundaries are not known at recognition time, search becomes more difficult. For example, if vocabulary has 50,000 words then a potential 50,000 words can start each frame.

## 4.2 Search Algorithms

As mentioned above, it is impossible to terminate the search in a practical amount of time if we consider all sequences of words during decoding. Therefore, we need to restrict our search space in some meaningful fashion. Some techniques for reconstructing the search space are given as follows [4]:

- *Optimized Hypothesis Generation* involves merging common partial hypotheses. A set of all partial sequences (hypotheses) is constructed in the form of a tree where common portions of the hypotheses are tied together. Refer to Figure 4.1 for an illustration.

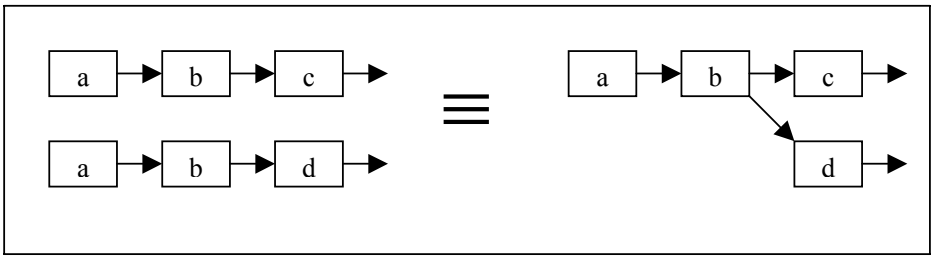



Figure 4.1: Optimized Hypothesis Generation

- *Problem-space Reduction* involves transforming the state space of the problem to make the search more efficient. This is done by merging common states in different hypotheses so that they need not be evaluated again and again. Figure 4.2 has an example.

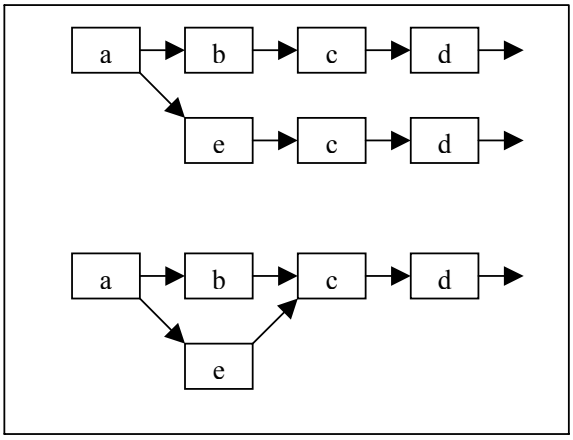



Figure 4.2: Problem space reduction by merging

- *Search Reduction* entails pruning away hypotheses that have partial evaluation scores less than some complete evaluation or some pre-determined threshold.
- *Knowledge Application* makes use of expert information to improve the efficiency of the search. A tight constraint in the knowledge base translates directly into smaller search space.

Decoder is forced to make sub-optimal chooses and this approximations does not affect the recognition error rate. Based on these approximations some algorithms are evolved: 1-)*Viterbi Decoding* [14,15], 2-)*Stack Decoding* [16,17]. **Viterbi decoding** is based on dynamic programming algorithm. This is a *time synchronous algorithm*, i.e., search progresses frame by frame, forward in time. **Stack decoding** is a *time*

*asynchronous algorithm*: the best scoring path or hypothesis, irrespective of time, is chosen for extension and this process is continued until a complete hypothesis is determined.

### 4.2.1 Viterbi decoding

Viterbi search and its variants belong to the breadth-first search techniques. Here, all hypotheses are pursued in parallel and gradually pruned away as the correct hypothesis emerges with the maximum score. It is based on the solution to Evaluation Problem for HMMs (discussed in chapter 2). Instead of taking summation over all possible state sequences to uncover the best word sequence, we can approximate the summation with the maximum to find the best state sequence. Therefore we can write equation (4.1) as:

$$\hat{W} = \arg\max_{W} P(W)P(O \mid W) \cong \arg\max_{W} \{P(W) \max_{s_1 \ldots s_N} P(O, s_1 \ldots s_N \mid W)\}$$

This equation is referred to as the *Viterbi approximation*. This means that the most likely word sequence is approximated by the most likely state sequence.

Viterbi search is a time-synchronous search algorithm. That means it completely processes time *t* before going on to time *t+1*. Using all states at time *t-1* each state is updated by the maximum score (instead of sum of all incoming paths) at time *t*. At the same time, it records the backtracking pointer to remember the best path. At the end of the search, these backtracking pointers are traced and the most probable state

sequence is found. We interested in optimal word sequence in speech recognition and not the optimal state sequence. Therefore, we keep the backtracking pointers to remember the word history of the current path. So when we reach at the end of the search, the optimal word sequence can be recovered. The benefit of keeping backtracking pointer is that we no longer need to keep the entire trellis. It is enough to keep the previous and the current time columns in the trellis computation.

For large vocabulary tasks, the complexity of the search space becomes larger and a *Viterbi beam search* is used to reduce the search space. In Viterbi beam search only the hypotheses whose likelihood falls within a fixed radius of the most likely hypothesis are considered. It exploits the observation that many states in the state lists have zero or near-zero scores and therefore need not be considered towards a solution. The best beam size can be determined empirically or adaptively. The advantage of the beam heuristics is that it allows the search to consider many good hypotheses in absence of a clearly dominant solution. Only a minor modification of the Viterbi algorithm results in the Beam search [17]. Figure 4.3 illustrates of single decoding process for Viterbi algorithm.

HTK uses *token passing* algorithm [18] to perform the search. A token represents a partial path through the network extending from time 0 through to time *t*. At time 0, a token is placed in every possible start node. Each time step, tokens are propagated along connecting transitions stopping whenever they reach an emitting HMM state. When there are multiple exits from a node, the token is copied so that all possible paths are explored in parallel. As the token passes across transitions and through nodes, its log probability is incremented by the corresponding transition and

emission probabilities. A network node can hold at most *N* tokens. Hence, at the end of each time step, *N* best tokens are survived. *N*=1 is sufficient for most purposes.

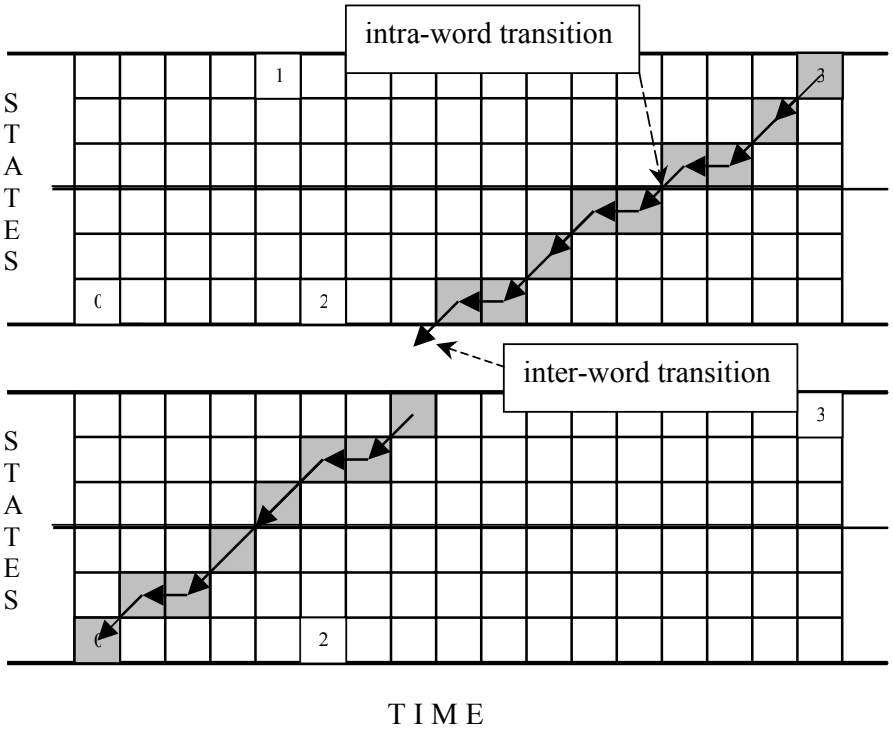



Figure 4.3: Viterbi decoding example for two words with two phonemes each: (0) possible start points of the sentence, (1) the first possible word end points, (2) first possible word transitions, (3) possible end points for the sentence. Best path obtained following bactracking pointer is shown in gray background.

### 4.2.2 Stack decoding

Since stack decoding is out of scope of this thesis, we will briefly discuss it. It is a depth-first technique in which the most promising hypothesis is pursued until the end of the speech data is reached. It is a heuristic search – the key is choosing the most probable hypothesis: the function used to choose this not only consider the

probability of the partial hypothesis or path, but also must estimate the probability of the hypothesis explaining the rest of the acoustic sequence.

An important advantage of the stack decoding algorithm is its consistency with the forward-backward training algorithm. The disadvantage is that an extra function is required for the comparison of the hypotheses of different lengths. The basic stack decoding algorithm [16,17] can be summarized as follows:

1. Pop the best partial hypothesis from the stack.
2. Apply acoustic and language model fast matches to shortlist the candidate next word.
3. Apply acoustic and language model detailed matches to candidate words.
4. Choose the most likely next word and update all hypothesis.
5. Insert surviving new hypotheses into the stack.

## 4.3 Multipass search

This class of search strategies have found widespread use in recent LVCSR systems. An approximate and efficient search with simple acoustic and language models is used to generate a lattice of alternatives or a subset of hypotheses that are more likely than others. In subsequent passes, more detailed decoding (by using more complex models) is used over this reduced search space to find correct hypothesis [19,20]. The first search pass produces either an N-best of possible word sequences or a word graph (or lattice) as its output. For example, an initial pass can be performed using word-internal context-dependent phones with a bigram language

model to generate a list of candidate hypotheses. Then, in a second pass of decoding, a trigram language modeling can be used with cross-word context dependent words.

The problem with this is that an error in the first pass can never be recovered, hence large lattices must be used. The advent of N-best search has been instrumental to the advancement of multipass search techniques.

### 4.3.1 N-best search

The optimal N-best decoding algorithm [21] is quite similar to the Viterbi search. Viterbi decoding is a simple case of N-best search. Viterbi exhibits a 1-best approach while N-best search finds all hypothesis sequences within the specified beam. It then allows only N top-scoring hypotheses to propagate to the next state. This state-dependent pruning is independent of the global Viterbi beam threshold.

The sources of information on speech used for recognition purposes can be extremely diverse and are correspondingly associated with different costs in terms of computation and memory requirements. A hypothesis that scores the highest given all these knowledge sources will be an optimal solution to the recognition problem. But this typically requires an impractically large search space. It is advantageous to use a strategy in which the most efficient knowledge sources are used first to generate a list of top N hypotheses. These hypotheses can later be re-evaluated with other, more expensive knowledge sources to arrive at the best hypothesis. N-best search provides an efficient method of integrating different knowledge sources and makes the search process more modular. The scores from different knowledge sources can be combined using weights chosen to minimize the recognition error.

For example, Table 4.1 shows an N-best list where N=5.

Table 4.1: An example of 5-best list.

| |
|---|
| 1. maçta olaylar yaşandı. |
| 2. çirkin olaylar yaşandı. |
| 3. bu anlar yaşandı. |
| 4. bu olaylar yaşanmazdı. |
| 5. bu görüntüler yaşandı. |

### 4.3.2 Word lattices and word graphs

The number of N-best hypotheses might grow exponentially with the length of the utterance. Thus, word lattices and word graphs are introduced to replace N-best list with a more compact representation of alternative hypotheses.

On the other hand, generating one-best results using detailed acoustic models and long span language models is a computationally expensive process and hence it takes lots of time. For every small changes made to the system, it is required to run the decoder from the beginning. We can speed up the decoding operation and see the effects of small changes in the system to the results by initially generating lattices. These lattices contain the most likely hypotheses for each test utterances instead of the single most likely sentence obtained in a single pass. Lattice generation is a fairly expensive process. But once it is produced, its rescoring takes lesser time.

Lattices can be used to constrain the search space (this is why system speeds up) for the second pass of decoding. They can be seen as a shrinking of search space. This scheme allows us to rescore lattices with new better (or computationally expensive) acoustic or language models without a full decoding. We applied only

larger (3-grams) language model to the lattices and then obtained one-best results. We used detailed acoustic models in the first pass, thus same acoustic models were used in the second pass.

#### 4.3.2.1 Lattice generation

As mentioned before HTK uses token passing algorithm for recognition. A token represents a partial path from time 0 to time t. In one best recognition case, the tokens are updated by adding language model likelihoods to them in word end instances. The tokens from equivalent partial paths are recombine and only the most likely token survives to propagate into the rest of the network. The less likely tokens are discarded. The calculations are performed only on the most likely one.

When a word lattice is needed then the less likely tokens are not discarded but are linked to the most likely one and the combined structure propagates into the following network. The calculations in the remainder of the network are performed on the most likely one but at the end of the utterance all of the tokens are used in construction of a lattice of alternative hypotheses [7]. An example lattice is shown in Figure 4.4.

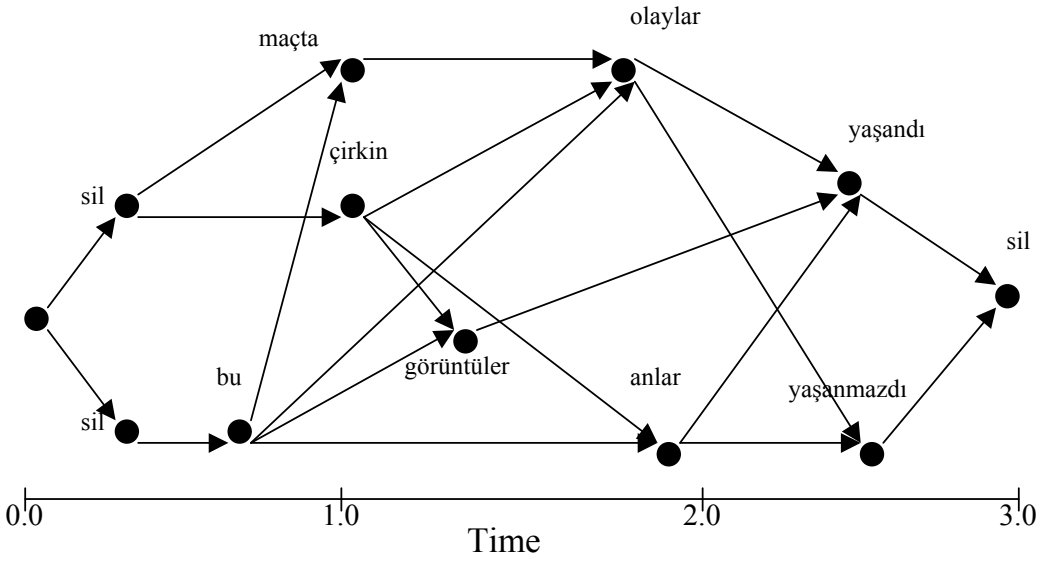



Figure 4.4: An example lattice.

#### 4.3.2.2 Lattice pruning

Lattices generated during decoding are very large and need to be shrunk since it also contains unlikely hypotheses. If we prune the lattices by discarding the paths that are unlikely, then the rescoring process becomes faster. Consequently, the size of the lattices can be reduced without altering the results obtained if it is rescored and reducing its accuracy.

When a lattice is pruned, the likelihood of the most likely path can be used to determine which parts of the lattice will survive. Every arc in the lattice has a log likelihood and these likelihoods compared with the log likelihood of the most likely hypothesis. Any arcs whose value is out of the predetermined beam width are removed from the lattice [7].

#### 4.3.2.3 Lattice expanding

New language models can be applied to lattices. The purpose of doing this is to allow higher-order-N-gram probabilities to be assigned to the word transitions so as to increase accuracy in the subsequent recognition pass. Thus, more context information is encoded in the lattice and rescoring the lattice can give better results.

To place trigram probabilities on the lattice generated with a bigram, each node must have the unique two word history [28]. For example, in Figure 4.5, a node labeled with **c** and its transitions **(c, d)** and **(c, e)** are duplicated to guarantee the uniqueness of the trigram context for placing **p(d|bc)** and **p(e|bc)** on the transitions (**c,d)** and **(c,e)**, respectively. When the node **c** has two predecessor nodes labeled with the same word, only one additional node and its corresponding outgoing transitions need to be duplicated as shown in Figure 4.6.

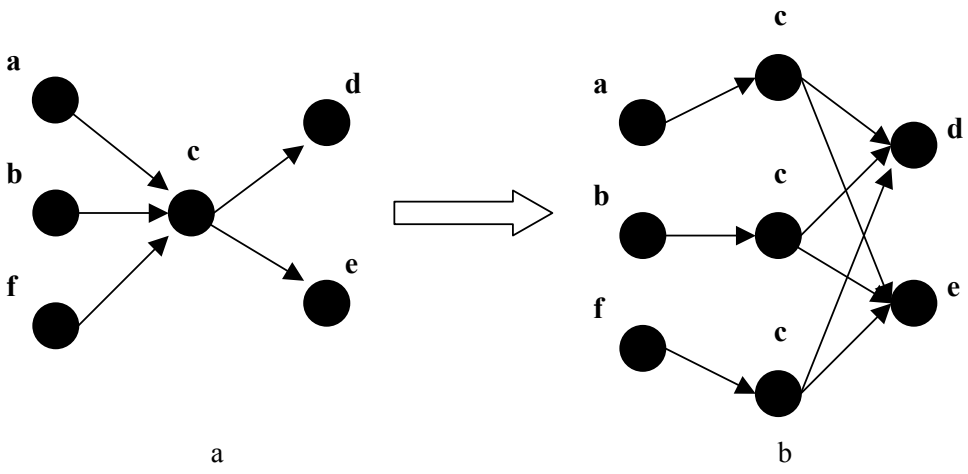
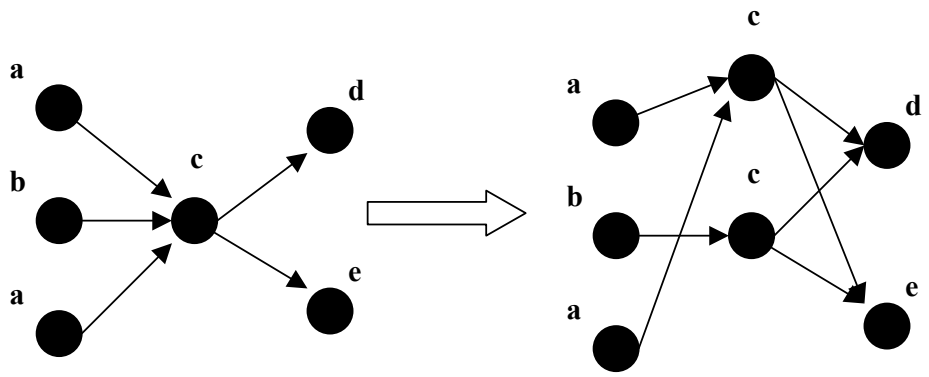



Figure 4.5: Trigram expansion case 1.a-) Bigram lattice before expansion b-) Trigram expansion




a b

Figure 4.6: Trigram expansion case 2. a-) Bigram lattice before expansion b-) Trigram expansion

## 4.4 Search space organization

If we use cross-word phonetic context instead of words in above word graph then we see that the size of the network will increase. This is because every word end needs to be hypothesized multiple times, once each for the possible next words. Especially, for large vocabulary size the search space explodes with the number of potential hypotheses.

We can reduce the size of search space by representing the lexicon as tree. The tree structured lexicon is called lexical tree or tree lexicon. It can save a lot of computation and storage. Since many words share common pronunciation prefixes, they can also share models and avoid duplication. A problem of lexical tree is that the word identity is unknown until a word-end lexical node is reached. When there is a transition from word 1 to word 2 , the word 2 is unknown until the end of the

lexical tree. Therefore the LM score can not be applied immediately upon the transition. That means pruning based on language models can not be applied as early as possible. An LM lookahead [22] technique is devised to overcome this problem.

Trees were initially used in fast match for producing candidate word lists for further search. Recently, they have been introduced in the main search component of several systems [23]. In a linear lexicon, each word is represented as a linear sequence of phonemes independent of other words. For example, though *sarı* and *sayı* share the same root, we do not share any of their history during the search process. Following figures are illustrates linear and tree lexicon respectively. Dark rectangles represent starts and ends of words.

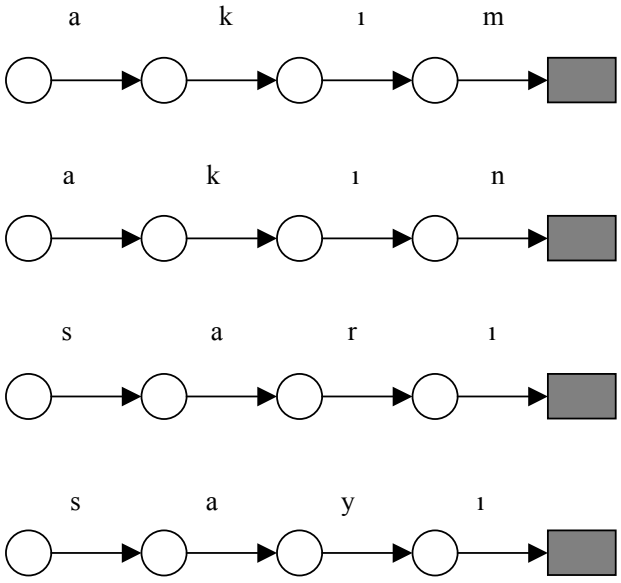



Figure 4.7: Linear lexicon example. It has 16 arcs.

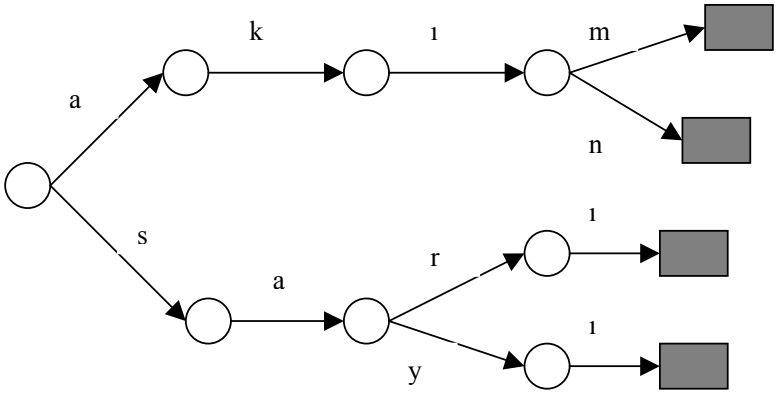



Figure 4.8: Tree lexicon example. It has 11 arcs.

## 4.5 Pruning

To save the computation and memory, it is imperative to stop less-likely partial paths from growing further. The idea of pruning is to only prune away those unlikely paths. However, it is possible that a path with a lower score in the early frame may grow to be the best path later. If the best path is pruned way, it will result in a recognition error.

There are many pruning techniques [24]. For decoding using cross-word context dependent models, multiple new tree hypotheses corresponding to different cross-word contexts are started for each active word-end sequence and at each time frame, causing a major increase in search complexity. In order to control the search space a standard pruning approach is applied during the search process. This pruning

approach consists of: *acoustic pruning*, *language model pruning (word-end pruning)* and *histogram pruning* [25], which are performed at every time frame.

- **Acoustic pruning:** In this method, only the hypotheses whose scores close to the best state hypothesis are considered for further calculations. If we denote acoustic pruning threshold as $f_{AC}$ and the score of the current best hypothesis as $Q_{AC}(t)$ then we prune the hypotheses whose scores are under the term $f_{AC}.Q_{AC}(t)$. The number of surviving state can be controlled by changing the value of $f_{AC}$.
- **Language model pruning:** This is only applied to tree startup hypotheses. For word end hypotheses, the bigram LM probability is incorporated into the accumulated score. Then, the best predecessor word is used to start up the corresponding tree hypothesis or is propagated into this tree hypothesis if it already exist.
- **Histogram pruning:** In this method, the number of state hypotheses that will be survive for further calculations is limited to a maximum number $M_{Sta}$. If the number of active states is larger than $M_{Sta}$, then only the best $M_{Sta}$ hypotheses are survived while the others are removed.

# CHAPTER 5

# EVALUATION RESULTS

The works which are performed in this thesis will be given in this chapter. Firstly, acoustic model training will be given. Then, proposed language model for Turkish and experimental results will be given, respectively.

## 5.1 Acoustic model training

The phones in the Turkish language are equivalent to the letters with 8 vowels and 21 consonants. If we take into account the acoustic variations of some phones because of the surrounding phones, we can represent the Turkish with 32 phones. Example of these additional phones are **kk** (like in word “kadar”), **kl** (like in word “lider”) and **kg** (like in word “gazete”). In addition of these phones, we have to use two silence models to represent the silences in the utterances. One of them is the silence model which occurs at the beginning and at the end of a sentence and we represent it by the symbol “smsm”. The other one is the short pause model which occurs between words in a sentence and we represent it by the symbol “spsp”.

Following figure is illustrates these silence models. The transitions from states 2 to 4 allows to absorb the various impulsive noises in the training data. Short pause model is also called *tee-model* which has a direct transition from entry to exit states.

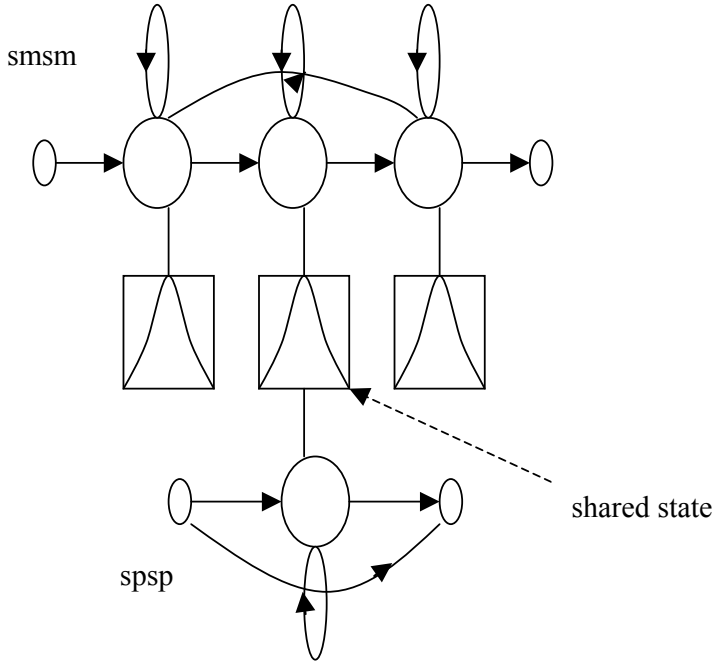



Figure 5.1: Silence models

**Data Preparation:**

The speech data which is used for acoustic modeling is recorded at Middle East Technical University. Database consists of **193** records that were spoken by **104** male and **89** female speakers. Since most of the speakers are university student, young voices dominate and the accent variation is very low. The number of sentences used in acoustic training is **7380**, the number of words is **52025** and the number of distinct words is **9165**.

It takes a long time to correct the sentences so that the text matces the records. For example, although the word is written as “kapayacak”, most people pronounce it like “kapıycak”. In this case we changed the text corpus to have the correct monophones for further processing. Word pronunciations are included in the dictionaries.

Some of our training data have been segmented by Özgül Salor [26] in order to initialize the monophone models. Since HTK uses units of 100 nanoseconds, the phone boundaries in segmented data were in terms of milliseconds.

**Training:**

Since we have used HTK in this thesis, it is useful to explain the training phases in terms of HTK tools.

First of all, the training data must be coded to use it in recognizer. In other words, front-end parameterization must be applied to the training data. This coding process can be performed using the tool **HCopy** configured to automatically convert its input into MFCC vectors. To do this, a configuration file is needed which specifies all of the conversion parameters. We have used configuration file in Table5.1. Feature vectors consist of zero mean power mel-cepstrum coefficients. The FFT uses a Hamming window and the signal should have first order preemphasis applied using a coefficient of 0.97. The frames of length 25 ms are taken every 10 ms from the speech signal.

Table 5.1: The content of the configuration file.

SOURCEKIND=WAVEFORM

SOURCEFORMAT=WAV

TARGETKIND=MFCC_E_D_A_Z

USEHAMMING=T

SAVEWITHCRC=T

SAVECOMPRESSED=T

PREEMCOEF=0.97

ENORMALISE=T

TARGETRATE=100000.0

WINDOWSIZE=250000.0

NUMCHANS=26

NUMCEPS=12

After defining a prototype HMM which consists of three states and a transition matrix, we can initialize all monophone models using segmented data and **HInit command**. **HInit** computes the parameters of a new HMM using a Viterbi style of estimation and performs this operation for every monophone one by one. After initializing all monophone models, **HRest is used** to refine the parameters. Its operation is very similar to **HInit** except that it expects the input HMM definition that have been initialised and uses Baum-Welch re-estimation in place of Viterbi training. Baum-Welch does soft decision that can be helpful when estimating phone-based HMMs since there are no hard boundaries between phones in real speech and using a soft decision may give better results.

Before using the tool **HERest** for embedded-unit training, in which all models are trained in parallel, training data has to be labelled at the phone level. This can be done by the tool **HLed**. A small part of training data with phone level transcription is shown in Figure 5.2. When there is limited training data and recognition in adverse noise environments is needed, so-called *fixed variance* models can offer improved robustness. These are models in which all the variances are set equal to the global speech variance and never subsequently re-estimated. The tool **HCompV** can be used to compute this global variance.

```
0 4900000 smsm
4900000 5700000 u
5700000 6800000 f
6800000 7800000 a
7800000 8100000 kk
8100000 8300000 spsp
8300000 9200000 kk
9200000 9500000 u
9500000 10100000 r
10100000 11400000 ts
11400000 11700000 u
11700000 12000000 n
```

Figure 5.2 : Example of phone level transcription. (Note ts = ş)

The HTK philosophy is to build systems incrementally. After each modification of the system (adding short pause model, parameter tying, decision tree clustring etc.) the main training tool **HERest** is performed several times. Unlike the processes described so far, embedded training, which is performed by **HERest** using Baum-Welch re-estimation, simultaneously updates all of the HMMs in a system using all of the training data. Before applying **HERest,** all monophone models must be collected in a file, since all models will be upgraded simultaneously.

At this stage, short pause model (see Figure 5.1) is added to the model by using the HTK HMM definition editor **HHed** and using the tool **HLed** , new transcription files with short pause model is created. After adding short pause model **HERest** is performed several times to make the model parameters more robust.

The phone models created so far can be used to align the training data. This can be done by single invocation of the HTK recognition tool **HVite**. Before going on state tying **HERest** is performed several times again.

**State tying:**

The final stage of model building is to create context-dependent triphone HMMs. This is done in two steps.

At the first step, the monophone transcriptions are converted to triphone transcriptions and a set of triphone models created by copying the monophones and re-estimating. The latter is done by label editor **HLed.** At the same time, a list of triphones used in training data is written to a file. Then, using the tool **HHed** model cloning is done. For example, for each model of the form "a-b+c" the parameters of monophone model "b" is copied and for all triphones like "*-b+*" same transition matrix is copied. After these modifications, we run **HERest** several times.

At second step, similar acoustic states of these triphones are tied to ensure that all state distributions can be robustly estimated. Decision tree state tying is performed by running **HHEd.** It is based on asking questions about the left and right contexts of each triphone. The questions asked in this work can be seen in Appendix B. At this stage all possible triphones in Turkish are considered except impossible

ones like “a-a+a”. Triphone statistics of training corpus can be seen in Table 5.2. After running **HERest** several times we go on mixture incrementing.

Table 5.2: Triphone counts.

| | |
|---|---|
| Total triphones in Turkish | 39304 |
| Usable triphones | 39104 |
| Triphones in acoustic data | 6687 |
| Triphones after decision tree clustering | 2316 |
| Triphones from text corpus | 9417 |

**Mixture incrementing:**

This is the final step of acoustic training. Tool for this is again **Hhed.** After incrementing mixtures by 2, we run **HERest** several times. This procedure goes on until incrementing the mixtures by 8. At last we run **HERest** several times and acoustic training is completed.

All of the process explained for acoustic model training can be seen as schematic representation in Appendix A. Monophone counts in acoustic training corpus can be seen in Appendix C.

## 5.2 Bigram language model training

Training text for bigram language model has been downloaded from internet. , The sentences which include foreign words have been deleted. However, some of these that were seen lots of times are kept in the corpus. For example, “Trapattoni” and “Juventus” were not deleted. We denoted the beginning of a sentence as “bbaassllaa” and the end of a sentence as “bbiittiirr”. These words are also included in the dictionary and they corresponds to the silence model at the beginning and at the end of the sentences.

**Parsing the words:**

Since Turkish is an agglutinative language, N-gram language modeling is not directly applied. We parsed the words into their stems and endings. The words to be parsed include derivational and inflectional suffixes. We parse a word from its first inflectional suffix if it has.

In the first experiment, we obtained bigram probabilities by treating stems and endings as individual items. This can be done by **HLStats.** As an example consider the sentence “ben yerli futbolcudan yanayım”; it is decomposed as

*Ben yerli futbolcu dan yana yım.*

Then, we have the bigram probabilities as P(yerli|Ben), P(futbolcu|yerli), P(dan|futbolcu), P(yana|dan) and P(yım|yana).

In the second experiment, we obtained bigram probabilities only over the stems. The underlying idea is that the actual informative part of the word is the stem. For the same example sentence;

*Ben yerli futbolcudan yanayım.*

we modify the sentence as being only stems;

*Ben yerli futbolcu yana.*

Finally, the bigram probabilities calculated over stems are assigned to word bigram probabilities;

P(yerli|Ben) = P(yerli|Ben)

P(futbolcudan|yerli) = P(futbolcu|yerli)

P(yanayım|futbolcudan) = P(yana|futbolcu).

Bigram text corpus statistics are shown in Table 5.3.

Table 5.3: Bigram text corpus statistics

| | Number of words | Vocabulary size |
|---|---|---|
| Unparsed text | 428,305 | 49,554 |
| Text with stem and endings | 648,695 | 15,191 |
| Text with only stems | 428,305 | 12,524 |

## 5.3 Trigram language model training

Training text for trigram language model has been downloaded from internet as in bigram model and again parsing rules are not different from the ones performed for bigram language model.

Construction of trigram language model is a three stage process:

1 – First of all, given language model training text, the tool **LGPrep** scans the input and generate a gram file holding the 3-grams seen in the training text along with their counts.

2 – The counts in the gram file are modified to vocabulary and class mapping. This step is optional and we have skipped this step.

3 – Finally, the 3-gram language model file is build by using the resulting gram file. This task is accomplished by **LBuild**. It reads the resulting gram file and generate a back-off 3-gram language model file. The LM can be built in steps (first a unigram, then a bigram and finally a trigram) or in a single pass if desired. LBuild supports Good-Turing and absolute discounting described in section 3.1.2.

Phases in creating trigram language model are shown in the figure in Appendix E. Following table shows statistics for trigram text corpus.

Table 5.4: Trigram text corpus statistics

| | Number of words | Number of distinct words |
|---|---|---|
| Unparsed text | 836,911 | 72,707 |
| Text with stems and endings | 1,268,186 | 19,733 |
| Text with only stems | 836,911 | 16,330 |

## 5.4 Test Results

The test utterances include **220** sentences that are arbitrarily selected from the text corpus that is used to extract bigram probabilities. They do not overlap with transcriptions of the speech data used in acoustic model training. These sentences were continuously spoken by 6 speakers (4 male, 2 female), who contributed in acoustic model training, too. These 220 utterances have a vocabulary of **1168** words.

### 5.4.1 Direct recognition results

**Experiment 1:** Since we parse the words different compared to [27] the results are also different. Before different parsing, we tested the system [27] with no silences at the beginning and at the end of the sentence. We cut out these silences from all of the test sentences. Results are as follows:

Table 5.5: Results with different s, p and u values for Experiment 1. Parsing in this test belongs to [27]. (CSRR: Correct Sentence Recognition Rate, CWRR: Correct Word Recognition Rate, Acc: Accuracy)

| **s** | **p** | **u** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|
| 20 | 15 | 3000 | 26.82% | 71.77% | 67.50% |
| 30 | 20 | 3000 | 27.73% | 69.51% | 66.71% |
| 30 | 20 | 8000 | 27.27% | 72.70% | 70.73% |

The variables **s** and **p** are *language model probability scaling factor* and *fixed penalty*, respectively. These are used to control word insertion and deletion levels. Every language model probability value is multiplied by **s** and **p** is added to the result. For example, if **s**=30 and **p**=-20, the language model log probability **x** becomes 30**x**-20. If **p** gets higher, more short words are inserted into the recognized sentence. This causes an increase in the insertion errors. The variable **u** represents *histogram pruning threshold*. This sets the maximum number of active models. Once the optimal alignment has been found, the number of substitution errors (S), deletion errors (D) and insertion errors (I) can be calculated. The correct sentence recognition rate (CSRR) and the correct word recognition rate (CWRR) are then defined as follows, respectively.

$$\text{CSRR} = \frac{H}{N} \times 100\%$$

$$\text{CWRR} = \frac{N - D - S}{N} \times 100\%$$

Where N is the total number of labels in the reference transcriptions. CWRR does not consider insertion errors, so accuracy is defined as an the alternative evaluation criterion as given below.

$$\text{Accuracy} = \frac{N - D - S - I}{N} \times 100\% .$$

It is more representative figure of recognizer performance.

After parsing, we tested the system both the records with no silences at the beginning and at the end and the vocabulary which is created corresponding to new parsed words. As mentioned before we have parsed the words from its first inflectional suffix. The results belonging the this test is given below:

Table 5.6: Results with different parsing for Experiment 1. Parsing in this test different from [27]. (CSRR: Correct Sentence Recognition Rate, CWRR: Correct Word Recognition Rate, Acc: Accuracy)

| | **s** | **p** | **u** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|---|
| Back-off not changed, with silences | 30 | 20 | 8000 | 15.91% | 60.93% | 52.86% |
| Back-off not changed, with no silences | 30 | 20 | 8000 | 29.09% | 71.61% | 68.68% |
| Back-off changed, with silences | 30 | 20 | 8000 | 17.73% | 61.62% | 51.12% |
| Back-off changed, with no silences | 30 | 20 | 8000 | 30.00% | 72.99% | 69.30% |

It is clear to see from the table no silences in the test record gives the best result. If we compare this results with the ones in [27] we see that CSRR is increased with 10%.

**Experiment 2:** In this experiment, we estimate bigram probabilities only over stems. Firstly, we tested the system in [27] with no silences at the beginning and at the end of test records. The results for this case are shown below:

Table 5.7: Results with different s, p and u values for Experiment 2. All results are with changed back-off and with no silences. Parsing in this test belongs to [27]. (CSRR: Correct Sentence Recognition Rate, CWRR: Correct Word Recognition Rate, Acc: Accuracy)

| s | p | u | CSRR | CWRR | Acc |
|---|---|---|---|---|---|
| 20 | 15 | 3000 | 40.91% | 71.07% | 56.72% |
| 30 | 20 | 4000 | 40.91% | 70.43% | 63.52% |
| 30 | 20 | 5000 | 43.64% | 74.58% | 68.38% |
| 30 | 20 | 6000 | 46.36% | 77.09% | 71.13% |
| 30 | 20 | 7000 | 51.36% | 79.98% | 74.91% |
| 30 | 25 | 7000 | 51.36% | 80.03% | 74.64% |
| 35 | 20 | 7000 | 44.09% | 74.74% | 71.02% |
| 40 | 20 | 7000 | 42.27% | 72.10% | 69.19% |
| 30 | 25 | 8000 | 53.64% | 81.33% | 76.69% |
| 30 | 25 | 9000 | 54.09% | 81.71% | 77.06% |
| 30 | 25 | 10000 | 55.91% | 83.05% | 78.68% |
| 30 | 25 | 11000 | 55.91% | 83.43% | 79.06% |
| **30** | **25** | **12000** | **56.82%** | **83.97%** | **79.87%** |

Then, we tested the system with new parsing. Here are the results:

Table 5.8: Results with different parsing for Experiment 2. Parsing in this test different from [27]. (CSRR: Correct Sentence Recognition Rate, CWRR: Correct Word Recognition Rate, Acc: Accuracy)

| | s | p | u | CSRR | CWRR | Acc |
|---|---|---|---|---|---|---|
| Back-off not changed, with silences | 30 | 20 | 8000 | 16.82% | 48.02% | 42.04% |
| Back-off not changed, with no silences | 30 | 20 | 8000 | 32.27% | 56.71% | 54.64% |
| Back-off changed, with silences | 30 | 20 | 8000 | 35.00% | 72.95% | 55.46% |
| Back-off changed, with no silences | 30 | 20 | 8000 | 63.64% | 84.52% | 77.02% |
| Back-off changed, with no silences | 30 | 25 | 12000 | 64.09% | 85.12% | 77.29% |
| Back-off changed, with no silences | 30 | 25 | 15000 | 64.09% | 85.12% | 77.29% |

### 5.4.2 Two pass recognition results

In this stage of our study, we obtained recognition results through lattices by using two pass instead of getting them in one pass. Figure in the Appendix F shows speech recognizer performing two pass. In the first pass, we used a bigram language model in recognition. Clearly, to get more accurate speech recognition, we have to use more powerful language model such as a trigram language model. In addition, we need some techniques to allow such models to be used to improve the speech recognition.

Our approach is not to modify the speech recognition at all. We use a bigram language model and **HVite** and then output a series of good hypothesis. This is called an N-best output (detailed explanation about N-best can be found in section 4.3). At the same time, we obtain lattices (from which the N-best lists are generated) belonging to each test utterances, instead of getting one best hypothesis.

For example, if we consider our first test utterance: “ben yerli futbolcuxdan yanaxyIm”, an N-best output with bigram model for this example is

*ben yerli futbolcuxdan yanaxyIm*
///
*ama ben yerli futbolcuxdan yanaxyIm*
///
*daha ilk yarIxsI golsUz sona erxdi*
///
*hele ilk yarIxsI golsUz sona erxdi*
///
*rumen yIldIz futbolcuxdan yanaxyIm*

Figure 5.3: An N-best output with bigram model

The lattice from which this N-best are generated is as follows:

```
N=61   L=104
I=0    t=0.00  W=!NULL
I=1    t=0.01  W=B                  v=1
I=2    t=0.01  W=B                  v=1
…
I=57   t=1.92  W=erxdi              v=1
I=58   t=1.92  W=yanaxyIm           v=1
I=59   t=1.93  W=!NULL
I=60   t=1.93  W=S                  v=1
J=0    S=0   E=1   a=-640.72        l=0.000
J=1    S=0   E=2   a=-640.72        l=0.000
J=2    S=0   E=3   a=-640.72        l=0.000
…
J=101  S=60  E=59  a=0.00           l=0.000
J=102  S=57  E=60  a=-111.28        l=-0.200
J=103  S=58  E=60  a=-111.28        l=-2.280
```

Figure 5.4: Word Lattice example (N: Total number of nodes, L: Total number of links, I : Node number, J : Link number, S: Start word, E: End word)

Each node in the lattice represents a point in time measured in seconds and each arc represents a word spanning the segment of the input starting at the time of its start node and ending at the time of its end node. For each such span, *v* gives the number of the pronunciation used, *a* gives the acoustic score and *l* gives the language model score. For example, consider the following links in the lattice

**J=0 S=0 E=1 *a*=-640.72 *l*=0.000 ,**

**J=53 S=1 E=35 *a*=-3356.22 *l*=-4.300,**

**J=66 S=35 E=41 *a*=-3015.74 *l*=-6.260,**

**J=93 S=41 E=55 *a*=-7346.56 *l*=-1.550,**

**J=100 S=55 E=58 *a*=-3340.81 *l*=-6.460,**

**J=101 S=60 E=59 *a*=0.00 *l*=0.000,**

**J=103 S=58 E=60 *a*=-111.28 *l*=-2.280.**

If the corresponding words are written instead of the node numbers then the structure of the lattice can be more understandable. Figure 5.5 shows the hypothesis that the above links represents. The numbers in the parentheses near words represents the node number belonging to that word (and the numbers seen on the arcs represents the arc number belonging to that arc.)

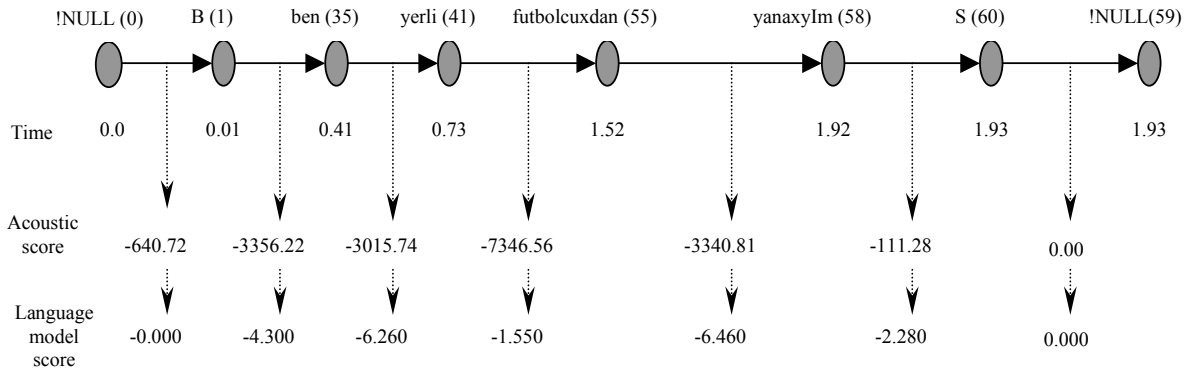



Figure 5.5: One of the hypothesis exists in word lattice.

Sum of the acoustic and language model scores from the beginning to the end of the hypothesis gives the overall likelihood of the that hypothesis. This total scores exist for every hypotheses in the lattice and they are used in making decision which hypotheses are less likely.

We process the lattice in Figure 5.4 (not N-best list) in order to obtain best hypothesis. Because, in N-best list, there are limited number of hypotheses. In spite of this, lattice encodes vastly more alternatives that realistically captured in N-best list.

We divide the recognition into two steps:

1-) Firstly, we do forward search through the observation sequence using a bigram language model and recognition tool **HVite** to produce lattices of hypotheses.

2-) Then, following processes are applied to these lattices by using **HLRescore**, which is lattice post processing tool. This process is shown in Appendix D.

- **Lattice Pruning :** Bigram lattices generated during recognition quite large and therefore difficult to expand. So, we obtain smaller bigram lattices by pruning.
- **Expansion of Lattices :** We apply trigram probabilities to pruned lattices. The purpose of trigram lattice expansion is to allow higher order N-gram probabilities to be assigned to the word transitions so as to increase accuracy in subsequent recognition pass. This is the core of the second pass recognition.
- **Finding 1-best Transcription in the Expanded Lattice :** Finally, the transcription that gives best score is selected through expanded lattices.

After pruning the lattice in Figure 5.4 takes form as shown in Figure 5.6 and this form can be changed according to pruning parameters. These parameters are

1) grammar scale factor (s). This factor post-multiplies the language model likelihoods from the word lattices,

2) the word insertion log probability (p),

3) lattice pruning parameter (u).

These parameters have the same function with **s, p** and **u** in direct recognition performed by **HVite**. In pruning, the effect of the parameter 'u' on lattice size is greater than those of 's' and 'p'. If 'u' increases, then lattice size increases. After pruning, we have smaller lattices in which the hypotheses that have less likelihood are eliminated.

```
N=18  L=20
I=0   t=0.00  W=!NULL
I=1   t=0.01  W=B                    v=1
I=2   t=0.01  W=B                    v=1
I=3   t=0.01  W=B                    v=1
I=4   t=0.15  W=ama                  v=1
I=5   t=0.18  W=daha                 v=1
I=6   t=0.40  W=ilk                  v=1
I=7   t=0.41  W=ben                  v=1
I=8   t=0.73  W=yerli                v=1
I=9   t=0.96  W=yarIxsI              v=1
I=10  t=1.25  W=golsUz               v=1
I=11  t=1.52  W=kapanxdI             v=1
I=12  t=1.52  W=futbolcuxdan         v=1
I=13  t=1.87  W=yanaxyIm             v=1
I=14  t=1.92  W=erxdi                v=1
I=15  t=1.92  W=yanaxyIm             v=1
I=16  t=1.93  W=!NULL
I=17  t=1.93  W=S                    v=1
J=0    S=0    E=1    a=-640.72        l=0.000    r=0.00
J=1    S=0    E=2    a=-640.72        l=0.000    r=0.00
J=2    S=0    E=3    a=-640.72        l=0.000    r=0.00
J=3    S=2    E=4    a=-1349.89       l=-3.980   r=0.00
J=4    S=3    E=5    a=-1507.77       l=-5.430   r=0.00
J=5    S=5    E=6    a=-1774.42       l=-5.520   r=0.00
J=6    S=1    E=7    a=-3356.22       l=-4.300   r=0.00
J=7    S=4    E=7    a=-2111.83       l=-3.430   r=0.00
J=8    S=7    E=8    a=-3015.74       l=-6.260   r=0.00
J=9    S=6    E=9    a=-5137.92       l=-1.360   r=0.00
J=10   S=9    E=10   a=-2917.06       l=-3.190   r=0.00
J=11   S=10   E=11   a=-2437.06       l=-2.840   r=0.00
J=12   S=8    E=12   a=-7346.56       l=-1.550   r=0.00
J=13   S=12   E=13   a=-2868.34       l=-6.460   r=0.00
J=14   S=13   E=14   a=-527.13        l=-30.310  r=0.00
J=15   S=11   E=15   a=-3340.81       l=-30.450  r=0.00
J=16   S=12   E=15   a=-3340.81       l=-6.460   r=0.00
J=17   S=17   E=16   a=0.00           l=0.000    r=0.00
J=18   S=14   E=17   a=-111.28        l=-0.200   r=0.00
J=19   S=15   E=17   a=-111.28        l=-2.280   r=0.00
```

Figure 5.6: Pruned Word Lattice.

Figure 5.7 shows the graphical representation of the lattice in Figure 5.6.

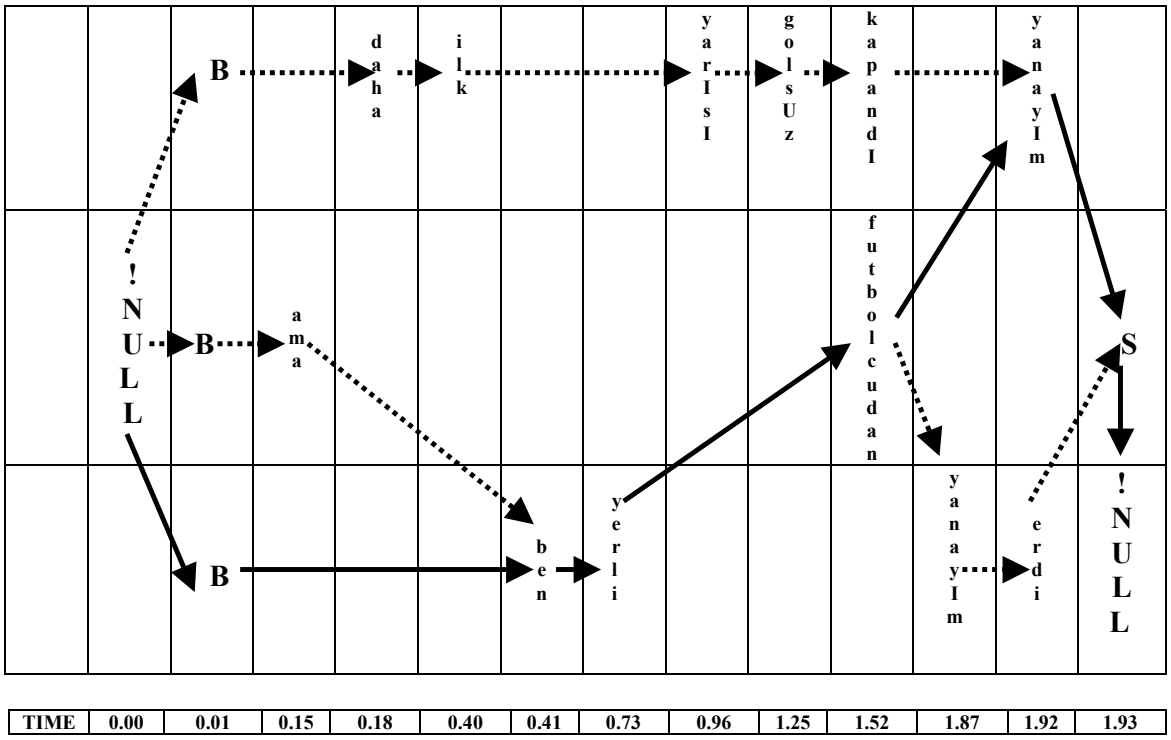



Figure 5.7: Word graph example. Shaded path is the most probable path in second pass.

**Experiment 3:** In this experiment, two pass recognition was applied on the utterances in which each word has parsed into stem and endings and each ending in the utterance was considered as a word (same in experiment 1 in one-pass recognition). Two trigram language model files were used in the this experiment. The first one is trigram file obtained directly from training text by using HTK. The second one is the file in which trigram probabilities were changed in the following manner:

There are 5 trigram structures that we can encounter:

1-) *stem1 stem2 ending* $\Rightarrow$ *P(ending|stem1 stem2) = P(ending|stem2),*

2-) *stem1 stem2 stem3* $\Rightarrow$ *P(stem3|stem1 stem2) = P(stem3|stem2),*

3-) *stem1 ending stem2* $\Rightarrow$ *P(stem2|stem1 ending) = P(stem2|stem1),*

4-) *ending stem1 stem2* $\Rightarrow$ *P(stem2|ending stem1) = P(stem2|stem1),*

5-)*ending1 stem1 ending2* $\Rightarrow$ *P(ending2|ending1 stem1) = P(ending2|stem1).*

In the first pass, lattices were obtained by following parameter set:

The grammar scale factor (s) = 30,

The word insertion log probability (p) = 25,

The maximum number of active models (u) = 8000.

Both in pruning and expanding in the second pass, **s** and **p** parameters are chosen as **20.0** and **5.0**, respectively.

Table 1 and Table 2 show the one-best recognition results from lattices generated by using 2 and 5 tokens, respectively.

Table 5.9: One-best recognition results for experiment 3 from lattices generated by using 2 tokens.

| | **u (in pruning)** | **u (in expanding)** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|
| Original Trigram LM | 130 | off | 21.82% | 72.01% | 58.83% |
| Original Trigram LM | 130 | 0.0 | 44.55% | 80.01% | 75.71% |
| Changed Probability Trigram LM | 130 | off | 20.45% | 71.51% | 58.04% |
| Changed Probability Trigram LM | 130 | 0.0 | 38.18% | 76.79% | 72.63% |

Table 5.10: One-best recognition results for experiment 3 from lattices generated by using 5 tokens.

| | **u (in pruning)** | **u (in expanding)** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|
| Original Trigram LM | 130 | off | 15.00% | 67.81% | 47.10% |
| Original Trigram LM | 130 | 0.0 | 56.82% | 84.54% | 80.52% |
| Changed Probability Trigram LM | 130 | off | 14.09% | 67.38% | 46.23% |
| Changed Probability Trigram LM | 130 | 0.0 | 40.91% | 78.78% | 74.58% |

**Experiment 4:** In this experiment, we estimate trigram probabilities only over stems. Again, we have used two trigram file in expanding lattices. The first one is the original file obtained from HTK and in the second file, we have changed trigram probabilities in the original LM file. For example, the probability of the trigram "*stem1+ending1 stem2+ending2 stem3+ending3" (*some words can not have any ending*)* changed as the probability of the trigram "*stem1 stem2 stem3"*.

Table 5.11: One-best recognition results for experiment 4 from lattices generated by using 2 tokens.

| | **u (in pruning)** | **u (in expanding)** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|
| Original Trigram LM | 130 | off | 56.36% | 85.77% | 71.81% |
| Original Trigram LM | 130 | 0.0 | 72.27% | 87.51% | 79.90% |
| Changed Probability Trigram LM | 130 | off | 56.36% | 85.71% | 71.75% |
| Changed Probability Trigram LM | 130 | 0.0 | 71.36% | 87.56% | 79.25% |

Table 5.12: The one-best recognition results for experiment 4 from lattices generated by using 5 tokens.

| | **u (in pruning)** | **u (in expanding)** | **CSRR** | **CWRR** | **Acc** |
|---|---|---|---|---|---|
| Original Trigram LM | 130 | off | 53.64% | 85.66% | 65.83% |
| Original Trigram LM | 130 | 0.0 | 75.91% | 88.86% | 81.21% |
| Changed Probability Trigram LM | 130 | off | 52.73% | 85.50% | 65.51% |
| Changed Probability Trigram LM | 130 | 0.0 | 74.55% | 88.32% | 79.85% |

# CHAPTER 6

# CONCLUSION

In this thesis, we have made four experiments. All experiments are related to Turkish Continuous Speech Recognition. In the first two experiments, results were obtained by a single pass approach. In the last two experiments, first, word lattices was generated from recognition network for every test utterance, then recognition results were obtained through these lattices in the second pass.

Two methods have been used for Turkish language modeling. In the first method, stems and endings were considered as separate words and language model probabilities were calculated by using stems and endings. This method was used in Experiment 1 and Experiment 3. In the second method, language model probabilities were calculated by using only stems. This method was used in Experiment 2 and Experiment 4.

We have used different parsing strategy from the one in [27]. In [27], parsing was done according to frequency of stems. But we parsed a word from its first inflectional suffix if it has. For example, the word “futbolcudan” was parsed as “futbol+cudan” in [27], but we parsed it as “futbolcu+dan”. Furthermore, we also

parsed the words that include one-letter morphemes. In other words, the accusative, dative and genitive forms of the stems are not left. For example, the word “oyuncum” was parsed as “oyuncu+m”. From the results, we saw that this does not cause acoustic confusion during recognition.

In the first experiment, we have used cross-word expanded network which is based on bigrams that include stems and endings. In this experiment we tested for the affect of the removal of silences at the beginning and at the end of the sentences and also for the new parsing of. An increase in CSRR by 7 % has been observed as a result of the removal of initial and final silences. Then new parsing further increased CSRR from 20 .09 % to 30.00 %.

In the second experiment, cross-word expanded network was used, which is based on bigrams that include only stems. Again, we first tested with old parsing (in [27]) and observed an increase in CSRR from 30.90 % to 51.36 %. Then, new parsing was tested; We observed an increase in CSRR from 30.90 % to 63 %.

In the third experiment, we have obtained trigram probabilities from a larger context again by using stems and endings. This time we have used the bigram probabilities in Experiment 1 to produce lattices for each test utterance in the first pass. Two cases were considered in the second pass; First , lattices generated by using 2 tokens and original trigram file were used to pick out the one-best hypothesis. This increased the CSRR from 30.00% (single pass result) to 44.55% (two pass result). Then lattices generated by 5 tokens and original trigram file were used to produce one-best results. This gave better results compared to 2-token-lattices. Changing probabilities in the trigram file (procedure was defined in

Experiment 3) did not improve the recognition results. On the contrary, it decreased the CSRR.

In the fourth experiment, we have used bigram probabilities in the second experiment to produce lattices belonging to test utterances. Trigram probabilities were obtained by using unparsed words. First of all, lattices generated by using 2 tokens and trigram file obtained from unparsed words were used to pick out the one-best hypothesis through lattices; this has increased the CSRR from 64.09% (single pass result) to 72.27% (two pass result). Then lattices generated by 5 tokens and trigram file obtained from unparsed words were used to produce one-best results. This gave better results again compared to 2-token-lattices. When the probabilities in the trigram file changed according to the procedure defined in Experiment 4 results were not improved.

We can infer from these results that our silence model parameters were not estimated robustly. Because test sentences with no silences increases the recognition rate considerably. That the environment was not purely silent may be another reason for these results. Sytem can perceive the noise as speech and this may cause recognition errors.

Lattice generation, at the end of the first pass, is time consuming. However once a lattice is created, recognition is very fast. Another advantage of generating lattices first and then using them in recogntion is that it sets an efficient way of experimentally determining optimum values for language model scale and penalty factors. Instead of having to rerun the decoder many times with different parameter settings the decoder is run once to generate lattices. Token number affects the size of generated lattices. If 5 tokens are used to create lattices, then generation time is

longer and the size of the lattices are larger compared to the lattices with 2-token case. But higher CSRR in 5-tokens case is its advantage.

As a conclusion, new language modeling techniques have been applied to Turkish. Experiments in which stems are used for language model probability computation gave better results as expected. Because stem holds the meaning of a word. For a future work, larger training text (including 20-30 millions words) will be used for language model training. Since HTK restricts us about using variability of an agglutinative language, we think of implementing a new decoder performing speech recognition.

# REFERENCES

[1] K. A. J. Riederer. Large Vocabulary Continuous Speech Recognition. Laboratory of Computational Engineering Helsinki University of Technology. 1999.

[2] S.Young. Large Vocabulary Continuous Speech Recognition: a Review. Cambridge University Engineering Department. 1996.

[3] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, P. Woodland. The HTK Book (for HTK Version 3.1). Cambridge University Engineering Department. 2002.

[4] N. Deshmukh. Efficient Search Algorithms for Large Vocabulary Continuous Speech Recognition. Institute for Signal and Information Processing (ISIP) Department of Electrical and Computer Engineering Mississippi State University. 1996.

[5] X. Huang, R. Reddy. Spoken Language Processing. Pearson Education. 2001.

[6] K. F. Lee. Large Vocabulary Speaker Independent Continuous Speech Recognition. Ph.D. Dissertation, Computer Science Department, Carnegie Mellon University. 1988.

[7] J. J. Odell. The Use of Context in Large Vocabulary Speech Recognition. Ph.D. Dissertation, Cambridge University. 1995.

[8] M. Woszczyna. Fast Speaker Independent Large Vocabulary Continuous Speech Recognition. Ph.D. Thesis. Karlsruhe University. 1998.

[9] S. Young. The General use of Tying in Phoneme based HMM Speech Recognizers. Proceedings ICASSP. pp. 569-572. 1992.

[10] S. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. IEEE Transactions on Acoustics, Speech and Signal Processing, vol.35, no.3, pp.400-01, March 1987.

[11] D. Hakkani-Tür, K. Oflazer, G. Tür. Statistical Morphological Disambiguation for Agglutinative Languages. Technical Report, Bilkent University. 2000.

[12] K. Oflazer. Two-level Description of Turkish Morphology. Literary and Linguistic Computing, vol.9, no.2. 1994.

[13] E. Mengüşoğlu, O. Derro. Turkish LVCSR: Database Preparation and Language Modeling for an Agglutinative Language. ICASSP'2001, Student Forum, Salt-Lake City. May 2001.

[14] R. Haeb-Umbach, H. Ney. Improvements in Time-Synchronous Beam Search for 10000-word Continuous Speech Recognition. IEEE Trans Speech and Audio Processing vol.2, pp. 353-356. 1994.

[15] A. Ganapathiraju. Implementation of Viterbi Search Algorithm. Institute for Signal and Information Processing, Department of Electrical and Computer Engineering, Mississippi State University. 1995.

[16] L. R. Rabiner, J. G. Wilpon, F. K. Soong. High Performance Digit Recognition Using Hidden Markov Models. IEEE Transactions on Acoustics Speech and Signal Processing, vol.37, no.8, pp.1214-1225. August 1989.

[17] D. B. Paul. An Efficient $A^*$ Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model. ICASSP, vol.1, pp.25-28. 1992.

[18] S. J. Young, N. H. Russell, J. H. S. Thornton. Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Technical Report. Cambridge University Engineering Department. 1989.

[19] S. Austin, R. Schwartz. The Forward-Backward Search Algorithm for Continuous Speech Recognition. Proc ICASSP, S.10.3. 1991.

[20] F. Richardson, M. Ostendorf, JR. Rohlicek. Lattice-Based Search Strategies for Large Vocabulary Recognition. Proc ICASSP, vol.1. pp.576-579. 1995.

[21] H. Purnhagen. N-Best Search Methods Applied to Speech Recognition. Diploma Thesis. Universitet Trondheim. 1994.

[22] S. Ortmanns, H. Ney and A. Eiden, "Language Model Look-ahead for Large Vocabulary Speech Recognition", Proceedings of the Fourth International Conference on Spoken Language Processing, pp. 2095-2098, October 1996.

[23] G. Antoniol, F. Brugnara, M. Cettolo, M. Federico. Language Model Representation for Beam Search Decoding. IEEE, ICASSP, pp.588-591. May 1995.

[24] N. Deshmukh, A. Ganapathiraju, J. Picone. Hierarchical Search for Large Vocabulary Conversational Speech Recognition. IEEE Signal Processing Magazine, pp.84-107. September 1999.

[25] H. Ney, S. Ortmanns. Progress in Dynamic Programming Search for LVCSR. Proceedings of the IEEE, vol.88, no.8. pp.1224-1239. August 2000.

[26] Ö. Salor, B. Pellom, T. Çiloğlu, K. Hacıoğlu, M. Demirekler. On Developing New Text and Audio Corpora and Speech Recognition Tools for the Turkish Language, ICSLP 2002, Denver Colorado.

[27] M. A. Çömez. Large Vocabulary Continuous Speech Recognition for Turkish Using HTK. M. Sc. Thesis. Middle East Technical University. 2003.

[28] F. Weng, A. Stolcke, A. Sankar. Efficient Lattice Representation and Generation. Speech Technology and Research Laboratory SRI International Menlo Park, California.

[29] F. Peng. Language Independent Text Learning with Statisticaln-Gram Language models. Thesis for the degree of Doctor of Philosophy in Computer Science. Waterloo, Ontario, Canada. 2003.

# APPENDIX A

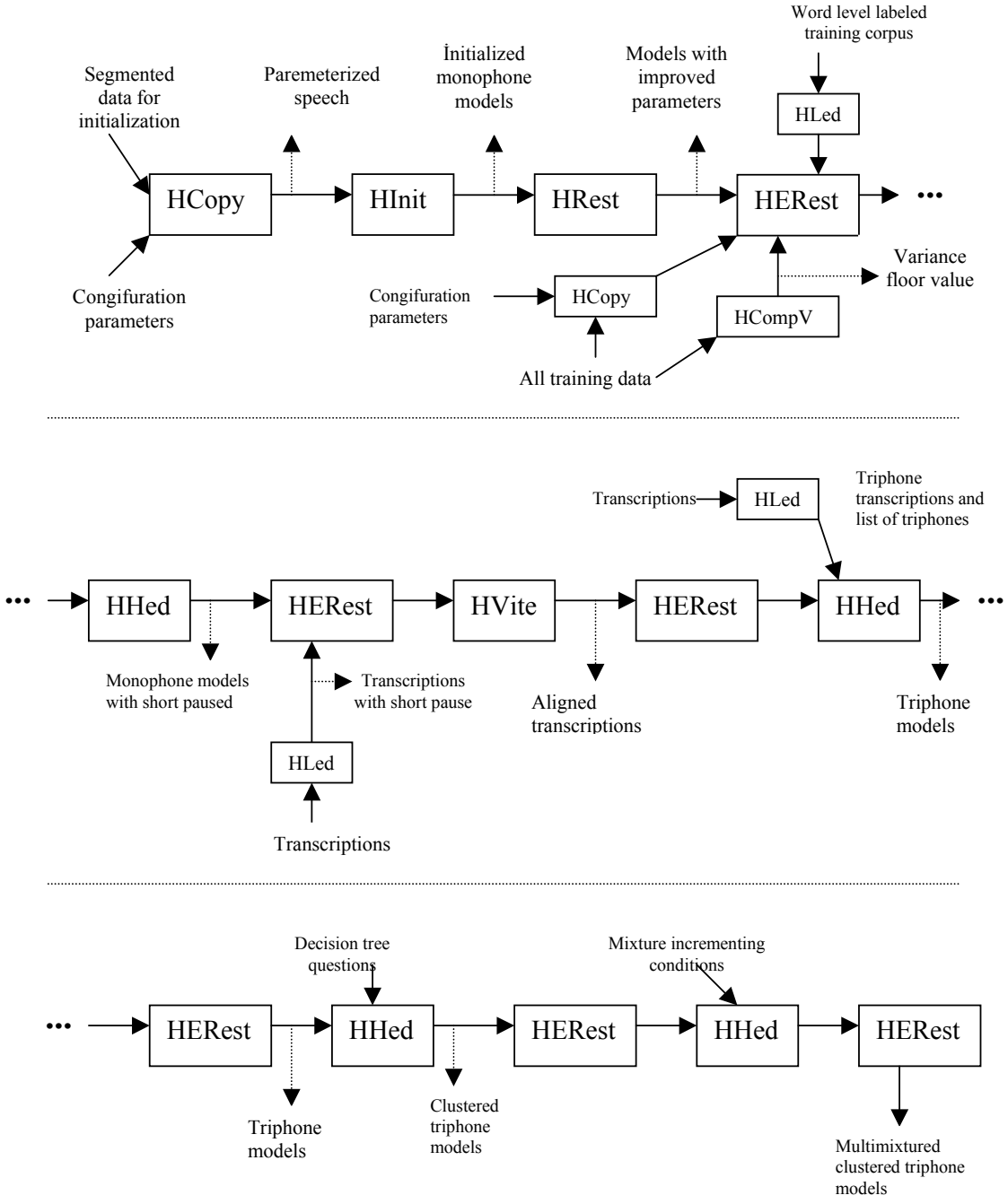



Figure A.1: Acoustic training phases.

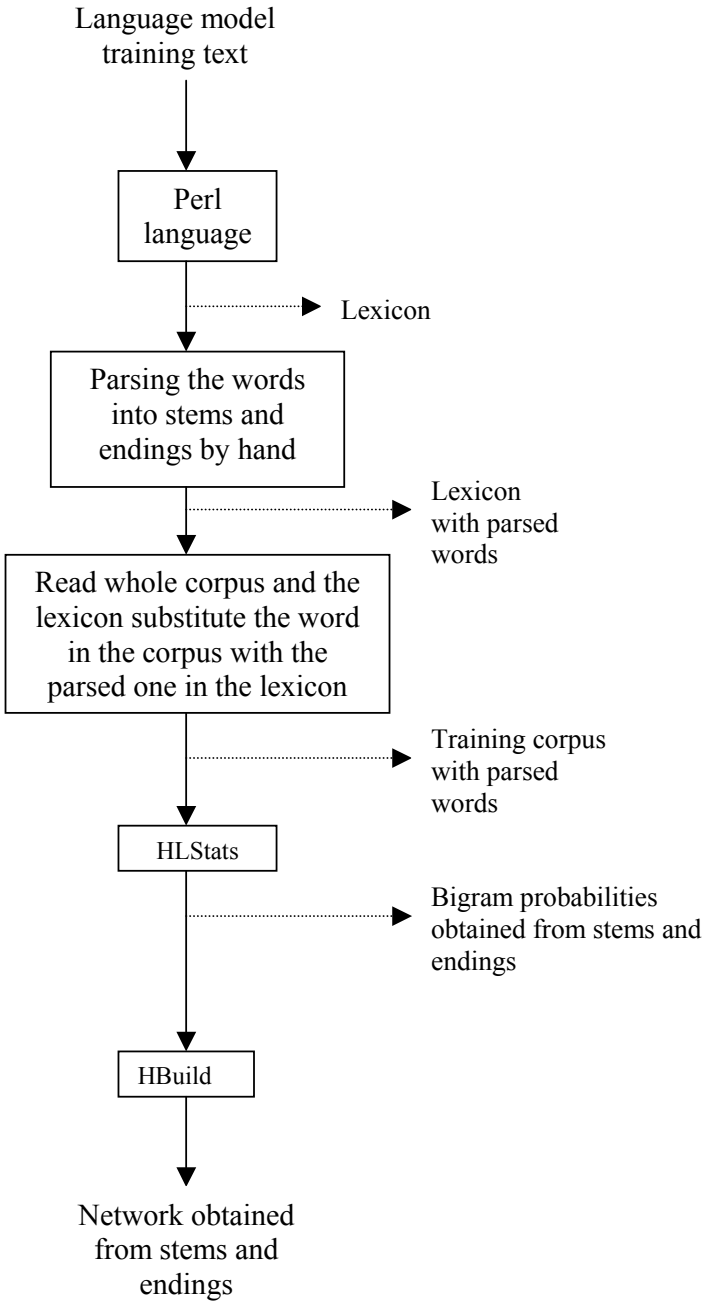



Figure A.2: Network construction for Experiment 1.

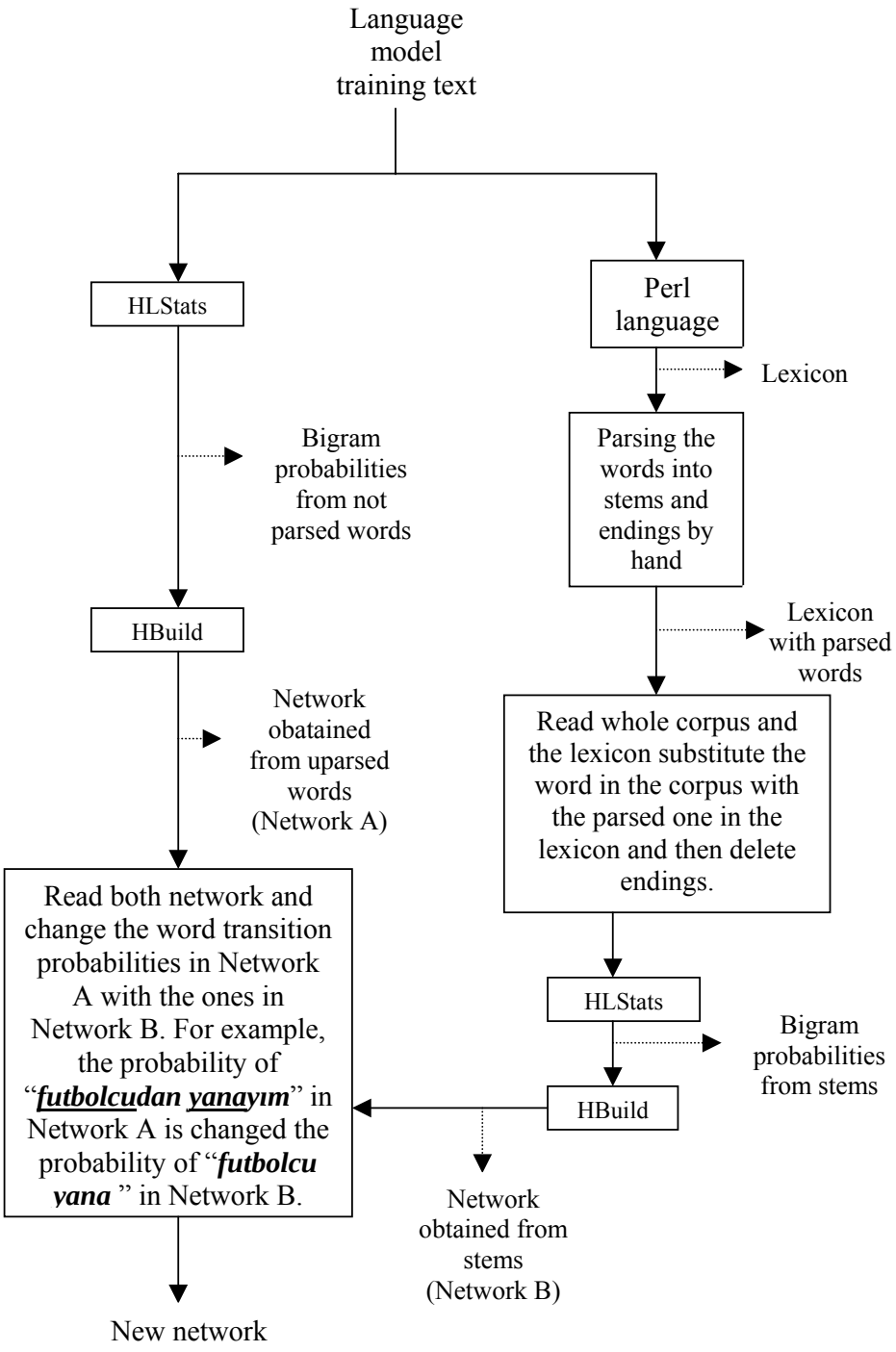



Figure A.3: Network Construction for Experiment 2.

# APPENDIX B

Table B.1: Decision tree questions.

QS 'L_v_ince_dar' {"i-*","e-*"}
QS 'R_v_ince_dar' {"*+i","*+e"}
QS 'L_v_ince_yuv' {"to-*","tu-*"}
QS 'R_v_ince_yuv' {"*+to","*+tu"}
QS 'L_v_kalin_dar' {"a-*","ti-*"}
QS 'R_v_kalin_dar' {"*+a","*+ti"}
QS 'L_v_kalin_yuv' {"o-*","u-*"}
QS 'R_v_kalin_yuv' {"*+o","*+u"}
QS 'L_nasal' {"n-*","m-*"}
QS 'R_nasal' {"*+n","*+m"}
QS 'L_v_stop' {"c-*","d-*","b-*"}
QS 'R_v_stop' {"*+c","*+d","*+b"}
QS 'L_v_stop1' {"kg-*"}
QS 'R_v_stop1' {"*+kg"}
QS 'L_v_stop2' {"g-*"}
QS 'R_v_stop2' {"*+g"}
QS 'L_unv_stop' {"t-*","p-*","tc-*"}
QS 'R_unv_stop' {"*+t","*+p","*+tc"}
QS 'L_unv_stop1' {"kk-*"}
QS 'R_unv_stop1' {"*+kk"}
QS 'L_unv_stop2' {"k-*"}
QS 'R_unv_stop2' {"*+k"}
QS 'L_v_fric' {"z-*","v-*","j-*"}
QS 'R_v_fric' {"*+z","*+v","*+j"}
QS 'L_unv_fric' {"f-*","s-*","ts-*"}

(continued)

```
QS 'R_unv_fric' {"*+f","*+s","*+ts"}
QS 'L_fisil' {"h-*"}
QS 'R_fisil' {"*+h"}
QS 'L_sessizlik' {"smsm-*","spsp-*"}
QS 'R_sessizlik' {"*+smsm","*+spsp"}
QS 'L_yum_g' {"tg-*"}
QS 'R_yum_g' {"*+tg"}
QS 'L_y' {"y-*"}
QS 'R_y' {"*+y"}
QS 'L_diger0' {"l-*"}
QS 'R_diger0' {"*+l"}
QS 'L_diger2' {"kl-*"}
QS 'R_diger2' {"*+kl"}
QS 'L_diger1' {"r-*"}
QS 'R_diger1' {"*+r"}
```

# APPENDIX C

Table C.1 : Monophone counts in acoustic training corpus.

| | Monophone model | Monophone count |
|---|---|---|
| 1 | "a" | 37698 |
| 2 | "b" | 8264 |
| 3 | "c" | 3264 |
| 4 | "d" | 13977 |
| 5 | "e" | 31026 |
| 6 | "f" | 1839 |
| 7 | "g" | 3606 |
| 8 | "h" | 3627 |
| 9 | "i" | 28763 |
| 10 | "j" | 235 |
| 11 | "k" | 6996 |
| 12 | "l" | 10373 |
| 13 | "m" | 10915 |
| 14 | "n" | 21990 |
| 15 | "o" | 9198 |
| 16 | "p" | 3043 |
| 17 | "r" | 24371 |
| 18 | "s" | 9751 |
| 19 | "t" | 11938 |
| 20 | "u" | 10190 |
| 21 | "v" | 3858 |
| 22 | "y" | 11104 |
| 23 | "z" | 5416 |
| 24 | "kg" | 753 |
| 25 | "kk" | 9009 |
| 26 | "kl" | 11272 |
| 27 | "smsm" (silence) | 7718 |
| 28 | "tc" | 4522 |
| 29 | "tg" | 3410 |
| 30 | "ti" | 16085 |
| 31 | "to" | 2670 |
| 32 | "ts" | 5614 |
| 33 | "tu" | 6560 |
| 34 | "spsp" (short pause) | 52054 |

# APPENDIX D

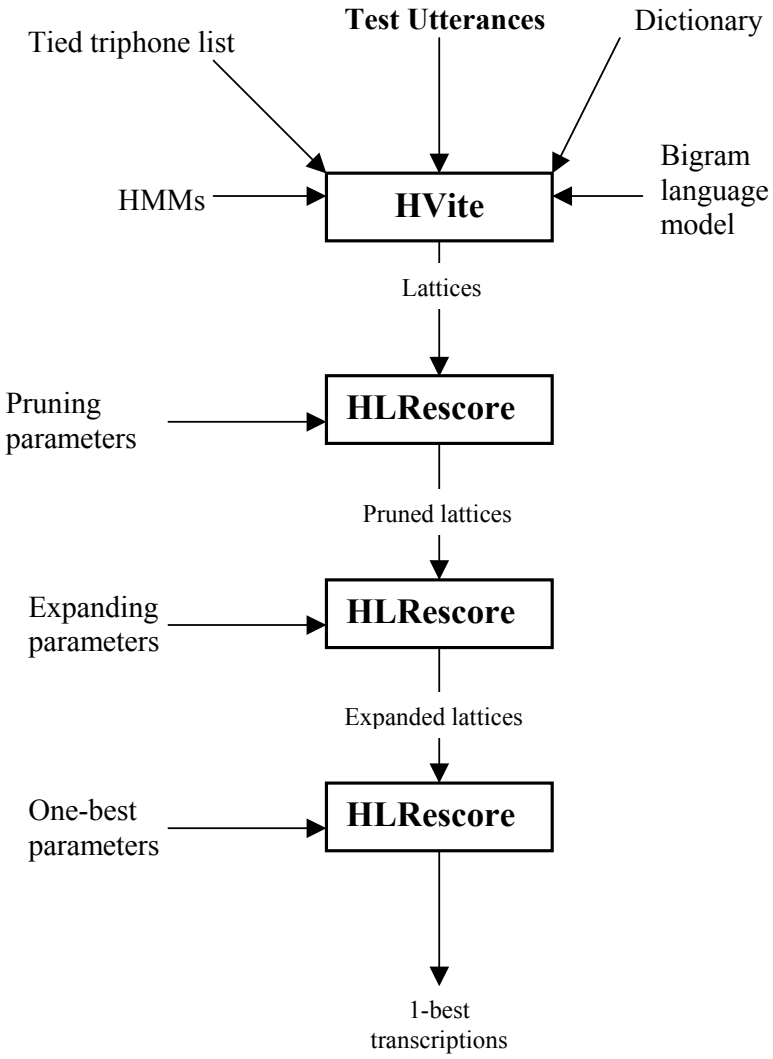



Figure D.1: Second pass phases in recognition.

# APPENDIX E

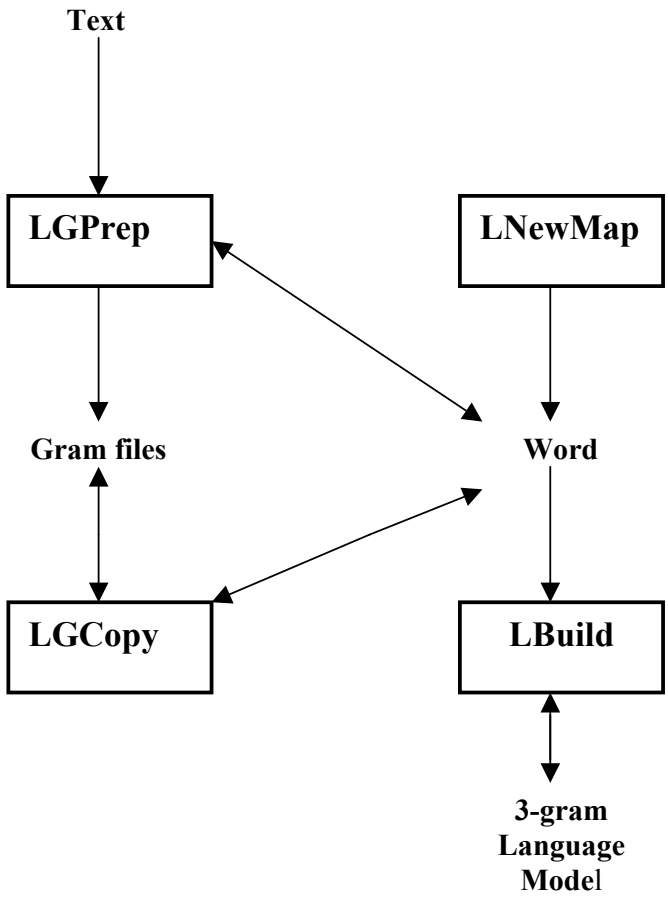



Figure E.1: The main stages in building an 3-gram language model.

# APPENDIX F

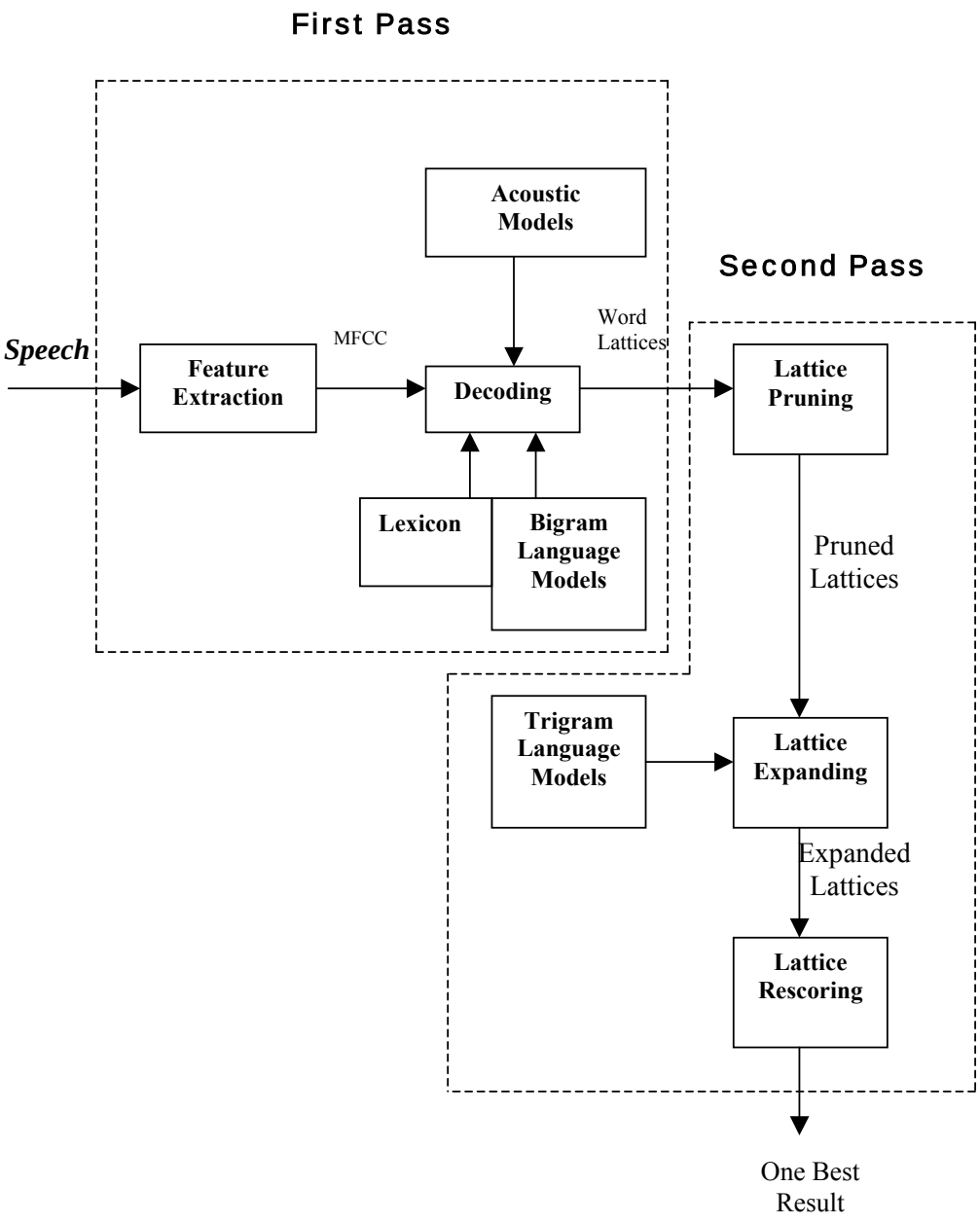



Figure F.1: Speech recognizer performing two pass.