Unsupervised Speech Processing with Applications to Query-by-Example Spoken Term Detection

by

Yaodong Zhang

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2013

Author .....
Department of Electrical Engineering and Computer Science
Jan 10, 2013

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

Accepted by .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Unsupervised Speech Processing with Applications to Query-by-Example Spoken Term Detection

by

Yaodong Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 10, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis is motivated by the challenge of searching and extracting useful information from speech data in a completely unsupervised setting. In many real world speech processing problems, obtaining annotated data is not cost and time effective. We therefore ask how much can we learn from speech data without any transcription. To address this question, in this thesis, we chose the query-by-example spoken term detection as a specific scenario to demonstrate that this task can be done in the unsupervised setting without any annotations.

To build the unsupervised spoken term detection framework, we contributed three main techniques to form a complete working flow. First, we present two posteriorgram-based speech representations which enable speaker-independent, and noisy spoken term matching. The feasibility and effectiveness of both posteriorgram features are demonstrated through a set of spoken term detection experiments on different datasets. Second, we show two lower-bounding based methods for Dynamic Time Warping (DTW) based pattern matching algorithms. Both algorithms greatly outperform the conventional DTW in a single-threaded computing environment. Third, we describe the parallel implementation of the lower-bounded DTW search algorithm. Experimental results indicate that the total running time of the entire spoken detection system grows linearly with corpus size. We also present the training of large Deep Belief Networks (DBNs) on Graphical Processing Units (GPUs). The phonetic classification experiment on the TIMIT corpus showed a speed-up of 36x for pre-training and 45x for back-propagation for a two-layer DBN trained on the GPU platform compared to the CPU platform.

Thesis Supervisor: James R. Glass
Title: Senoir Research Scientist

Acknowledgments

I would like to thank my advisor Jim Glass for his guidance, encouragement and support over the years. He is always smiling and always saying yes to let me try all my random ideas. I am also thankful to my thesis committee members Victor Zue and Tommi Jaakkola. They provided insightful suggestions to broaden and improve the research in this thesis.

I am grateful to the colleagues in the SLS group. This work would have not been possible without the support of them: Stephanie, Scott, Marcia, Ian, Jingjing, T.J., Najim, Ekapol, Jackie, Ann, Stephen, William, David, Tuka, Carrie, Xue, Yu, Xiao, Daniel, Hung-an, Yushi, Kiarash, Nate, Paul, Tara, Mitch, Ken, etc.

Finally, I would like to thank my family and friends for their constant love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Conventional automatic speech recognition (ASR) can be viewed as a nonlinear transformation from the speech signal to words [101]. Over the past forty years, the core ASR architecture has developed into a cogent Bayesian probabilistic framework. Given the acoustic observation sequence, $X = x_1, \cdots, x_n$, the goal of ASR is to determine the best word sequence, $\hat{W} = w_1, \cdots, w_m$ which maximizes the posterior probability $P(W|X)$ as

$$\hat{W} = \arg\max_W P(W|X) = \arg\max_W \frac{P(W)P(X|W)}{P(X)} \tag{1.1}$$

The speech signal X is fixed throughout the calculation so that $P(X)$ is usually considered to be a constant and can be ignored [54]. As a result, modern ASR research faces challenges mainly from the language model term $P(W)$, as well as the acoustic model term $P(X|W)$. In order to train complex statistical acoustic and language models, conventional ASR approaches typically require large quantities of language-specific speech and the corresponding annotation. Unfortunately, for real world problems, the speech data annotation is not always easy to obtain. There are nearly 7,000 human languages spoken around the world [137], while only 50-100 languages have enough linguistic resources to support ASR development [91]. Therefore, there is a need to explore ASR training methods which require significantly less supervision than conventional methods.

In recent years, along with the fast growth of speech data production, less supervised speech processing has attracted increasing interest in the speech research community [96, 97, 98, 88, 147, 59, 119]. If no human expertise exists at all, speech processing algorithms can be designed to operate directly on the speech signal with no language specific assumptions. In this scenario, the intent is not to build connections between the speech signal and the corresponding linguistic units like phones or words. With only the speech signal available, to extract valuable information, a logical framework is to simulate the human learning process. An important ability in human language learning is to learn by matching re-occurring examples [85, 122]. Information can be then inferred from the repeated examples. To apply a similar mechanism to speech signals, there are two major challenges to solve. Since the speech signal varies greatly due to different speakers or speaking environments, in order to operate directly on the signal level, the first challenge is to find robust speech feature representation methods. The feature representation needs to be carefully designed to not only capture rich phonetic information in the signal but also maintain a certain level of speaker independence and noise robustness.

The second challenge is to determine a good matching mechanism on the feature representation. Since matching will operate directly at the feature level, instead of discrete symbols, such as phones or words, matching accuracy needs to be addressed, as well as the matching speed. Note that, if most speech data is unlabeled but there a small amount of labelled data available, speech processing algorithms can be designed to make use of all speech data and try to leverage any available supervised information into the process as much as possible. Moreover, it should not be difficult to incorporate any labelled data into the original system to continuously improve the performance in the future.

In this chapter, we will describe four speech processing scenarios, requiring various degrees of human supervision. We then present our vision of the current development of unsupervised speech processing techniques. Then, we present a summary of the proposed solutions to the two challenges of representation and matching in the unsupervised speech learning. The main contributions of this thesis and a brief chapter

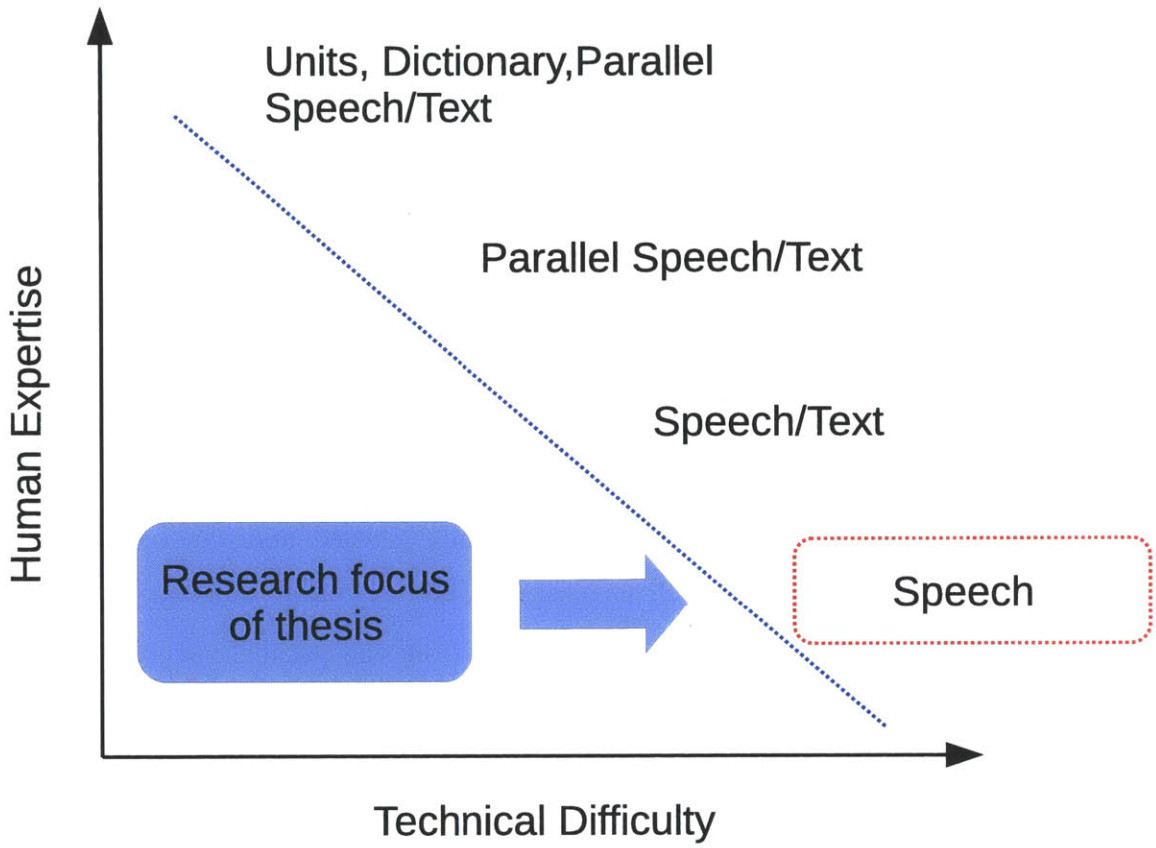


Figure 1-1: Different ASR learning scenarios. By increasing the human expertise (supervision), the technical difficulty becomes smaller for most modern statistical learning frameworks. However, less human expertise generally would cause an increase in the learning difficulty which needs to be carefully addressed and explored. The figure is adapted from [39]. This thesis primarily addresses the speech-only scenario.

summary will be also presented.

1.1 Vision

Although the conventional ASR framework has achieved tremendous success in different applications, there are alternative ways of processing speech data in different learning scenarios [39]. Shown in Figure 1-1, there are four different ASR learning scenarios classified by the different degrees of human supervision. By increasing the human expertise (supervision), the technical difficulty becomes smaller for most modern statistical learning frameworks. However, less human expertise generally

would cause an increase in the learning difficulty which needs to be carefully addressed and explored. In the following paragraphs, we will describe all four scenarios and point out what scenario this thesis is going to focus on.

The first scenario consists of problems where detailed linguistic resources including parallel speech/text transcription and phoneme-based dictionary are available. Most modern speech systems fall into this scenario and decent speech recognizers can be trained by using the well-known sub-word hidden Markov models (HMMs) [76, 6].

In the second scenario, dictionaries and linguistic units are not provided except for the parallel speech/text transcription. The learning problem becomes harder because the phoneme inventory needs to be automatically learned and the corresponding dictionary needs to be automatically generated. There has been some research aiming to learn acoustically meaningful linguistic units from the parallel speech/test data, such as the Grapheme-based letter-to-sound (L2S) learning [65], Grapheme based speech recognition [30, 69] and the pronunciation mixture models (PMMs) [5].

If only independent speech and text data are available, it is difficult to determine what words exist where in the speech data. In this scenario, it is impossible to build connections between the speech signals and the corresponding linguistic units. However, a few speech processing tasks have been explored and shown to be feasible such as discovering self-organizing units (SOUs) [119], sub-word units [134, 58], etc.

In the scenario where only speech signals are available, speech processing becomes an extreme learning case for machines, often referred to as the zero resource learning scenario for speech [37]. In recent years, this scenario has begun to draw more attention in the speech research community. Prior research showed that a variety of speech tasks can be done by looking at the speech signal alone, such as discovering word-like patterns [98, 120], query-by-example spoken term detection [38], topic segmentation [78, 79, 31] and phoneme unit learning [73], etc.

The latter, speech only scenario is particularly interesting to us for the following reasons:

1. Along with the fast growth of Internet applications and hand-held smart devices, the speed of speech data generation has greatly increased. For instance,

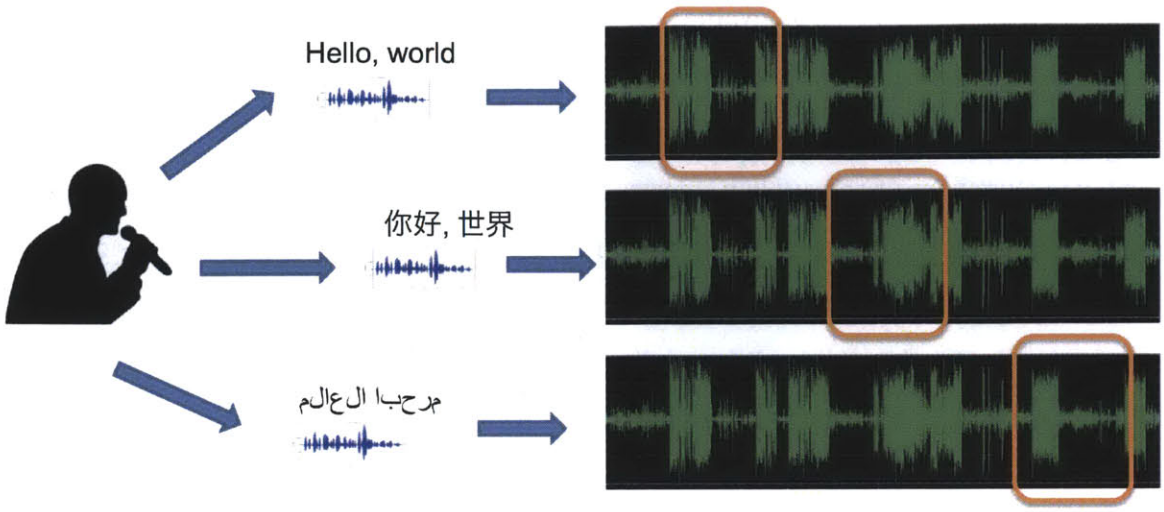


Figure 1-2: Query-by-Example Spoken Term Detection (QE-STD). A user/tester makes spoken queries and the QE-STD system locates occurrences of the spoken queries in test recordings.

hundreds of hours of news broadcasts in many languages have been recorded and stored almost every media provider. Most companies record customer service calls for future training use. Voice-based applications such as Google Voice Search [43] and Youtube [144] receive a large amount of speech data in every second. However, since none of those speech data has any form of transcription, it becomes difficult to leverage these data into the conventional highly supervised ASR training paradigm.

2. Transcribing speech data is a long and expensive process [54]. According to a variety of research reports, it is common to take four hours to produce the orthographic transcription of only one hour of speech data without time alignment [92]. In some acoustic modeling scenarios, in order to produce a time aligned phonetic transcription, it would take a professional linguistic expert a hundred hours to transcribe one hour of speech data [92]. Total cost is estimated based on $200 per hour for word level transcription and $2000 per hour for phonetic level transcription.

Therefore, in this thesis, we focus on techniques where only speech data are available, and perform an unsupervised query-by-example spoken term detection (QE-

STD) task as a means of evaluation. Spoken term detection (STD) has been an interesting research topic over many years. Figure 1-2 illustrates the concept of the QE-STD task, where a user/tester makes spoken queries and the QE-STD system locates occurrences of the spoken queries in test recordings. Conventional STD systems have developed into two directions. One direction is based on post-processing ASR output, focusing on detecting spoken terms at the recognition text level. The other direction is based on directly modeling speech and spoken terms, detecting spoken terms on the speech segment level without running ASR on every word. Although several systems [109, 140, 124, 62, 21] have demonstrated the effectiveness of both methods, both require a large amount of supervised training. For instance, the post-processing based approach requires an ASR engine trained using supervised speech data, while the model based approach needs enough examples of each spoken term, which can be comparable to the data requirements for a standard speech recognizer.

In this thesis, we focus on investigating techniques to perform spoken term detection directly on the speech signal level without using any supervision. Two robust speech feature representations and the corresponding fast matching algorithms will be proposed. We demonstrate that the proposed feature representations can reduce the speaker dependency problem, while maintaining a good level of similarity among spoken term appearances. The fast matching algorithms outperform conventional matching algorithm by a factor of four orders of magnitude.

1.2 Main Contributions

The main contributions of this thesis can be summarized as follows:

Representation. Two different robust feature representations are proposed for the QE-STD task. One is a Gaussian posteriorgram based features which are speaker independent and can be generated in completely unsupervised conditions. The other is a Deep Belief Network (DBN) posteriorgram based features which can be used to refine the Gaussian posteriorgram features in the unsupervised setting or directly generate posteriorgram features in semi-supervised or supervised settings. The fea-

sibility and effectiveness of both posteriorgram features are demonstrated through a set of spoken term detection experiments on different datasets.

Matching. Three lower-bounding based fast matching algorithms are proposed for locating spoken terms on posteriorgram features. Two algorithms can be used in single-threaded computing environments, while the third algorithm is designed to run in multi-threaded computing environments. All three algorithms greatly outperform the conventional Segmental Dynamic Time Warping (S-DTW) algorithm for the QE-STD task.

1.3 Chapter Summary

The remainder of this thesis is organized as follows:

Chapter 2 provides background knowledge and some pre-existing techniques used in this thesis.

Chapter 3 gives an overview of the proposed Gaussian posteriorgram based QE-STD framework. Using Gaussian posteriorgram features, a new unsupervised spoken term discovery system is presented to show that Gaussian posteriorgrams are able to efficiently address the speaker dependency issue in tasks other than QE-STD.

Chapter 4 introduces the new Deep Belief Network (DBN) posteriorgram based QE-STD framework. Given different levels of speech annotation, three DBN posteriorgram configurations are described. The evaluation results on TIMIT and the Babel corpus are reported and discussed. Furthermore, denoising DBN posteriorgrams are presented to show some promising results for QE-STD on noisy speech data.

Chapter 5 presents two fast matching algorithms on posteriorgram features. Both algorithms utilize a lower-bounding idea but operate on different approximation levels. Experiments and comparisons are reported and discussed.

Chapter 6 further explores a fast matching algorithm in a multi-threaded computing environment – Graphical Processing Unit (GPU) computing. A fully parallel lower-bounding based matching algorithm is described. Experimental results on a huge artificially created speech corpus are presented and discussed. In addition, effi-

cient GPU based DBN training algorithms are described and speed comparisons are presented.

Chapter 7 concludes this thesis and provides discussion of some potential improvement of the proposed QE-STD framework.

Chapter 2

Background

This chapter provides background about the techniques used in the following chapters. The conventional speech feature representation – Mel-scale cepstral coefficients (MFCCs) and the most commonly used acoustic model Gaussian Mixture Model (GMM) will be described. The Segmental Dynamic Time Warping (S-DTW) algorithm will be reviewed since it will be used in experiments as a matching method on the speech representations. Finally, we present several speech corpora that are used in the experiments performed in this thesis.

2.1 MFCC Representation

When speech is recorded by a microphone, the signal is first digitized and represented by discrete amplitudes as a function of time given a fixed sampling rate. Most modern speech recording devices have a default sampling rate of at least 16kHz for human speech, while the standard telephone speech coding method only supports 8kHz in order to save transmission bandwidth.

From the statistical learning point of view, with a high sampling rate of 16kHz or 8kHz, it is difficult to process speech directly from the waveform. Therefore, there has been a number of signal processing methods focusing on converting the speech waveform to a short-time spectral representation. A spectral representation has inherent advantages such as having lower dimensionality, yet preserving relevant

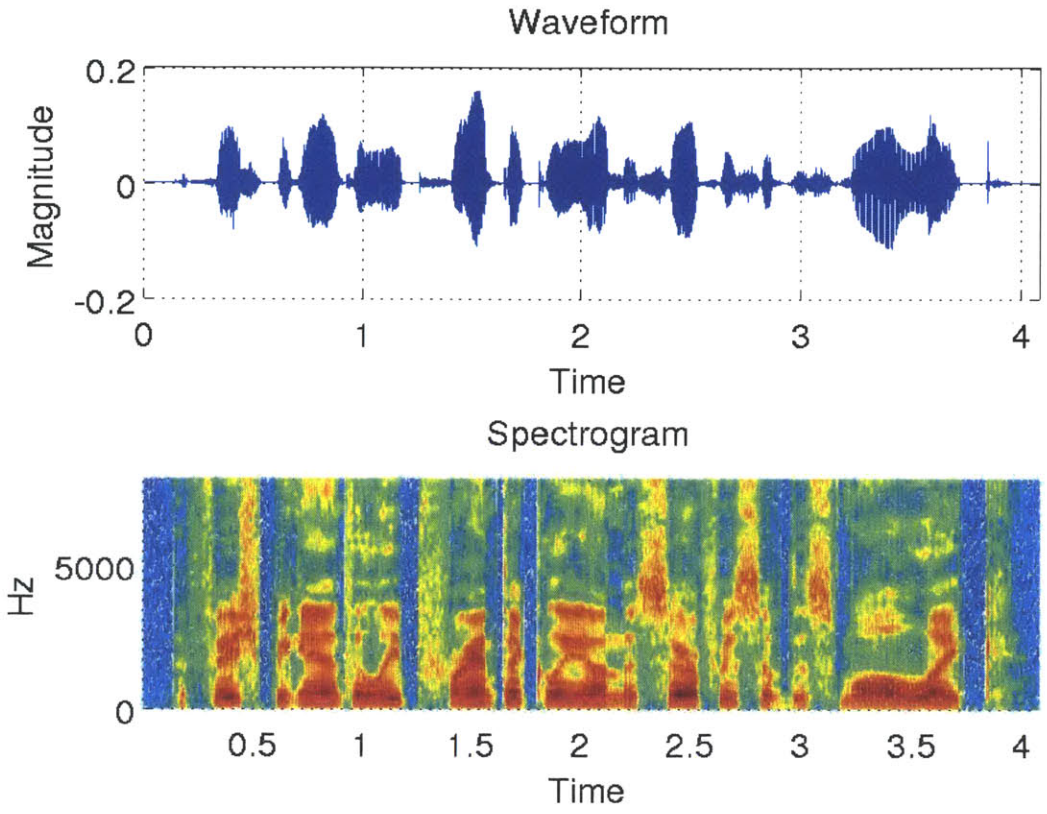


Figure 2-1: Waveform representation and the corresponding spectrogram representation of a speech segment.

phonetic information [54]. Mel-frequency cepstral coefficients (MFCCs) are one of the widely used spectral representations for ASR and have become a standard front-end module for feature extraction in most modern ASR systems.

In order to compute MFCCs for a recorded speech signal $x[t]$, the following standard steps are applied.

1. Waveform normalization and pre-emphasis filtering. A common pre-processing approach is to apply mean and magnitude normalization, followed by a pre-emphasis filter:

$$x_n[t] = \frac{x[t] - \text{mean}(x[t])}{\max|x[t]|} \tag{2.1}$$

$$x_p[t] = x_n[t] - 0.97x_n[t-1] \tag{2.2}$$

2. Calculation of short-time Fourier transform (STFT). A short-time Fourier trans-

form (STFT) is performed on the waveform with a window size of 25ms, and an analysis shift of 10ms. The most commonly used window is the Hamming window [93]. After the STFT, the speech can be represented in the spectrogram form, shown in Figure 2-1. In the following chapters, each analysis is often referred as a speech frame.

$$X_{\text{STFT}}[t,k] = \sum_{m=-\infty}^{\infty} x_p[t]w[t-m]e^{-j2\pi mk/N} \tag{2.3}$$

where w is the Hamming window, N is the number of points for the discrete Fourier transform (DFT) and $X_{\text{STFT}}[t,k]$ is the k-th spectral component at time t.

3. Calculation of Mel-frequency spectral coefficients (MFSCs). The Mel-frequency filter is designed based on an approximation of the frequency response of inner ear [85]. The Mel-filter frequency response is shown in Figure 2-2. On each speech frame after the STFT, a Mel-frequency filter is used to reduce the spectral resolution, and convert all frequency components to be placed according to the Mel-scale.

$$X_{\text{MFSC}}[t,i] = \frac{\sum_{k=-\infty}^{\infty} M_i(k)|X_{\text{STFT}}[t,k]|^2}{|M_i|} \tag{2.4}$$

where i denotes the i-th Mel-frequency filter and $|M_i|$ represents the energy normalizer of the i-th filter.

4. Calculation of discrete cosine transform (DCT). The DCT is then applied to the logarithm of the MFCSs to further reduce the dimensionality of the spectral vector. Typically only the first 12 DCT coefficients are kept. Prior research has shown that adding more DCT components does not help increase the quality of the MFCCs for ASR, although more MFCCs are often used for speaker identification tasks [107, 106, 67].

$$X_{\text{MFCC}}[t,i] = \frac{\sum_{k=0}^{M-1} 10\log_{10}\left(X_{\text{MFSC}}[t,i]\right)\cos\left(\frac{2\pi}{M}ki\right)}{M} \tag{2.5}$$

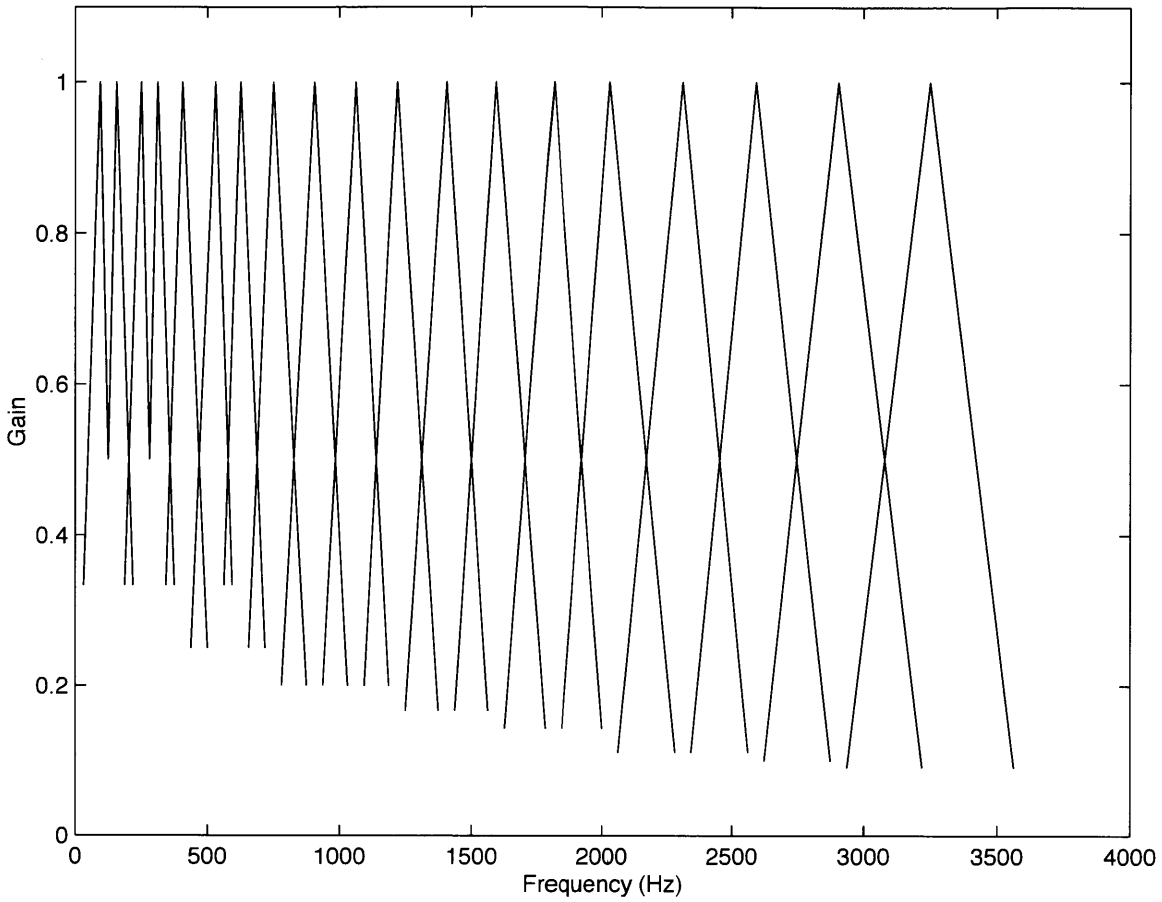


Figure 2-2: Triangular filters placed according to Mel frequency scale. A Mel-frequency filter is used to reduce spectral resolution, and convert all frequency components to be placed according to the Mel-scale.

where M represents the number of Mel-frequency filters and $X_{\text{MFCC}}[t, i]$ denotes the i-th MFCC component at time t.

After the above four steps, the calculation of MFCCs given a speech signal is complete. In practice, when using MFCCs in the acoustic modeling, long-term MFCCs features are often considered, such as delta (Δ) MFCCs and delta-delta ($\Delta\Delta$) MFCCs [54]. The Δ MFCCs are the first derivatives of the original MFCCs and the $\Delta\Delta$ MFCCs are the second derivatives of the original MFCCs. A common configuration of the modern ASR feature extraction module is to use the original MFCCs stacked with the Δ and $\Delta\Delta$ features. The original MFCCs are represented by the first 12 components of the DCT output plus the total energy (+Δ Energy

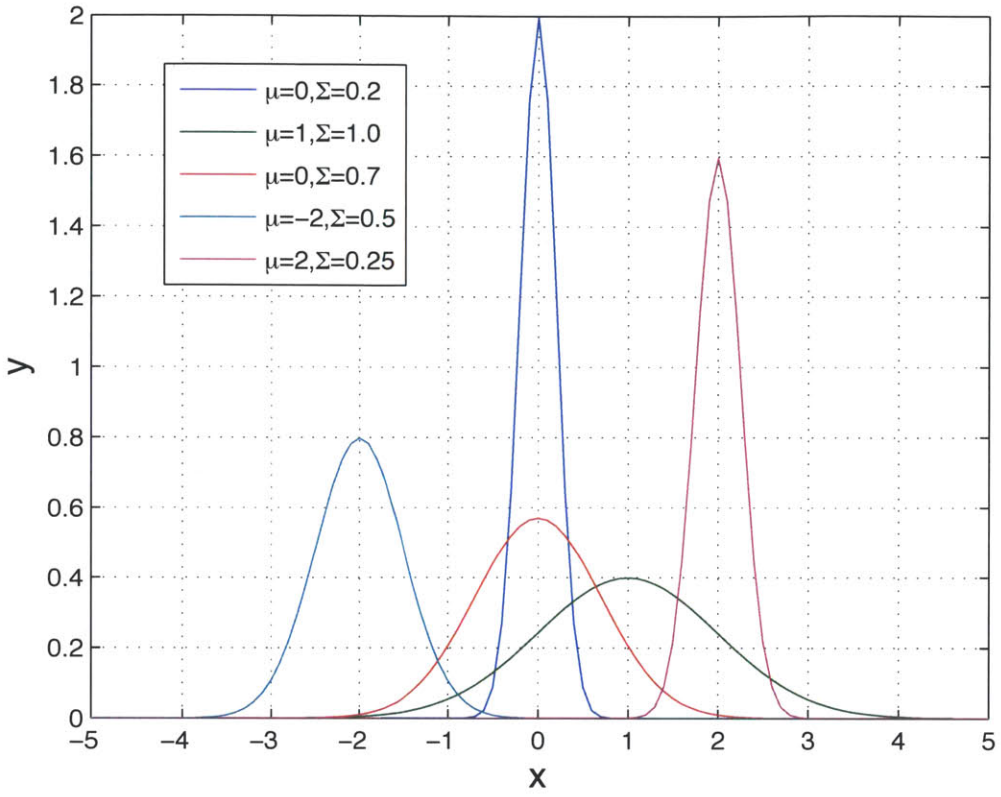


Figure 2-3: A GMM with five Gaussian components with equal weights.

$+\Delta\Delta$ Energy), which results in a 13+13+13=39 dimensional feature vector for each speech frame.

2.2 Gaussian Mixture Models

The Gaussian Mixture Model (GMM) is a widely used technique for modeling speech features such as MFCCs [101]. A standard GMM with K Gaussian components can be written as

$$P(x) = \sum_{i=1}^{K} w_i \mathbf{N}(x; \mu_i, \Sigma_i), \quad \sum_{i=1}^{K} w_i = 1 \tag{2.6}$$

where x is the speech observation (e.g., the 39-dimensional MFCCs), w_i represents the scaling factor and sums to one, $\mathbf{N}$ is a multivariate Gaussian distribution with

mean μ and variance Σ. Figure 2-3 shows a GMM with five components with equal weights. Considering one Gaussian distribution, given the observation vector x, the log-probability can be written as

$$\log(\mathbf{N}(x;\mu,\Sigma)) = -\frac{\log(2\pi)D}{2} - \frac{\log(|\Sigma|)}{2} - \frac{(x-\mu)^\top \Sigma^{-1}(x-\mu)}{2} \tag{2.7}$$

where D is the dimensionality of the observation vector and $|\Sigma|$ denotes the determinant of the covariance matrix. Given a speech corpus $X = x_1, x_2, \cdots, x_N$, the log-probability of the entire speech corpus is

$$\sum_{i=1}^{N} \log(\mathbf{N}(x_i;\mu,\Sigma)) \tag{2.8}$$

2.3 Dynamic Time Warping

Dynamic Time Warping (DTW) is a well-known dynamic programming technique for finding the best alignment between two time series patterns [54]. DTW became popular in the speech research community from the late 1970's to mid 1980's, and was used for both isolated and connected-word recognition with spectrally-based representations such as Linear Prediction Coding (LPC) [101]. DTW allowed for minor local time variations between two speech patterns which made it a simple and efficient search mechanism. Over time, DTW-based techniques were supplanted by hidden Markov models (HMMs) which were a superior mathematical framework for incorporating statistical modeling techniques. However, DTW-based search has remained attractive and has been used by researchers incorporating neural network outputs for ASR [74, 48], and more recently for scenarios where there is little, if any, training data to model new words [27, 139, 98].

One attractive property of DTW is that it makes no assumptions about underlying linguistic units. Thus, it is amenable to situations where there is essentially no annotated data to train a conventional ASR engine. In this thesis, we are interested in developing speech processing methods that can operate in such unsupervised conditions. For the QE-STD task, we have an example speech query pattern and we wish

to find the top K nearest-neighbor (KNN) matches in some corpus of speech utterances. DTW is a natural search mechanism for this application, though depending on the size of the corpus, there can be a significant amount of computation involved in the alignment process.

Formally, given two time series P and Q with length n and m

$$P = p_1, p_2, \cdots, p_n \tag{2.9}$$

$$Q = q_1, q_2, \cdots, q_m \tag{2.10}$$

The DTW can be used to construct an optimal alignment $\hat{A}$ between P and Q as

$$\hat{A}_\phi = \phi_1, \phi_2, \cdots, \phi_K \tag{2.11}$$

where K is the warping length and each ϕ_i is a warping pair $\phi_i = (\phi_a, \phi_b)$ which aligns p_a with q_b according to the pre-defined distance metric $D(p_a, q_b)$. Note that both p_a and q_b can be multi-dimensional as long as the distance function D can be properly defined. It is clear that the warping begins with $\phi_1 = (p_1, q_1)$ and ends with $\phi_K(p_n, q_m)$ and the length K is bounded by $\max(n, m) \leq K \leq n + m$. The DTW can find the optimal warping path $\hat{A}$ given the following objective function

$$\text{DTW}(P, Q) = \hat{A} = \min_A \sum_{i=1}^{K_A} D(\Phi(A)_i) \tag{2.12}$$

where Φ denotes the set of all possible warping paths and $\Phi(A)_i$ represents the i-th warping pair of the warping path A. Figure 2-4 shows two sinusoid signals with random Gaussian noise. If we perform DTW on those two signals and find an optimal alignment in terms of Euclidean distance, the alignment is shown in red in the middle sub-figure of Figure 2-5. Lighter pixels denote smaller Euclidean distance, while darker pixels denote larger distance.

One important property of the DTW alignment is that the warping process must be monotonically increasing and every p_i and q_j must appear in the optimal alignment

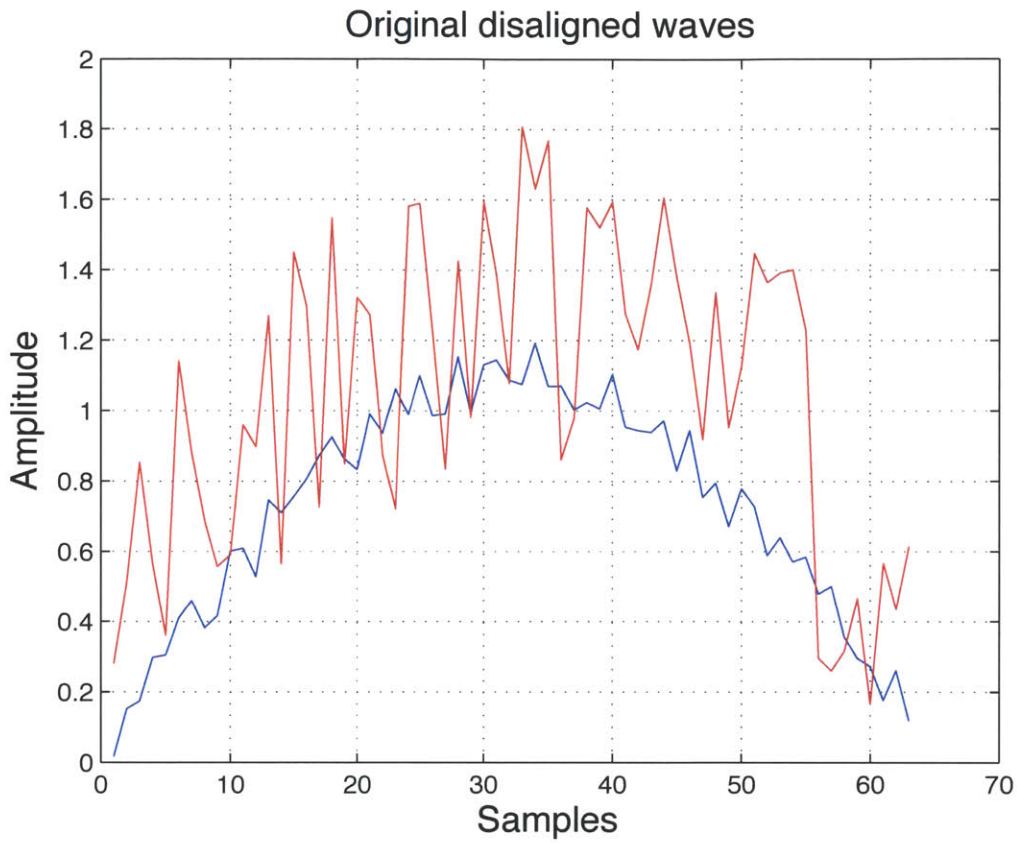


Figure 2-4: Two sinusoid signals with random Gaussian noise. DTW is used to find an optimal alignment between those two signals in terms of Euclidean distance.

$\hat{A}$ at least once. This property leads to an efficient dynamic programming algorithm that computes the optimal warping path $\hat{A}$ in $O(nm)$. Specifically, the algorithm starts from solving a minimum subsequence of P and Q and grows the minimum subsequence iteration by iteration until the full optimal warping path is found. For the above time series P and Q, the core idea of computing DTW can be illustrated by a cost matrix C of size n by m. Each element $C_{i,j}$ represents the minimum total warping path cost/distance from (p_1, q_1) to (p_i, q_j) and it can be calculated as

$$C_{i,j} = D(p_i, q_j) + \min \begin{cases} C_{i-1,j} \\ C_{i,j-1} \\ C_{i-1,j-1} \end{cases} \tag{2.13}$$

Given the initial values $C_{0,0} = 0, C_{0,j} = \infty, C_{i,0} = \infty$, it is clear that starting from

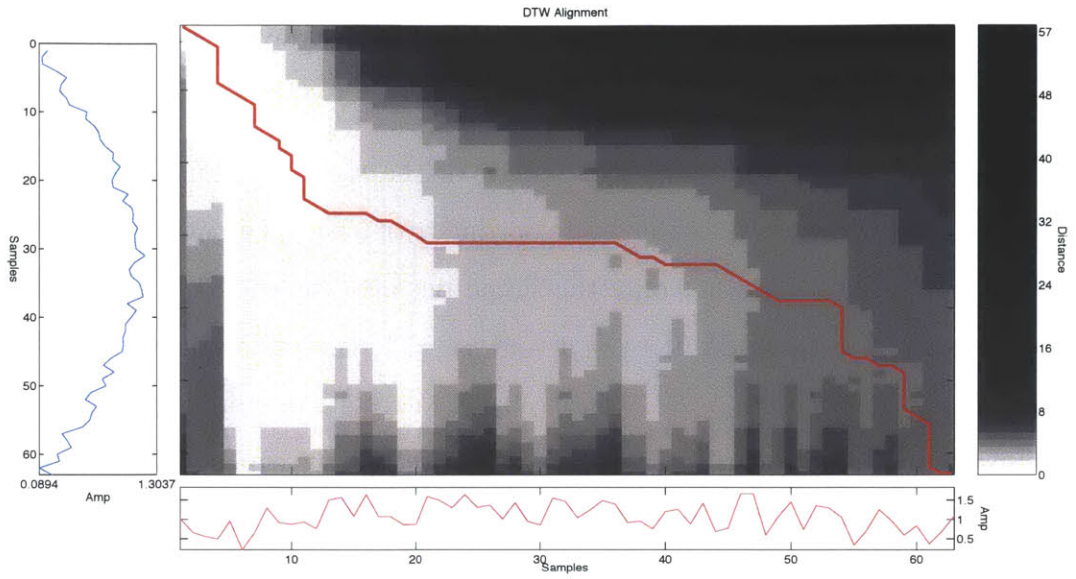


Figure 2-5: The optimal alignment of the two sinusoid signals after performing DTW with Euclidean distance as the local distance metric. The red line represents the optimal alignment. Lighter pixels denote smaller Euclidean distance, while the darker pixels denote lager Euclidean distance.

$C_{1,1}$, C can be constructed row by row and the total cost/distance of the optimal warping path is $C_{n,m}$.

2.3.1 Segmental Dynamic Time Warping

Segmental Dynamic Time Warping (S-DTW) is a segmental variant of the original DTW algorithm that finds optimal alignments among subsequences of two time series [98]. The S-DTW is a two-staged algorithm. In the first stage, the S-DTW finds local alignments among subsequences of the input two time series. In the second stage, a path refinement approach is used to eliminate lower distortion regions in the local alignments found in the first stage.

S-DTW defines two constraints on the DTW search. The first one is the commonly used adjustment window condition [112]. In our case, formally, suppose we use the same above two time series A and B, the warping pair $w(\cdot)$ defined on a $n \times m$ timing difference matrix is given as $w_i = (i_k, j_k)$ where i_k and j_k denote the k-th coordinate of the warping path. Due to the assumption that the duration fluctuation is usually

small in speech [112], the adjustment window condition requires that $|i_k - j_k| \leq R$. This constraint prevents the warping process from going too far ahead or behind in either A or B.

The second constraint is the step length of the start coordinates of the DTW search. It is clear that if we fix the start coordinate of a warping path, the adjustment window condition restricts not only the shape but also the ending coordinate of the warping path. For example, if $i_1 = 1$ and $j_1 = 1$, the ending coordinate will be $i_{end} = n$ and $j_{end} \in (1 + n - R, 1 + n + R)$. As a result, by applying different start coordinates of the warping process, the difference matrix can be naturally divided into several continuous diagonal regions with width $2R + 1$, shown in the Figure 2-6. In order to avoid the redundant computation of the warping function as well as taking into account warping paths across segmentation boundaries, an overlapped sliding window strategy is usually used for the start coordinates ($s1$ and $s2$ in the figure). Specifically, with the adjustment window size R, every time we move R steps forward for a new DTW search. Since the width of each segmentation is $2R + 1$, the overlapping rate is 50%.

By moving the start coordinate along the i and j axis, a total of $\lfloor \frac{m-1}{R} \rfloor + \lfloor \frac{n-1}{R} \rfloor$ warping paths can be obtained, each of which represents a warping between two subsequences in the input time series.

The warping path refinement is done in two steps. In the first step, a length L constrained minimum average subsequence finding algorithm [77] is used to extract consecutive warping fragments with low distortion scores. In the second step, the extracted fragments are extended by including neighboring frames below a certain distortion threshold α. Specifically, neighboring frames are included if their distortion scores are within $1 + \alpha$ percent of the average distortion of the original fragment.

In this thesis, for the QE-STD task, we use a modified single-sided S-DTW algorithm which finds optimal alignments of a full time series against all subsequences in another time series. For the spoken term discovery task, the full S-DTW algorithm is used.

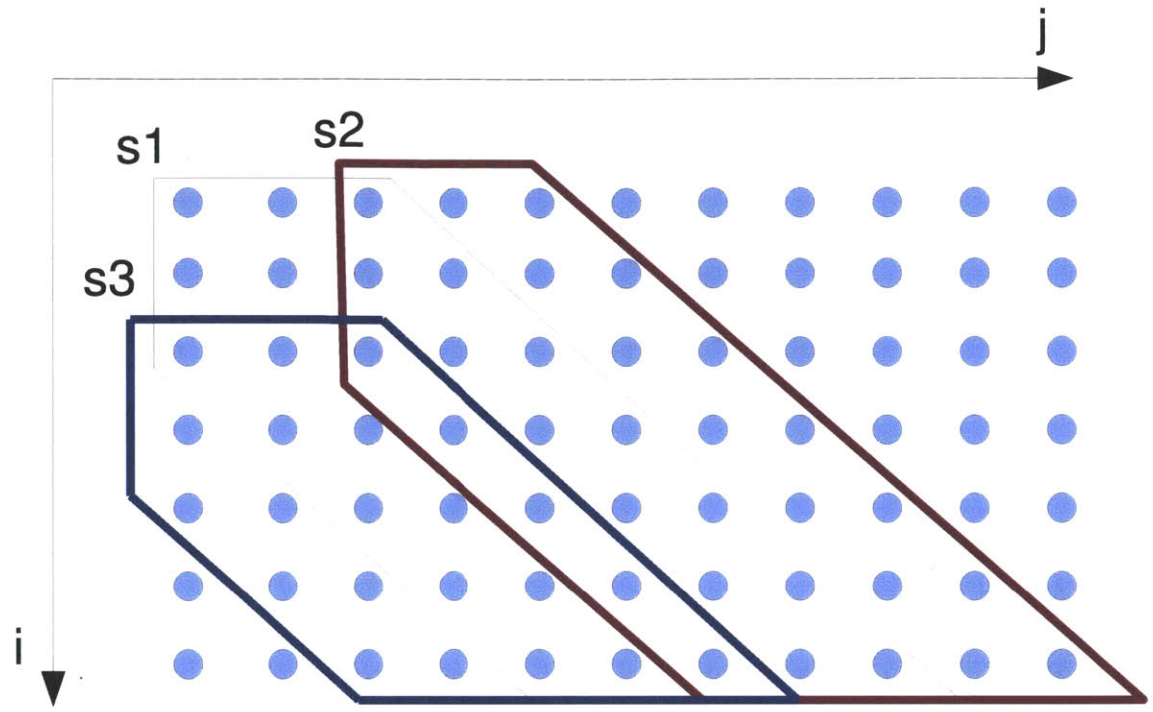


Figure 2-6: An illustration of S-DTW between two utterances with $R = 2$. The blue and red regions outline possible DTW warping spaces for two different starting times. In order to avoid the redundant computation of the warping function as well as taking into account warping paths across segmentation boundaries, an overlapped sliding window strategy is usually used for the start coordinates ($s1$ and $s2$ in the figure).

2.4 Speech Corpora

Four speech corpora are used in the experiments in this thesis. A brief overview of each speech corpus will be described in the following sections.

2.4.1 TIMIT

The TIMIT corpus consists of read speech recorded in quiet environments. It is designed for acoustic-phonetic studies and for the development and evaluation of ASR systems. The TIMIT corpus contains broadband recordings of 630 speakers of 8 major dialects of American English. Each speaker reads 10 phonetically rich sentences, include 2 "sa" sentences representing dialectal differences, 5 "sx" sentences covering phoneme pairs and 3 "si" sentences that are phonetically diverse. The TIMIT

transcriptions have been hand verified, including time-aligned orthographic, phonetic and word transcriptions. The TIMIT corpus is well-known for its balanced phoneme inventory and dialectal coverage [71].

The TIMIT corpus is divided into three sets. The training set contains 3,696 utterances from 462 speakers. The development set consists of 400 utterances from 50 speakers. The test set includes 944 utterances from 118 speakers. A commonly used core test set for ASR evaluations is a subset of the full test set, containing 192 utterances from 24 speakers. The results in this thesis are reported on the full test set.

2.4.2 NTIMIT

NTIMIT is a noisy version of the TIMIT corpus. It was collected by transmitting all 6,300 original TIMIT recordings through a telephone handset and over various channels in the NYNEX telephone network and re-digitizing them [57]. The recordings were transmitted through ten Local Access and Transport Areas, half of which required the use of long-distance carriers. The re-recorded waveforms were time-aligned with the original TIMIT waveforms so that the TIMIT time-aligned transcriptions can be used with the NTIMIT corpus as well. The training/development/test set division is the same as the original TIMIT corpus. This corpus is used for evaluating spoken term detection in noisy conditions in this thesis.

2.4.3 MIT Lecture

The MIT Lecture corpus consists of more than 300 hours of speech data recorded from eight different subjects and over 80 general seminars [41]. In most cases, the data is recorded in a classroom environment using a lapel microphone. The recordings were manually transcribed including tags for disfluencies. The vocabulary size is 27,431 words. A standard training set contains 57,351 utterances and a test set is comprised of 7,375 utterances.

The MIT Lecture corpus is generally recognized as a difficult dataset for ASR

evaluations for three reasons. First, the data is comprised of spontaneous speech as well as many disfluencies such as filled pauses, laughter, false starts and partial words. Second, since the data was recorded in a classroom environment, there are many non-speech artifacts that occur such as background noise and students' random talking. Third, some lecture specific words are uncommon and can result in significant out-of-vocabulary problems. This corpus is used in both spoken term detection and spoken term discovery experiments in this thesis.

2.4.4 Babel Cantonese

The Babel Cantonese corpus contains a training set of 50 hours of telephone speech and a test set of 200 minutes of telephone speech. The speech data was produced by presenting a topic to native Cantonese speakers and asking them to make a 10-minute long telephone call about the topic. The telephone calls were recorded by different telephone carriers, which results in very different channel noise levels for each call. This corpus is used to demonstrate the language independent feature of the proposed spoken term detection system in this thesis.

2.5 Summary

In this chapter, we described some well-established speech processing techniques that will be utilized in this thesis. We first discussed how to calculate the MFCC features for speech. Next, we presented a common acoustic modeling framework using GMMs on MFCC features. The well-known DTW and its variant S-DTW algorithms were reviewed. In the end, we described four datasets that will be used in the following experiments.

Chapter 3

Unsupervised Gaussian Posteriorgram

In this chapter, we present an overview of the unsupervised Gaussian posteriorgram framework. The Gaussian posteriorgram framework was our first attempt to represent speech in the posteriorgram form without using any supervised annotation. The core idea is to train a Gaussian mixture model (GMM) without using any supervised annotation, and represent each speech frame by calculating a posterior distribution over all Gaussian components. A modified DTW matching algorithm can be used to evaluate the similarity between two speech segments represented by Gaussian posteriorgrams in terms of an inner-product distance. The entire process is completely unsupervised and does not depend on speakers. After the success of using Gaussian posteriorgrams on a spoken term discovery task [147], a query-by-example spoken term detection task was then performed [146], which further demonstrate the effectiveness of using the Gaussian posteriorgram as a robust unsupervised feature representation of speech.

3.1 Posteriorgram Representation

The posteriorgram representation for speech data was inspired by the widely used posterior features in template-based speech recognition systems [48, 32, 3, 4]. For

example, in the Tandem [48, 32] speech recognition system, a neural network is discriminatively trained to estimate posterior probability distributions across a phone set. The posterior probability for each frame on each phone class is then used as the feature input for a conventional Gaussian mixture model with hidden Markov model (GMM-HMM) based speech recognition system. The motivation behind using posterior features instead of spectral features is that by passing through a discriminatively trained classifier, speaker dependent, unevenly correlated and distributed spectral features are converted into a simpler; and speaker-independent statistical form while still retaining phonetic information. The subsequent modeling process can focus more on capturing the phonetic differences rather than directly dealing with the speech spectrum. Previous results showed that a large improvement in terms of word recognition error rate could be obtained [48, 32].

The most recent work by Hazen et al. [46] showed a spoken term detection system using phonetic posteriorgram templates. A phonetic posteriorgram is defined by a probability vector representing the posterior probabilities of a set of pre-defined phonetic classes for a speech frame. By using an independently trained phonetic recognizer, each input speech frame can be converted to its corresponding posteriorgram representation. Given a spoken sample of a term, the frames belonging to the term are converted to a series of phonetic posteriorgrams by phonetic recognition. Then, Dynamic Time Warping (DTW) is used to calculate the distortion scores between the spoken term posteriorgrams and the posteriorgrams of the test utterances. The detection result is given by ranking the distortion scores.

To generate a phonetic posteriorgram, a phonetic classifier for a specific language is needed. In the unsupervised setting, there is no annotated phonetic information that can be used to train a phonetic classifier. Therefore, instead of using a supervised classifier, our approach is to directly model the speech using a GMM without any supervision. In this case, each Gaussian component approximates a phone-like class. By calculating a posterior probability for each frame on each Gaussian component, we can obtain a posterior feature representation called a Gaussian posteriorgram.

While the discriminations of a Gaussian posteriorgram do not directly compare

to phonetic classes, the temporal variation in the posteriorgram captures the important phonetic information in the speech signal, providing some generalization to a purely acoustic segmentation. With a Gaussian posteriorgram representation, some speech tasks could be performed without any annotation information. For example, in a query-by-example spoken term detection system, the input query which is represented by a series of Gaussian posteriorgrams can then be searched in the working data set which is also in the Gaussian posteriorgram representation [146]. In a spoken term discovery system, given a speech recording, if we want to extract frequently used words/short phrases, a Gaussian posteriorgram can be used to represent the entire recording, and a pattern matching algorithm can be applied to find similar segments of posteriorgrams [147]. In this chapter, we will demonstrate how the Gaussian posteriorgram can be used as a robust, speaker independent representation of unlabeled speech data.

3.2 Gaussian Posteriorgram Generation

If a speech utterance S contains n frames $S = (\vec{f_1}, \vec{f_2}, \cdots, \vec{f_n})$, then the Gaussian posteriorgram for this utterance is defined by $\mathrm{GP}(S) = (\vec{q_1}, \vec{q_2}, \cdots, \vec{q_n})$. The dimensionality of each $\vec{q_i}$ is determined by the number of Gaussian components in the GMM, and each $\vec{q_i}$ can be obtained by

$$\vec{q_i} = \{P(C_1|\vec{f_i}), P(C_2|\vec{f_i}), \cdots, P(C_m|\vec{f_i})\} \tag{3.1}$$

where the j-th dimension in $\vec{q_i}$ represents the posterior probability of the speech frame $\vec{f_i}$ on the j-th Gaussian component C_j. m is the total number of Gaussian components.

The GMM is trained on all speech frames without any transcription. In this work, each raw speech frame is represented by the first 13 Mel-frequency cepstrum coefficients (MFCCs). After pre-selecting the number of desired Gaussian components, the K-means algorithm is used to determine an initial set of mean vectors. Then, Expectation-maximization (EM) based GMM training is performed.

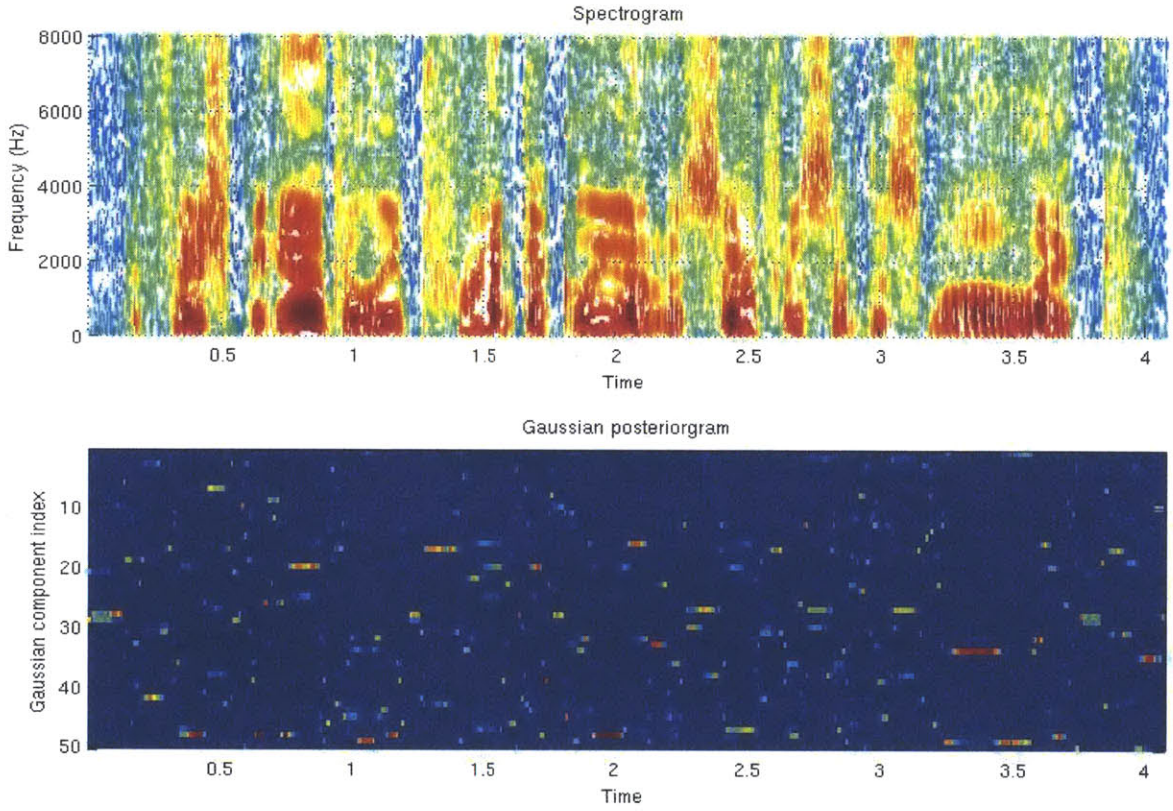


Figure 3-1: A spectrogram (top) and Gaussian posteriorgram (bottom) of a TIMIT utterance. To generate posteriorgrams, a 50-component GMM was trained on 13-MFCC features on the standard TIMIT training set. In the bottom figure, the y axis denotes the Gaussian component indices. Each pixel (x, y) represents the posterior probability of Gaussian component y for speech frame x. Blue pixels denote lower probabilities while red pixels indicate higher probabilities. It can be seen from the bottom figure that Gaussian posteriorgrams are able to represent phonetically similar speech frames. For example, the frames from 3.25s to 3.48s have the most posterior probability mass on the 33^{rd} GMM component, while the frames from 1.30s to 1.37s have the most posterior probability mass on 17^{th} GMM component and some mass on the 43^{rd} component.

Figure 3-1 shows an example of the Gaussian posteriorgram representation for a TIMIT utterance. The top figure is the spectrogram representation. In the bottom figure, the y axis denotes the Gaussian component indices. Each pixel (x, y) represents the posterior probability of Gaussian component y for speech frame x. Blue pixels denote lower probabilities while red pixels indicate higher probabilities. It can be seen from the bottom figure that Gaussian posteriorgrams are able to represent phonetically similar speech frames. For example, the frames from 3.25s to 3.48s have the most posterior probability mass on the 33^{rd} GMM component, while the frames from 1.30s to 1.37s have the most posterior probability mass on 17^{th} GMM component and some mass on the 43^{rd} component. By looking at the spectrogram, we could derive that the 33^{rd} GMM component probably corresponds to back vowels, while the 17^{th} and 43^{rd} GMM components probably correspond to fricatives.

3.3 Analysis of Gaussian Posteriorgram

In Gaussian posteriorgram generation, we assume that through unsupervised GMM training, each Gaussian component can be viewed as a self-organizing phonetic class. To validate this assumption, a phonetic histogram analysis is applied to visualize the underlying acoustically meaningful information represented by each Gaussian component. Specifically, after unsupervised GMM training, each speech frame is artificially labeled by the index of the most probable Gaussian component. Since the TIMIT dataset provides a time-aligned phonetic transcription, for each Gaussian component, we can calculate the normalized percentage of how many times one Gaussian component represents a particular phone. By drawing a bar for every TIMIT phone for one Gaussian component, we can have a histogram of the underlying phonetic distribution of that Gaussian component.

Figure 3-2 shows the phonetic distribution of frames assigned to Gaussian component 13. Each bar in the figure represents the percentage of time that this cluster represents a phone class in TIMIT. Note that the size of the TIMIT phone inventory is 61. The top sub-figure is the histogram of the training set, while the bottom sub-

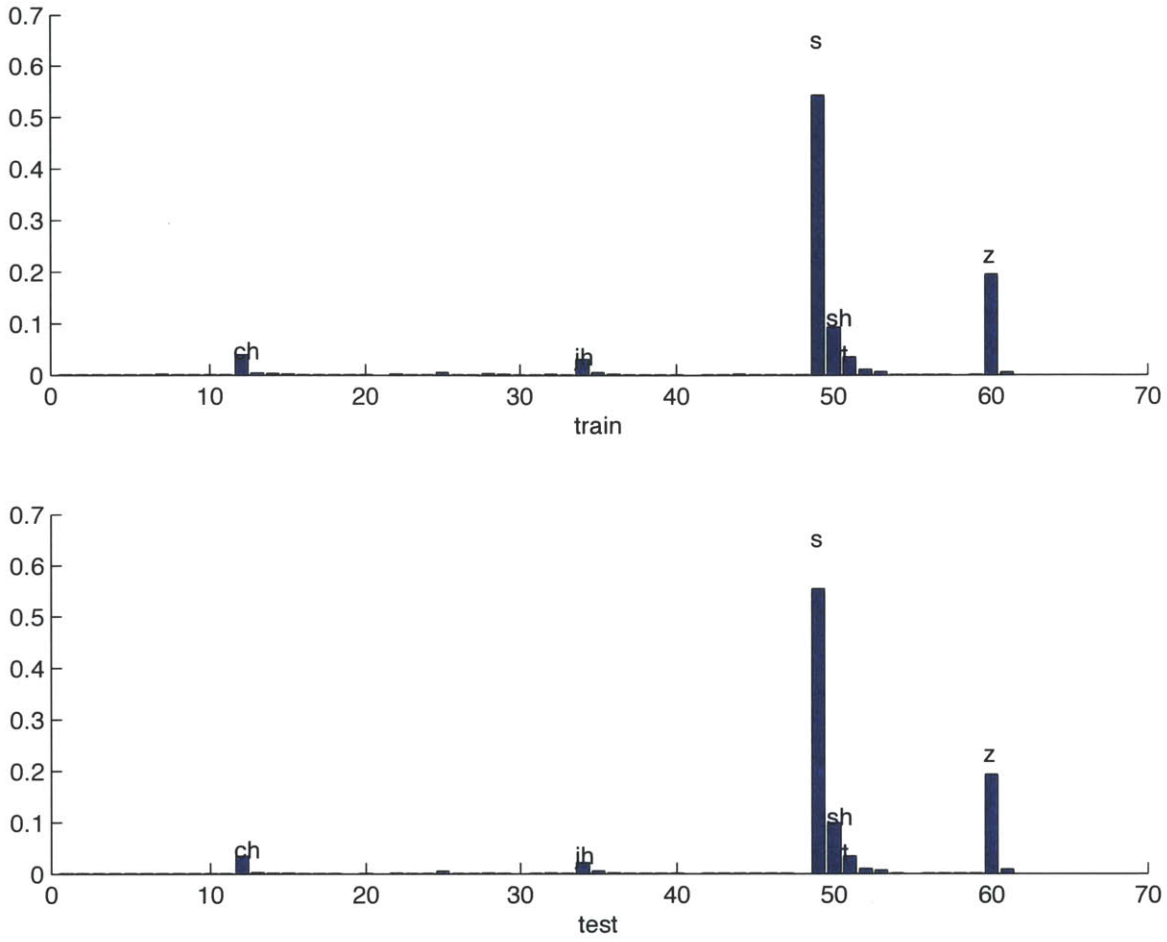


Figure 3-2: This figure illustrates the phonetic distribution of frames assigned to Gaussian component 13. Each bar in the figure represents the percentage of times that this cluster represents a phone. The top sub-figure is on the training set, while the bottom sub-figure is on the test set. It is clear that this cluster mainly represents fricatives and affricates.

figure is the histogram of the test set. It is clear that this cluster mainly represents fricatives and the release of affricates.

Figure 3-3 illustrates the phonetic distribution of frames assigned to Gaussian component 41. Since most bars are labeled as vowels or semi-vowels, this cluster mainly represents vowels and semi-vowels. Note that most vowels are front-vowels.

Figure 3-4 illustrates the phonetic distribution of frames assigned to Gaussian component 14. This cluster mainly represents retroflexed sound such as /r/, /er/ and /axr/.

It can seen from these three figures that although the GMM is trained without

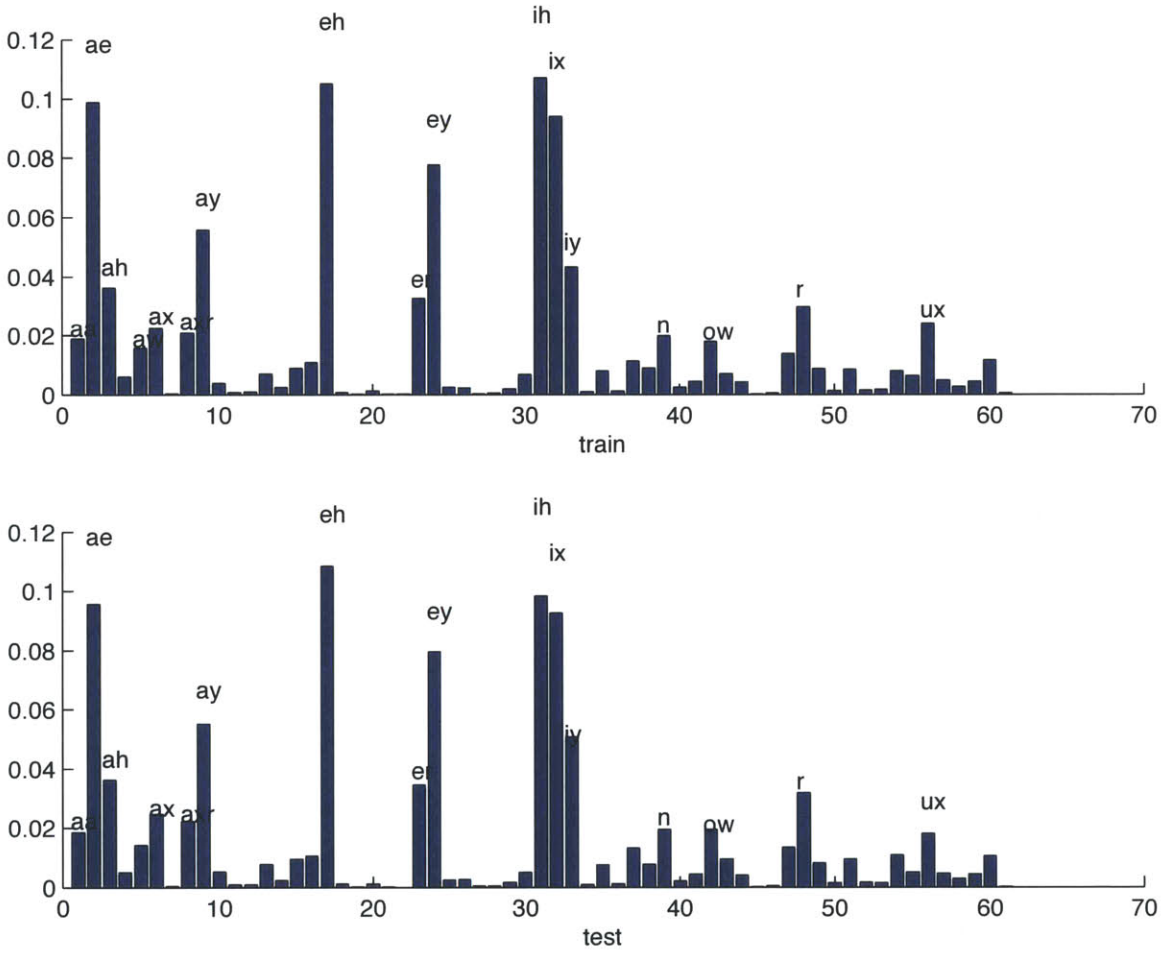


Figure 3-3: This figure illustrates the phonetic distribution of frames assigned to Gaussian component 41. Each bar in the figure represents the percentage of times that this cluster represent a phone. The top sub-figure is on the training set, while the bottom sub-figure is on the test set. Since most bars are labeled as vowels or semi-vowels, it is clear that this cluster mainly represents vowels and semi-vowels.

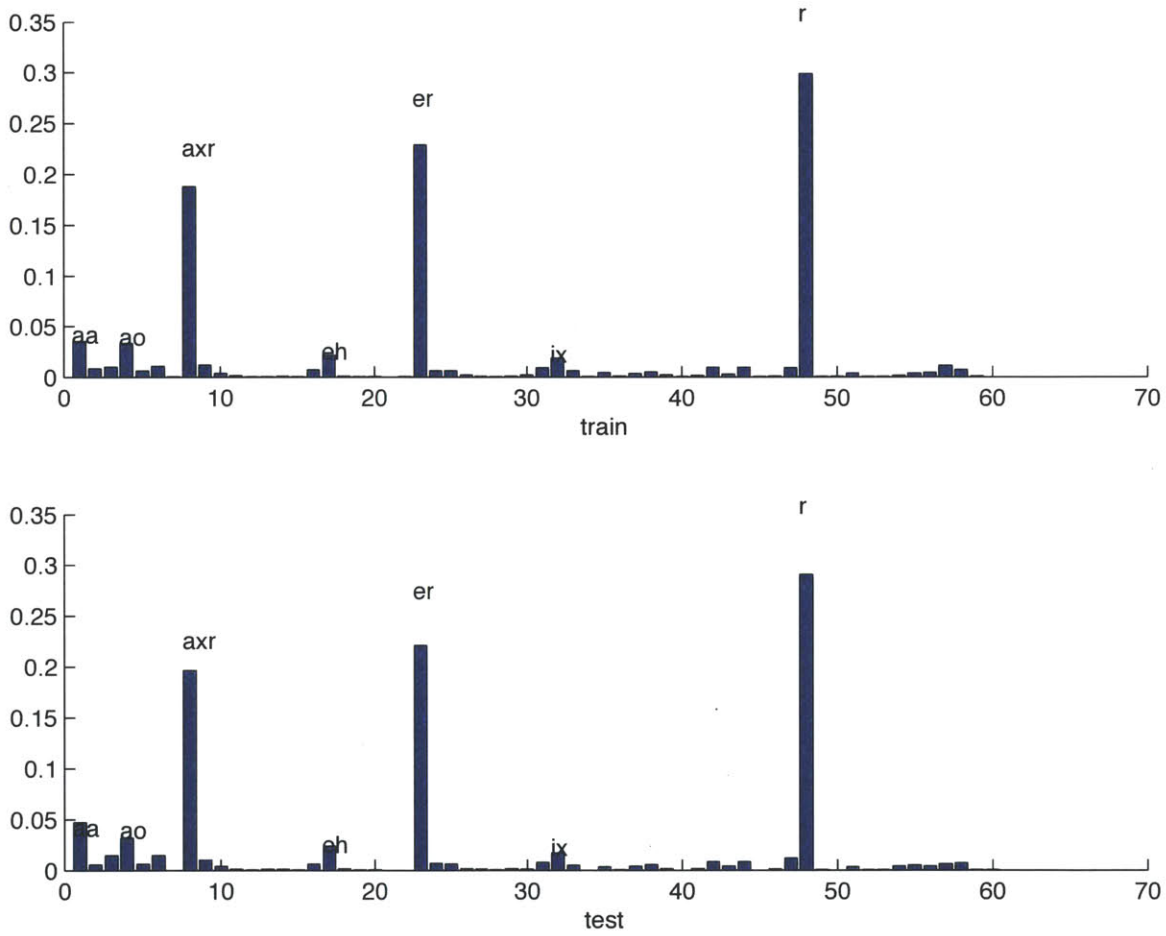


Figure 3-4: This figure illustrates the phonetic distribution of frames assigned to Gaussian component 14. Each bar in the figure represents the percentage of times that this cluster represent a phone. The top sub-figure is on the training set, while the bottom sub-figure is on the test set. This cluster mainly represents retrofled sound.

any supervised information, each Gaussian component models a relatively small set of phones sharing similar acoustic characteristics. In addition, the phonetic histograms are consistent on both training and test sets, which indicates that learned unsupervised knowledge generalizes well to unseen data. Therefore, this analysis provides qualitative explanation of why an unsupervised Gaussian posteriorgram is a reasonable representation of speech. More figures can be found in Appendix A.

3.4 Search on Posteriorgram

To compare the similarity between posteriorgrams, Dynamic Time Warping (DTW) is introduced to calculate pairwise warping distances. Consider the two posteriorgram sequences for a speech segment, $Q = \{\vec{q}_1, \ldots, \vec{q}_M\}$, and a speech segment, $S = \{\vec{s}_1, \ldots, \vec{s}_N\}$, where $\vec{q}_i$ and $\vec{s}_j$ are D-dimensional posterior probability vectors. The local distance between $\vec{q}_i$ and $\vec{s}_j$ can be defined by their inner product as $d(\vec{q}_i, \vec{s}_j) = -\log(\vec{q}_i \cdot \vec{s}_j)$. Since both $\vec{q}_i$ and $\vec{s}_j$ are probability vectors, the inner product gives the probability of these two vectors drawing from the same underlying distribution [46]. Other distance measures can be also used to compute the local distance between two posteriorgram vectors, while in this thesis, we focused on the inner product distance. Given a particular point-to-point alignment warp, $\phi = (\phi_q, \phi_s)$, of length K_ϕ between Q and S, the associated alignment score, $A_\phi(Q, S)$, is based on the sum of local distances

$$A_\phi(Q, S) = \sum_{k=1}^{K_\phi} d(\vec{q}_{\phi_q(k)}, \vec{s}_{\phi_s(k)})$$

where $1 \leq \phi_q(k) \leq M$ and $1 \leq \phi_s(k) \leq N$. The overall best alignment score, $\text{DTW}(Q, S) = \min_\phi A_\phi(Q, S)$.

If all possible warping paths, ϕ, are considered between Q and S, then there are $O(\text{MN})$ inner-product distances that will need to be computed. In order to eliminate unreasonable warping paths, a global path constraint is usually used to keep the warping paths between Q and S from being too far out of alignment [101]. This can

be accomplished, for example, by ensuring that $|\phi_q(k) - \phi_s(k)| \leq r$ so that the warp will keep local distances within r frames of each other along the entire alignment.

3.5 Spoken Term Discovery Using Gaussian Posteriorgrams

To further validate the effectiveness of a Gaussian posteriorgram as a robust feature representation of speech, we conduct a spoken term discovery task using Gaussian posteriorgrams. Although the original unsupervised acoustic pattern discovery work was effective in finding re-occurring instances of spoken words, it used whitened MFCCs as the acoustic representation to perform pattern matching [96]. This representation was effective for the task of academic lecture data, since the majority of the lecture was recorded from a single talker. However, the natural question to ask is how we can generalize this procedure to handle multiple talkers. In this section, we will replace the MFCC-based representation with the Gaussian posteriorgram representation of the speech signal and perform similar experiments on the multi-speaker TIMIT corpus and the single-speaker MIT Lecture corpus. The results from both show that Gaussian posteriorgrams outperform the previous whitened MFCC-based metho in spoken term discovery.

3.5.1 Spoken Term Discovery

Given a speech recording, if all speech frames are represented by Gaussian posteriorgrams, we can run Segmental Dynamic Time Warping (S-DTW) for the speech recording against itself to find re-occurring speech patterns such as frequently used words or short phrases. As mentioned in Chapter 2, S-DTW is a variant of the basic DTW algorithm. It computes multiple partial alignments between two utterances instead of a single end-to-end alignment. Specifically, after generating the Gaussian posteriorgrams for all speech utterances, S-DTW is performed on every utterance pair to find candidate co-occurring subsequences in the two utterances.

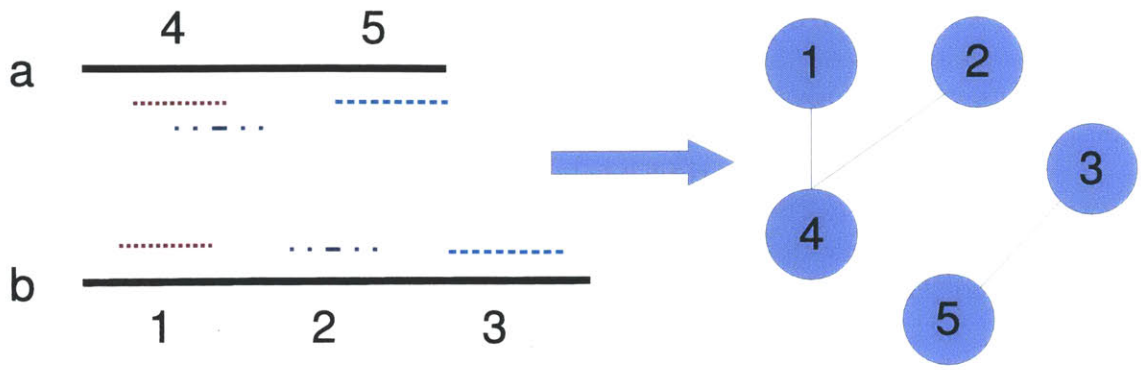


Figure 3-5: Converting all matched fragment pairs to a graph. Each numbered node corresponds to a temporal local maximum in fragment similarity in a particular utterance (e.g., 1-5). Each matching fragment is represented by a connection between two nodes in the graph (e.g., 1-4, 2-4, 3-5).

After collecting refined warping fragments for every pair of speech utterances, we try to cluster similar fragments. Since each warping fragment provides an alignment between two segments, if one of the two segments is a common speech pattern (i.e., a frequently used word), it should appear in multiple utterance pair fragments.

The basic idea is to cast this problem into a graph clustering framework, illustrated in Figure 3-5. Consider one pair of utterances in which S-DTW determines three matching fragments (illustrated in different colors and line styles). Each fragment corresponds to two segments in the two speech utterances, one per utterance. Since in general there could be many matching fragments with different start and end times covering every utterance, a simplification can be made to find local maxima of matching similarity in each utterance and to use these local maxima as the basis of nodes in the corresponding graph [98]. As a result, each node in the graph can represent one or more matching fragments in an utterance. Edges in the graph then correspond to fragments occurring between utterance pairs, and each associated weight corresponds to a normalized matching score. After the conversion, a graph clustering algorithm proposed by Newman [90] is used to discover groups of nodes (segments) in terms of graph distance. The role of the clustering algorithm is to decide which edges to group together, and which edges to eliminate. This latter point is especially important since it is possible to have partial overlapping matches ending

Table 3.1: Comparison of spoken term discovery performance using MFCCs and Gaussian posteriorgrams on the TIMIT corpus. MFCC represents the speaker dependent framework with default settings in [98]. GP stands for the Gaussian posteriorgram based method. Each cluster is given a purity score which represents the percent agreement of the underlying word label of each node in a cluster with the majority vote (e.g., a cluster with 2 nodes out of 4 with the same label would have a purity of 50%). We also calculated the number of speakers covered by the clusters and the gender ratio (Female/Male).

Method	# Clusters	Avg. Purity	# Speakers	F/M
MFCC	11	9.1%	457	0.42
GP	264	79.3%	408	0.43

at the same node that are unrelated to each other. The clustering output is a list of disjoint groups of nodes which represent the underlying speech fragments.

3.5.2 TIMIT Experiment

The TIMIT experiment was performed on a pool of 580 speakers (we combined the standard 462 speaker training set, and the larger 118 speaker test set). We excluded the dialect "sa" utterances since they were spoken by all speakers, and used the remaining 5 "sx" and 3 "si" utterances per speaker. A single GMM with 50 components was created from all the data using 13 dimensional MFCC feature vectors that were computed every 10ms. Since TIMIT consists of read speech in a quiet environment, the non-speech removal process was not applied.

The clustering result is shown in Table 3.1. The row labeled "MFCC" represents the speaker dependent framework with default settings used in [98]. "GP" stands for the Gaussian posteriorgram based method. Each cluster is given a purity score which represents the percent agreement of the underlying word label of each node in a cluster with the majority vote (e.g., a cluster with 2 nodes out of 4 with the same label would have a purity of 50%). From the table it is clear that the TIMIT task is very difficult for the original MFCC-based method due to the small number of utterances spoken by every talker, and the large number of talkers in the pool. The results did not change significantly when the clustering parameters were modified. Both the number of clusters that were automatically found, and the purity of these

clusters increased substantially with the posteriorgram-based representation. Within each cluster, on average nearly 80% of the nodes agree with the majority word identity of the cluster.

Since one of the properties we wished to explore was speaker variation, we also calculated the number of speakers covered by the clusters. The clusters determined using the Gaussian posteriorgram representation covered over 70% of the 580 speakers. Although the clusters obtained by the MFCC representation incorporated more speakers, the corresponding low purity score indicated that the clusters were fairly random. The gender ratio (Female/Male) of the entire corpus is 174/406=0.43, so it appeared that there was no obvious gender bias for the Gaussian posteriorgram method.

Table 3.2 shows the top 5 clusters ranked by increasing average distortion score. The transcription column represents the word identity of the cluster. These top 5 clusters all have a purity score of 100% and they are all from different speakers. Note that since we ranked the clusters by distortion, the cluster sizes are small, even though we observed that several clusters had the same underlying word identity. Since the goal of this work was to demonstrate that the Gaussian posteriorgram representation can solve the multi-speaker case which our earlier work could not handle, we leave the improvement of the clustering algorithm as future work. Another interesting point is that the top 5 clusters are identical for different parameter settings, which indicates that the phrases/words in each cluster are acoustically similar using the Gaussian posteriorgram representation. Since each "sx" sentence in TIMIT was said by seven talkers, we believe this contributed to the multi-word clusters that were observed, although the "si" data made up approximately 40% of the cluster data, which is the same proportion that it had in the overall corpus.

To better understand why using the Gaussian posteriorgram representation reduces the speaker variation issue, we calculated the alignment cost matrices of two speech segments of the word "organizations", produced by a male and a female talker, respectively, using the S-DTW search, illustrated in Figure 3-6. The top and bottom sub-figures correspond to the cost matrix generated by MFCC and the Gaussian

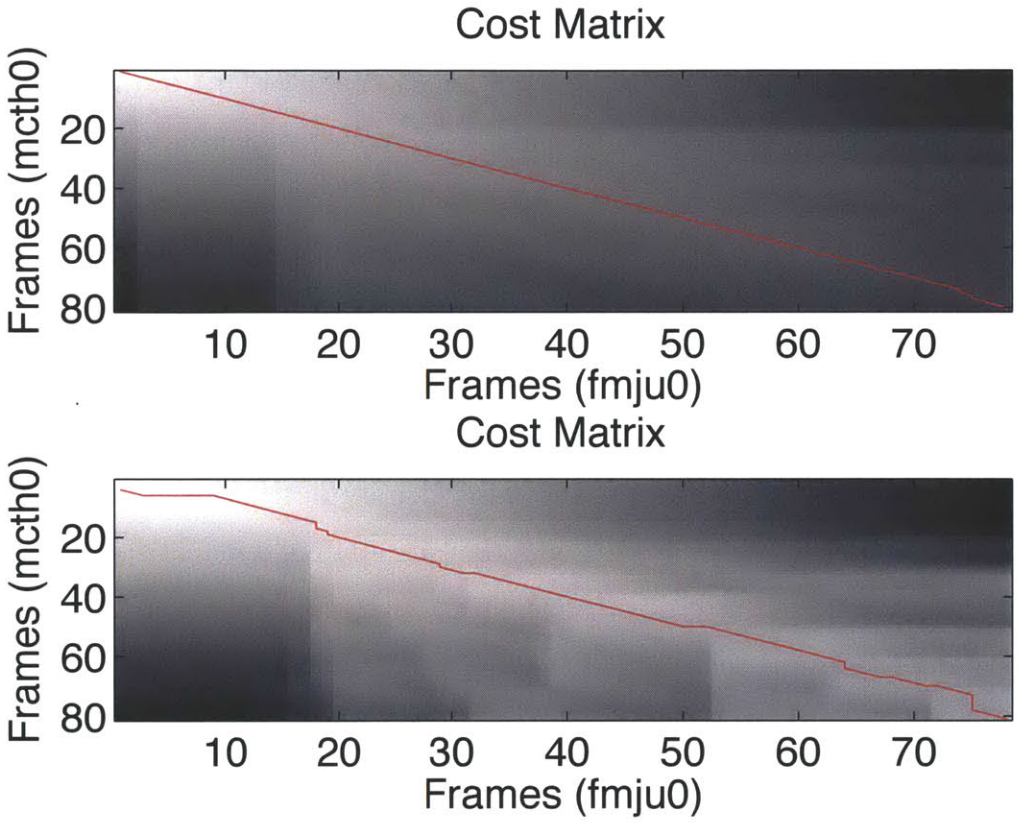


Figure 3-6: Cost matrix comparison for a male and female speech segment of the word "organizations". The top and bottom sub-figures correspond to the cost matrix generated by MFCC and the Gaussian posteriorgram representation, respectively. The cost values were normalized into a grey scale for the purposes of the figure. The lighter the pixel is, the more similar the corresponding two frames are. The red line in each sub-figure corresponds to the best-scoring alignment path. On the MFCC-based representation, the cost values around the warping path show no strong difference from the values away from the warping path, especially at the end of the warping path. On the Gaussian posteriorgram representation, there is a better delineation between the low-cost alignment path and the region around the warping path.

Table 3.2: Top 5 clusters on TIMIT found by Gaussian posteriorgram based spoken term discovery. The transcription column represents the word identity of the cluster. These top 5 clusters all have a purity score of 100% and they are all from different speakers.

ID	Cluster Size	Avg. Distortion	Transcription
1	2	0.87	shredded cheese
2	2	0.94	stylish shoes
3	3	1.02	each stag
4	3	1.06	the lack of heat
5	2	1.18	iris

posteriorgram representation, respectively. The cost values were normalized into a grey scale for the purposes of the figure. The lighter the pixel is, the more similar the corresponding two frames are. The red line in each sub-figure corresponds to the best-scoring alignment path. From the figures, it appears that on the MFCC-based representation, the cost values around the warping path show no strong difference from the values away from the warping path, especially at the end of the warping path. However, on the Gaussian posteriorgram representation, there is a better delineation between the low-cost alignment path and the region around the warping path. This observation suggests that the Gaussian posteriorgram representation is better at modeling phonetic similarities across talkers, and is thus better able to make distinctions between phones.

3.5.3 MIT Lecture Experiment

To further investigate the effect of using the Gaussian posteriorgram representation for unsupervised spoken term discovery, in this section, we present some results that directly compare the MFCC and Gaussian posteriorgram representations in a single speaker environment. Through a set of systematic experiments, we demonstrate that even in the single speaker case, spoken term discovery using the Gaussian posteriorgram representation still performs better than the whitened MFCC representation. As shown in Table 3.3, six lectures spoken by different talkers were used for these experiments. To generate Gaussian posteriorgrams, a single GMM with 50

components was created for each lecture using 13 dimensional MFCC feature vectors that were computed every 10ms. Prior to unsupervised GMM training, a speech detection module was used to remove non-speech from each lecture. Using our distributed computing environment with 200 CPUs, on average it took ten minutes to process one hour of speech data.

Table 3.3: Academic lectures used for spoken term discovery.

Lecture Topic	Duration
Economics	1 hr 15 mins
Speech Processing	1 hr 25 mins
Clustering	1 hr 18 mins
Speaker Adaptation	1 hr 14 mins
Physics	51 mins
Linear Algebra	47 mins

We compared our proposed Gaussian posteriorgram method with the original MFCC method in terms of number of clusters, average purity and term-frequency inverse-document-frequency (TFIDF) hits [102]. The results are shown in Table 3.4. In this table, GP denotes the Gaussian posteriorgram method. The TFIDF hit was calculated by observing how many words in the top 20 TFIDF word list also corresponded to discovered word clusters. The IDF score was computed from the 2,000 most common words in the Brown corpus [36].

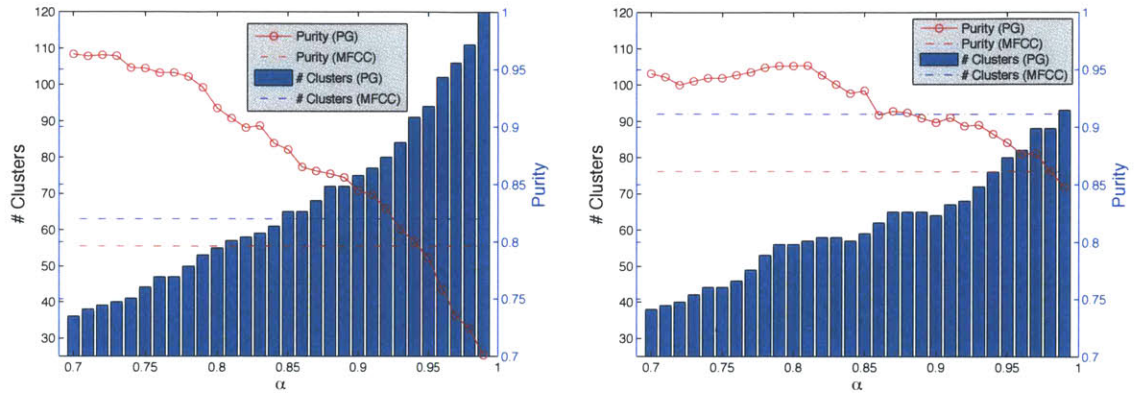
Table 3.4: Performance comparison of spoken term discovery in terms of # clusters found, average purity, and top 20 TFIDF hit rate. MFCC stands for the whitened MFCC based method in [98]. GP stands for the Gaussian posteriorgram based method. The TFIDF hit was calculated by observing how many words in the top 20 TFIDF word list also corresponded to discovered word clusters. The IDF score was computed from the 2,000 most common words in the Brown corpus [36].

Lecture	# Clusters		Avg. Purity(%)		TFIDF Hit(20)	
	MFCC	PG	MFCC	PG	MFCC	PG
Economics	63	65	79.6	**88.0**	11	**14**
Speech Processing	92	72	86.1	**90.2**	15	**19**
Clustering	87	68	93.2	**93.4**	16	**17**
Speaker Adaptation	63	66	91.9	**92.0**	13	**19**
Physics	51	58	89.5	**91.2**	17	**18**
Linear Algebra	41	25	94.0	**95.2**	**17**	16

From the results in Table 3.4, we can see that the Gaussian posteriorgram representation produces better clusters in terms of the average purity score and the number of TFIDF hits. The purity gain is larger on lectures that attained low purity scores using the MFCC method. We listened to the audio files and found these lectures contain more noise than other lectures, which may indicate that the Gaussian posteriorgram method is more robust to noisy acoustic condition than the MFCC method. On lectures with good MFCC purity scores, the Gaussian posteriorgram method often still produces a small improvement.

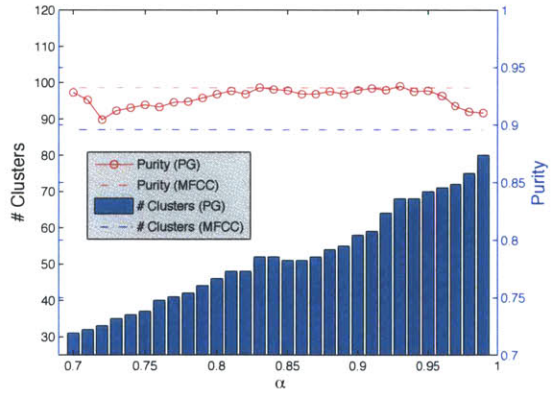
In terms of the TFIDF hit rate, the Gaussian posteriorgram representation can discover 85% percent of the top 20 TFIDF words while the MFCC method can find 75% percent [95]. We believe discovering more important words might lead to a better understanding of the content of an audio file. The only exception was the Algebra lecture where the Gaussian posteriorgram method found only 25 clusters. However, these 25 clusters still covered 16 words in the TFIDF list, which might also indicate that the Gaussian posteriorgram method tends to produce high quality clusters.

It is difficult to find a good evaluation criterion for the quality of posteriorgrams. In the current spoken term discovery work, we implicitly evaluate the quality of posteriorgrams by measuring the purity of the clusters found. However, the purity metric is not a direct measure of how well Gaussian posteriorgrams represent speech data in terms of speaker independence, noise robustness or other parameters as a result of post-processing. Figure 3-7 illustrates spoken term discovery results on four lectures with different clustering parameters. Although the S-DTW is performed on the same posteriorgrams, it is clear that cluster purity scores as well as the total number of found clusters are quite different. Therefore, one future challenge is to develop a general evaluation criterion which can directly assess the quality of generated posteriorgrams in a non task-specific setting, and preferably evaluate the quality in terms of speaker/language independence and noise robustness.

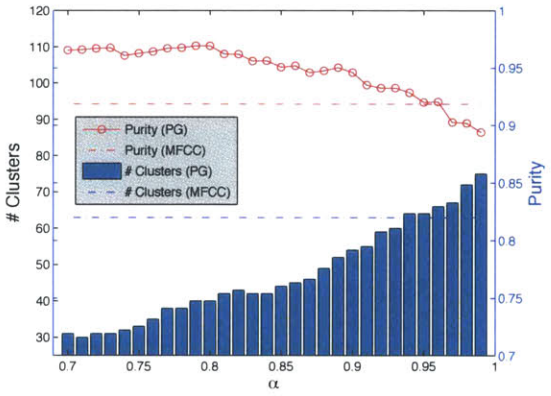


Lecture: Economics

Lecture: Speech Processing



Lecture: Clustering



Lecture: Speaker Adaptation

Figure 3-7: Effect of changing clustering stopping factor α on # clusters found and cluster purity on four MIT lectures. It is clear that different clustering parameters generate clusters with different purity scores as well as different number of found clusters although the S-DTW is performed on the same posteriorgrams.

3.6 Spoken Term Detection Using Gaussian Posteriorgrams

As mentioned in the previous section, the spoken term discovery task involves some post-processing steps, such as pruning and clustering, which make it difficult to evaluate the quality of Gaussian posteriorgrams. Therefore, in this section, we explore a slightly different task known as spoken term detection, which finds re-occurring speech patterns given one example of the pattern. We hope that, without requiring too many post-processing techniques, spoken term detection can help better evaluate the performance of Gaussian posteriorgrams.

Specifically, given a spoken term query and a test corpus, if all speech frames are represented by Gaussian posteriorgrams, we can run S-DTW for the spoken term query against each test utterance to find possible locations containing the input spoken term [146]. The system work flow is demonstrated in Figure 3-8. Note that the extra speech data box in the figure represents that since the training of GMM is unsupervised, more unlabeled speech data could be used to train the GMM and improve the quality of Gaussian posteriorgrams. After collecting all warping paths with their corresponding distortion scores for each test utterance, we simply choose the warping region with the minimum distortion score as the candidate region of the spoken term occurrence for that utterance. However, if multiple spoken term samples are provided and each sample provides a candidate region with a distortion score, we need a scoring strategy to calculate a final score for each test utterance that takes into account the contribution of all spoken term samples.

In contrast to the direct merging method used in [46], we considered the reliability of each warping region on the test utterance. Given multiple spoken term samples and a test utterance, a reliable warping region on the test utterance is the region where most of the minimum distortion warping paths of the spoken term samples are aligned. In this way a region with a smaller number of alignments to spoken term samples is considered to be less reliable than a region with a larger number of alignments. Therefore, for each test utterance, we only take into account the warping

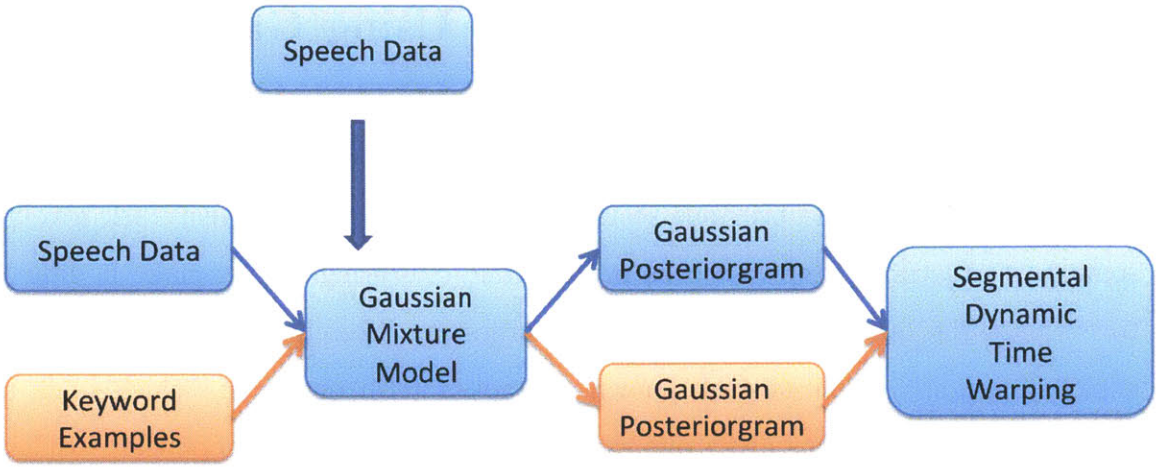


Figure 3-8: System work flow for spoken term detection using Gaussian posteriorgrams. Note that the extra speech data box in the top represents that since the training of GMM is unsupervised, more unlabeled speech data could be used to train the GMM and improve the quality of the Gaussian posteriorgrams generated.

paths pointing to a region with alignments to multiple spoken samples.

An efficient binary range tree is used to count the number of overlapped alignment regions on a test utterance. After counting, we consider all regions with only one spoken term sample alignment to be unreliable, and thus the corresponding distortion scores are discarded. We are then left with regions having two or more spoken term samples aligned. We then apply the same score fusion method proposed by Hazen et al. [46]. Formally, if we have $k \geq 2$ spoken term samples s_i aligned to a region r_j, the final distortion score for this region is:

$$S(r_j) = -\frac{1}{\alpha} \log \frac{1}{k} \sum_{i=1}^{k} \exp(-\alpha S(s_i)) \tag{3.2}$$

where varying α between 0 and 1 changes the averaging function from a geometric mean to an arithmetic mean. Note that since one test utterance may have several regions having more than two spoken term alignments, we choose the one with the smallest average distortion score. An extreme case is that some utterances may have no warping regions with more than one spoken term alignment (i.e., all regions are unreliable). In this case we simply set the distortion score to a very big value. After merging the scores, every test utterance should have a distortion score for the given

spoken term. We rank all the test utterances by their distortion scores and output the ranked list as the spoken term detection result.

We evaluated this idea on two different corpora. We initially used the TIMIT corpus for developing and testing the ideas we have described in the previous section. Once we were satisfied with the basic framework, we performed more thorough large vocabulary spoken term detection experiments on the MIT Lecture corpus [41]. The evaluation metrics that we report follow those suggested by [46]: 1) P@10 : the average precision for the top 10 hits; 2) P@N : the average precision of the top N hits, where N is equal to the number of occurrences of each spoken term in the test data; 3) EER : the average equal error rate at which the false acceptance rate is equal to the false rejection rate. Note that we define a putative hit to be correct if the system proposes a spoken term that occurs somewhere in an utterance transcript.

3.6.1 TIMIT Experiment

The TIMIT experiment was conducted on the standard 462 speaker training set of 3,696 utterances and the common 118 speaker test set of 944 utterances. The total size of the vocabulary was 5,851 words. Each utterance was segmented into a series of 25 ms frames with a 10 ms window shift (i.e., centi-second analysis); each frame was represented by 13 MFCCs. Since the TIMIT data consists of read speech in quiet environments, speech detection was unnecessary. All MFCC frames in the training set were used to train a GMM with 50 components. We then used the GMM to decode both training and test frames to produce a Gaussian posteriorgram representation. For testing, we randomly generated a 10-term set and made sure that they contained a variety of numbers of syllables. Table 3.5 shows the 10 spoken terms and their number of occurrences in both training and test sets (# training : # test). The best EER obtained is 16.4%.

Table 3.5: TIMIT 10 spoken term list with number of occurrences in training and test set (# training : # test).

age(3:8)	warm(10:5)	year(11:5)	money(19:9)
artists(7:6)	problem(22:13)	children(18:10)	surface(3:8)
development(9:8)	organizations(7:6)		

Table 3.6: MIT Lecture 30 spoken term list with number of occurrences in the training and test set (# training : # test).

zero (247:77)	space (663:32)	solutions (33:29)
examples (137:29)	performance (72:34)	matter (353:34)
molecule (28:35)	pretty (403:34)	results (121:35)
minus (103:78)	computer (397:43)	value (217:76))
situation (151:10)	therefore (149:46)	important (832:47)
parameters (21:50)	negative (50:50)	equation (98:61)
distance (58:56)	algorithm (35:36)	direction (214:37)
maximum (20:32)	responsible (92:10)	always (500:37)
likelihood (13:31)	mathematical (37:15)	never (495:21)
membrane (19:27)	problems (270:23)	course (847:76)

3.6.2 MIT Lecture Experiment

For the MIT Lecture experiments we used a standard training set containing 57,351 utterances and a test set with 7,375 utterances [41]. The vocabulary size of both the training and the test set is 27,431 words. Since the data were recorded in a classroom environment, there are many non-speech artifacts that occur such as background noise, filled pauses, laughter, etc. These non-speech data could cause serious problems in the unsupervised learning stage of our system. Therefore, prior to GMM training, we ran a speech detection module [40] to filter out non-speech segments. GMM learning was performed on frames within speech segments. Note that the speech detection module was trained independently from the Lecture data and did not require any transcription of the Lecture data. 30 spoken terms were randomly selected; all of them occur more than 10 times in both the training and test sets. All spoken terms occur less than 80 times in the test set to avoid using spoken terms that are too common in the data. Table 3.6 shows all the spoken terms and the number of their occurrences in the training and test sets.

Table 3.7: MIT Lecture spoken term experiment results when given different numbers of spoken term examples for the 30-word list. We fixed the smoothing factor α to be 0.0001, the S-DTW window size to 6 and the score weighting factor to 0.5. The evaluation metrics that we report follow those suggested by [46]: 1) P@10 : the average precision for the top 10 hits; 2) P@N : the average precision of the top N hits, where N is equal to the number of occurrences of each spoken term in the test data; 3) EER : the average equal error rate at which the false acceptance rate is equal to the false rejection rate.

# Examples	P@10	P@N	EER
1	27.0%	17.3%	27.0%
5	61.3%	33.0%	16.8%
10	68.3%	39.3%	15.8%

Table 3.8: Individual spoken term detection result ranked by EER on the MIT Lecture dataset for the 30-word list (%).

responsible (0.2)	direction (10.3)	matter (22.8)
situation (0.5)	parameters (10.5)	always (23.0)
molecule (4.9)	algorithm (11.3)	therefore (23.9)
mathematical (6.7)	course (11.4)	membrane (24.0)
maximum (7.5)	space (13.8)	equation (24.9)
solutions (8.1)	problems (17.8)	computer (25.3)
important (8.5)	negative (18.0)	minus (25.7)
performance (8.8)	value (19.4)	examples (27.0)
distance (9.0)	likelihood (19.4)	pretty (29.1)
results (9.3)	zero (22.7)	never (29.5)

Table 3.7 shows the spoken term detection performance when different numbers of spoken term samples are given. As a result of the TIMIT experiments, we fixed the smoothing factor α to be 0.0001, the S-DTW window size to 6 and the score weighting factor to 0.5. All three evaluation metrics improve dramatically from the case in which only one spoken term sample is given to the case in which five samples are given. Beyond five examples of a spoken term, the trend of the performance improvement slows. We believe the reason for this behavior is that the improvement from one sample to five samples is mainly caused by our voting based score merging strategy. When going from five samples to ten samples, we gain additional performance improvement, but there are always some difficult spoken term occurrences in the test data. Table 3.8 gives the list of 30 spoken terms ranked by EER in the

Table 3.9: MIT Lecture 60 spoken term list. All 60 spoken terms occur more than 10 times in the training set and less than 80 times in the test set.

iteration	probability	molecule	distortion	semester
matrix	information	several	derivative	properties
analysis	distance	factor	frequency	performance
solutions	important	elections	results	notations
increase	proteins	parameters	metric	space
specific	differential	whatever	second	criterion
thousand	theorem	similar	functions	negative
merge	together	often	general	usually
students	particular	reference	matter	computer
energy	equation	minutes	therefore	obviously
examples	lecture	amount	pretty	oxygen
lactose	smaller	omega	probably	phones

Table 3.10: MIT Lecture spoken term experiment results when given different numbers of spoken term examples for the 60-word list. The result showed similar characteristics with the results reported on the 30-word set. Given more spoken term examples, the system produced better detection results in terms of P@10, P@N and EER.

# Examples	P@10	P@N	EER
1	24.2%	14.7%	25.8%
5	36.3%	24.3%	18.2%
10	41.8%	27.6%	16.3%

10-example experiment. We observe that the words with more syllables tend to have better performance than ones with only two or three syllables.

In a separate experiment, another set of 60 spoken terms were randomly selected; similar to the 30-word set, all 60 spoken terms occur more than 10 times in the training set and less than 80 times in the test set. Table 3.9 shows the list of 60 spoken terms selected. Table 3.10 shows the spoken term detection performance when different numbers of spoken term examples are given. Table 3.11 gives the list of 60 spoken terms ranked by EER in the 10-example experiment. This result shows similar characteristics with the results reported on the 30-word set: words with more syllables tend to have better performance than ones with only two or three syllables.

Table 3.11: Individual spoken term detection result ranked by EER on the MIT Lecture dataset for the 60-word list (%). The result showed similar characteristics with the results reported on the 30-word set: words with more syllables tend to have better performance than ones with only two or three syllables.

iteration(3.0)	probability(3.7)	molecule(3.7)	distortion(4.6)	semester(4.8)
matrix(5.9)	information(5.9)	several(6.3)	derivative(6.3)	properties(6.7)
analysis(7.4)	distance(7.5)	factor(7.9)	frequency(8.0)	performance(8.1)
solutions(8.1)	important(8.5)	elections(9.0)	results(9.3)	notations(9.3)
increase(10.0)	proteins(10.3)	parameters(10.5)	metric(10.9)	space(11.8)
specific(13.0)	differential(13.6)	whatever(14.9)	second(16.3)	criterion(16.5)
thousand(17.4)	theorem(17.6)	similar(18.2)	functions(18.2)	negative(18.4)
merge(18.6)	together(19.0)	often(19.2)	general(19.2)	usually(20.5)
students(20.6)	particular(21.6)	reference(22.5)	matter(22.9)	computer(24.1)
energy(24.2)	equation(24.7)	minutes(26.1)	therefore(26.1)	obviously(26.3)
examples(26.6)	lecture(26.3)	amount(27.9)	pretty(28.5)	oxygen(28.6)
lactose(28.9)	smaller(29.0)	omega(29.3)	probably(30.3)	phones(35.1)

3.7 Summary

In this chapter, we presented the unsupervised Gaussian posteriorgram framework, which includes the generation of a Gaussian posteriorgram and its associated DTW-based search algorithm. A Gaussian mixture model is trained without using any supervised annotation, and represents each speech frame by calculating its posterior distribution over all Gaussian components. A modified DTW matching algorithm can be used to evaluate the similarity between two speech segments represented by Gaussian posteriorgrams in terms of an inner-product distance. The entire process is completely unsupervised and does not depend on speakers or languages.

We also presented two applications using Gaussian posteriorgrams. The results demonstrate the viability of using Gaussian posteriorgrams for both query-by-example based spoken term detection and speech pattern discovery in a multi-speaker environment.

Chapter 4

Resource Configurable DBN Posteriorgram

In the previous chapter, we presented the query-by-example spoken term detection (QE-STD) system based on Gaussian posteriorgrams. Although the Gaussian posteriorgram framework showed promising results in both unsupervised spoken term detection and spoken term discovery tasks, there are remaining problems requiring further investigation, such as how to achieve more robustness in representing speech, and how to fit in different training conditions. In this chapter, we describe a QE-STD framework based on posteriorgrams generated from Deep Belief Networks (DBNs). Through experiments, we show that the DBN-based posteriorgrams produce better spoken term detection performance, and are able to be adapted in unsupervised, semi-supervised and supervised training conditions. At the end of this chapter, we apply the DBN posteriorgrams to noisy speech data. A noise robust QE-STD system based on cross-entropy denoising DBN posteriorgrams is presented. Experiments show that the denoising DBN posteriorgrams on noisy speech can achieve almost the same spoken term detection performance on clean speech.

4.1 Introduction

In the previous chapter, we demonstrated an ability to perform spoken term detection without using a speech recognizer. By converting both queries and documents to a posterior probability-based representation called a Gaussian posteriorgram, a Segmental Dynamic Time Warping (S-DTW) algorithm [148] can be used to locate matches in speech documents. The Gaussian posteriorgram is a series of probability vectors computed on frame-based speech features such as MFCCs. Specifically, for each speech frame, a posteriorgram vector is generated by calculating the posterior probability of the MFCCs being generated by each component in a Gaussian mixture model (GMM). The GMM is trained on all MFCCs without requiring any labels.

There are two main drawbacks when using a GMM to generate posteriorgrams. First, it is clear that the quality of the posteriorgram representation is crucial to the spoken term detection performance. The current unsupervised GMM learning can only represent limited phonetic information about speech. It is well-known that speech has a hierarchical phonetic structure in terms of manner and place of articulation [101]. Thus, an explicit hierarchical modeling approach might lead to a better posteriorgram generation. Second, it is difficult for GMMs to incorporate partially annotated data to generate better posteriorgrams. In the previous chapter, we assumed that the data are unannotated, and spoken term detection was performed in a completely unsupervised setting. However, if a small portion of annotated data were available, it ideally would be easily incorporated to help produce better posteriorgrams.

To address these two main drawbacks, in this chapter, we propose a new class of posteriorgrams generated from Deep Belief Networks (DBNs). Compared with GMM-based posteriorgram generation, the DBN-based posteriorgrams have two advantages. First, the DBN explicitly models hierarchical structure using multiple layers of stochastic binary units. Second, DBN training can be performed in a semi-supervised setting [113]. Specifically, on the forward layer training stage, the DBN does not require any supervised information, and can use both labeled and unlabeled

data [114, 115]. On the backward fine-tuning stage, any amount of labeled data can be nicely embedded to improve modeling accuracy. In evaluation, we conduct several spoken term detection experiments. On the TIMIT dataset, in the semi-supervised setting, results show that 30% of labeled data are enough to obtain spoken term detection performance that is comparable to the case in which all labeled data are used. In the unsupervised setting, compared with the original Gaussian posteriorgram, using a GMM-seeded DBN posteriorgrams can improve the spoken term detection performance by 10% relative, in terms of the equal error rate (EER) [150]. On a Babel multi-lingual dataset, DBN posteriorgrams show similar performance gains compared to GMM posteriorgrams, which further demonstrates the DBN posteriorgram based QE-STD system is a language independent approach.

In the final section of this chapter, we apply the DBN posteriorgrams to noisy speech conditions. By refining noisy DBN posteriorgrams with the corresponding clean DBN posteriorgrams through back-propagation, on the noisy NTIMIT dataset, we are able to achieve almost the same spoken term detection performance as on the clean data.

4.2 Related Work

The use of artificial neural networks (ANNs) for modelling speech data has a long history. A variety of approaches have been explored since the late 1980s. We will review the two main architectures of using ANNs in speech acoustic modelling.

In late 1980s, several hybrid ANN-HMM speech recognition systems were developed [130]. The core idea among these approaches was to use ANNs to model the HMM's state emission distributions. The HMM output state posteriors are essentially the ANNs output distribution [47, 141]. Since these hybrid ANN-HMM systems can be trained efficiently using the Viterbi algorithm [13], and ANNs are inherently good at capturing nonlinear relationships in speech data, on several small but popular speech evaluation tasks, these systems showed promising results compared to the classic continuous density GMM-HMM structure [87, 12, 108].

A subsequent approach used the ANN output posterior distribution as an observation in the continuous density HMM acoustic modelling framework – the TANDEM approach [48]. The most successful TANDEM approach combined features from fixed window spectral analysis (such as MFCCs or PLPs [54]) with the posteriors from long-window articulatory feature detectors modelled by ANNs [48]. This TANDEM method addressed the issues brought by the conventional speech feature extraction method where only short-term spectral features are used, that ignore long-term signal analysis [151]. The TANDEM approaches have shown good speech recognition performance on languages with long-term acoustic characteristics such as tonal languages [131, 132, 135].

These earlier attempts of using ANNs in speech acoustic modeling have important limitations. Due to computer hardware constraints in the 1990s, ANNs used for acoustic modeling usually had only one or two hidden layers [105]. Researchers now believe that these structures were not deep enough to capture high-order nonlinear activities in speech [25]. Moreover, most early work only used context-independent phone labels for the ANN's output layer in order to reduce the time/memory cost for computing the output state posterior probability [13, 87]. Most importantly, by only using back-propagation to train the entire network discriminatively, the performance of ANNs highly depends on the initialization conditions [22]. A variety of different ANN initialization methods have been proposed in the literature, but none of them seem to generalize well across different datasets. Improper initialization often leads to overfitting after supervised back-propagation. Therefore, in many ANN-based systems, consistently obtaining a good ANN becomes a real challenge compared to the traditional GMM-HMM approach, which can be trained fairly well from a flat start using the Baum-Welch method [131].

Over the past two years, due to advances in ANN learning techniques, as well as the increase of processing capabilities of modern computer hardware, ANN-based acoustic modeling approaches have been revisited, and have achieved much success on a variety of speech tasks [84, 26, 19, 50]. The modern ANN-based approaches address the two main drawbacks of the previous ANN methods. First, the current ANN approaches

often use much deeper structures, such as five or six layers of hidden units, and a large output softmax layer (e.g. thousands of tri-phone states as output classes). This deep hidden structure can accommodate the rich nonlinear relationship in the speech data, while the large output softmax layer is well-matched to the modern tri-phone based context-dependent HMM architecture. Second, instead of initializing hidden weights with little guidance, the recent ANN-based approaches adopt a generative pre-training process [113]. The generative pre-training not only requires no supervised information, but can also put all hidden weights into a proper range which can be used to avoid local optima in the supervised back-propagation based fine-tuning.

In addition to the success of using deep ANN-HMMs for speech recognition, many researchers have reported different ways of using deep ANNs for speech related tasks. One class of methods is to use deep ANNs to generate features for speech systems similar to the TANDEM approach. The most successful approach is to use a deep ANN with a narrow bottleneck middle layer and to use the activations of the bottleneck hidden units as feature vectors [86, 121]. Recently, Sainath et al. explored using a deep bottleneck autoencoder to produce features for GMM-HMM and good recognition results were obtained [111]. Moreover, Vinyals et al. investigated the effectiveness of deep ANNs for detecting articulatory features. The detected articulatory events were combined with MFCC features for robust ASR tasks [136]. Lee et al. introduced the convolutional deep ANNs for audio classification tasks. The convolutional ANNs were designed to capture long-term temporal information and achieved good classification results [75]. Mikolov et al. explored using deep recurrent ANNs to train large scale language models [82]. Their recent results demonstrated that the deep ANN based language models are among the most successful techniques for statistical language modeling in both ASR and machine translation [81].

Although our DBN posteriorgram framework is similar to the idea of using ANNs output posteriors as feature vectors, it differs from the previous work in two key aspects. First, the DBN is used to directly model the speech data and generate posteriorgrams. Second, a generative pre-training stage is used to initialize all hidden weights in the DBN without using any labeled information. In the following sections,

a brief introduction of the DBN will be presented.

4.3 Deep Belief Networks

There are several papers with detailed description of Deep Belief Networks (DBNs). We briefly review some basic concepts in the following sections. More detailed descriptions can be found in [51, 50, 114, 113, 7].

4.3.1 DBN Definition

A DBN is an M-layer network of symmetrically coupled, stochastic binary units where M is usually greater than two [113, 114]. It contains a set of visible units $\mathbf{v} \in \{0,1\}^D$ and a sequence of layers of hidden units $\mathbf{h}_1 \in \{0,1\}^{L_1}$, $\mathbf{h}_2 \in \{0,1\}^{L_2}$,...,$\mathbf{h}_L \in \{0,1\}^{L_M}$, where D represents the input data dimension and L_i denotes the number of hidden units in each hidden layer. There are undirected connections only between hidden units in adjacent layers, as well as between the visible units and the hidden units in the first hidden layer. There are no within layer connections.

Consider learning a DBN with three hidden layers (i.e. $M = 3$), the energy of the joint configuration $\{\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3; \theta) = -\mathbf{v}^T \mathbf{W}_1 \mathbf{h}_1 - \mathbf{h}_1^T \mathbf{W}_2 \mathbf{h}_2 - \mathbf{h}_2^T \mathbf{W}_3 \mathbf{h}_3 \tag{4.1}$$

where $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$ are the model parameters, representing visible-to-hidden and hidden-to-hidden symmetric interaction terms. We omit bias terms for clarity of presentation. The probability of an input vector $\mathbf{v}$ is given by

$$P(\mathbf{v}; \theta) = \frac{1}{\mathbb{Z}(\theta)} \sum_{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3} \exp(-E(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3; \theta)) \tag{4.2}$$

where $\mathbb{Z}(\theta)$ is the normalization term defined as

$$\mathbb{Z}(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3} \exp(-E(\mathbf{v}, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3; \theta)) \tag{4.3}$$

On the top of the hidden layers, there is a softmax layer to convert the output of hidden units to class probabilities. For this 3-layer DBN, the conditional probability for the class c_j in the softmax layer can be written as

$$P(C = c_j|\mathbf{v}_3) = \frac{\left[\exp(\mathbf{v}_3^T\mathbf{W}_s + \mathbf{b}_s)\right]_j}{\sum_{i\in N}\left[\exp(\mathbf{v}_3^T\mathbf{W}_s + \mathbf{b}_s)\right]_i} \tag{4.4}$$

where $\mathbf{v}_3$ is the output from the third hidden layer, $\mathbf{W}_s$ and $\mathbf{b}_s$ are the weighting matrix and bias for the softmax layer, the sub-script in $[\cdots]_i$ denotes the i-th component of that vector and N represents the total number of classes.

4.3.2 DBN Inference

Exact maximum likelihood learning in this model is intractable, but efficient approximate learning of DBNs can be carried out by using a mean-field inference together with an Markov chain Monte Carlo (MCMC) based stochastic approximation procedure [113, 7]. Furthermore, the entire model can be efficiently pre-trained one layer at a time using a stack of modified Restricted Boltzmann machines (RBMs) [51].

Restricted Boltzmann Machines

An RBM is an undirected graphical model consisting of a layer of stochastic hidden binary variables and a layer of stochastic visible binary/real-valued variables shown in Figure 4-1. Within each RBM, connections only exist between hidden and visible variables while no hidden-hidden and visible-visible connections exist. In an energy-based representation, an RBM with binary visible variables can be written in the following form

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{4.5}$$

where $\mathbf{a}$ is the visible bias, $\mathbf{b}$ is the hidden bias and $\mathbf{W}$ is the visible-hidden weights [51]. When modeling a real-valued speech feature input such as MFCCs and PLPs, we use Gaussian-Bernoulli RBMs which are able to handle real-valued visible input [50]. The formal energy form is

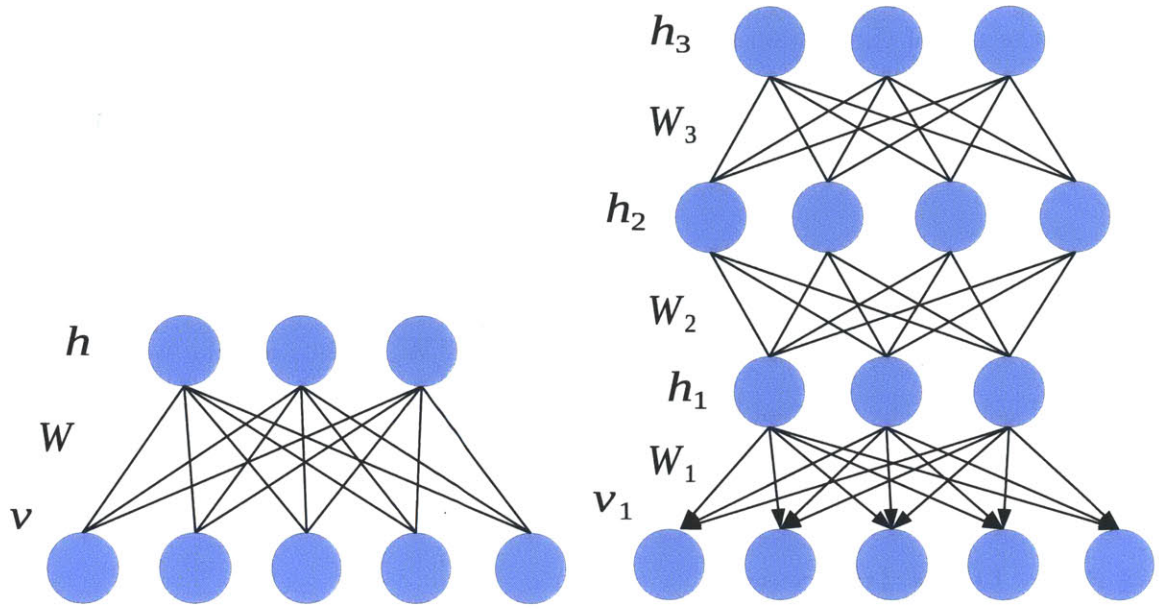


Figure 4-1: Left: A Restricted Boltzmann Machine (RBM). Right: A Deep Belief Network (DBN) with 3 hidden layers where each layer is an RBM. Within each RBM, connections only exist between hidden and visible variables while no hidden-hidden and visible-visible connections exist.

$$E(\mathbf{v}, \mathbf{h}) = \sum_{i \in \mathrm{V}} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \mathbf{b}^T \mathbf{h} - \sum_{i \in \mathrm{V}, j \in \mathrm{h}} \frac{v_i}{\sigma_i} h_j w_{ij} \tag{4.6}$$

where v_i denotes the i-th component of $\mathbf{v}$, h_j denotes the j-th component of $\mathbf{h}$, w_{ij} denotes one element in the weight matrix $\mathbf{W}$ and σ_i is the stand deviation of the visible unit v_i. Since there is no connection within the hidden and visible layer, the conditional probability $P(\mathbf{h}|\mathbf{v})$ can be factored given the visible units as follows.

$$P(\mathbf{h}|\mathbf{v}) = \prod_i P(h_i|\mathbf{v}) \tag{4.7}$$

Since h_i is binary, the per unit conditional probability can be written as

$$P(h_j = 1|\mathbf{v}) = \text{sigmoid}\left(b_j + \sum_{i \in \mathrm{V}} \frac{v_i}{\sigma_i} w_{ij}\right) \tag{4.8}$$

Using the same factorization trick, $P(\mathbf{v}|\mathbf{h})$ can be decomposed as

$$P(v_i = 1|\mathbf{h}) = \mathbb{N}\left(a_i + \sigma_i \sum_{j \in \mathrm{h}} h_j w_{ij}, \sigma_i^2\right) \tag{4.9}$$

where $\mathbb{N}(\cdot)$ is a Gaussian distribution. It is clear that if the input data is normalized to have unit variance, the energy form becomes

$$E(\mathbf{v}, \mathbf{h}) = \frac{1}{2}(\mathbf{v} - \mathbf{a})^T(\mathbf{v} - \mathbf{a}) - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{4.10}$$

and the corresponding conditional probabilities become:

$$P(h_j = 1|\mathbf{v}) = \text{sigmoid}\left(b_j + \sum_{i \in \mathrm{v}} v_i w_{ij}\right) \tag{4.11}$$

$$P(v_i = 1|\mathbf{h}) = \mathbb{N}\left(a_i + \sum_{j \in \mathrm{h}} h_j w_{ij}, 1\right) \tag{4.12}$$

The conditional probabilities for Bernoulli-Bernoulli RBMs are slightly different:

$$P(h_j = 1|\mathbf{v}) = \text{sigmoid}\left(b_j + \sum_{i \in \mathrm{v}} v_i w_{ij}\right) \tag{4.13}$$

$$P(v_i = 1|\mathbf{h}) = \text{sigmoid}\left(a_i + \sum_{j \in \mathrm{h}} h_j w_{ij}\right) \tag{4.14}$$

To construct a DBN accepting real-valued input features, Gaussian-Bernoulli RBM is used for the first hidden layers and the following layers are Bernoulli-Bernoulli RBMs.

Generative Pre-training of RBMs

To learn an RBM, maximum likelihood training is not feasible due to the large computational cost on computing the gradient over the log-probability of the entire data. However, Hinton et al. have proposed an efficient approximation learning algorithm called one-step contrastive divergence (CD-1) [51]. The CD-1 method is similar to a stochastic sampling process which repeatedly calculates the difference

between the input data and the data reconstructed from the RBM. The reconstruction difference approximates the true gradient and is used to update the weights. Formally, the CD-1 update rules for a Gaussian-Bernoulli RBM are

$$\Delta W_{ij} = E\left[\frac{v_i}{\sigma_i^2} h_j\right]_{\text{data}} - E\left[\frac{v_i}{\sigma_i^2} h_j\right]_{\text{1-reconstruction}} \tag{4.15}$$

$$\Delta a_i = E\left[\frac{v_i}{\sigma_i^2}\right]_{\text{data}} - E\left[\frac{v_i}{\sigma_i^2}\right]_{\text{1-reconstruction}} \tag{4.16}$$

$$\Delta b_j = E\left[h_j\right]_{\text{data}} - E\left[h_j\right]_{\text{1-reconstruction}} \tag{4.17}$$

where $E[\cdot]_{\text{data}}$ is the expected frequency with which the visible unit v_i and the hidden unit h_j are both "on" together. The "on" status of both visible and hidden units are determined by Eq. 4.8 and Eq. 4.9 directly from the training data. The second $E[\cdot]_{\text{1-reconstruction}}$ is the expected frequency with which the visible unit v_i and the hidden unit h_j are both "on". The difference is that the status of the visible unit v_i is determined by the one-step reconstructed data from the current model parameters [113]. Similar to the pre-training stage, if the input data is pre-normalized to have unit variance, we have $\sigma_i = 1$ for easy calculation. Stochastic gradient descent (SGD) is used to update the parameters [7]. Previous research suggests that momentum based SGD can be used to smooth the weight update [110]. Formally, the momentum based update rule can be written as

$$W_{ij}^{t+1} = mW_{ij}^{t} - \frac{\alpha}{B}\Delta W_{ij} \tag{4.18}$$

$$a_i^{t+1} = ma_i^{t} - \frac{\beta}{B}\Delta a_i \tag{4.19}$$

$$b_j^{t+1} = mb_j^{t} - \frac{\gamma}{B}\Delta b_j \tag{4.20}$$

where t is the step variable, α, β, γ are learning rates for $\mathbf{W}, \mathbf{a}, \mathbf{b}$, m denotes the

momentum and B is the batch size of the SGD.

After learning one RBM, the status of the learned hidden units given the training data can be used as feature vectors for the second RBM layer. The CD-1 method can be used to learn the second RBM in the same fashion. Then, the status of the hidden units of the second RBM can be used as the feature vectors for the third RBM, etc. In sum, the CD-1 method can be repeated to learn the subsequent RBMs given the status of the hidden units in the previous layer. By applying this greedy learning framework, multiple RBMs can be stacked together and pre-trained. An important property of pre-training is that parameter learning does not require any supervised information. Hierarchical structural information can be automatically extracted as an unsupervised density model to maximize the approximated data likelihood.

Back-propagation

If any amount of labelling information is given, a standard back-propagation algorithm [110, 51] for a multi-layer neural network can be applied to fine-tune the DBN model discriminatively [113]. The back-propagation can be implemented in an online update scheme, hence any future additional labels could be used in online fine-tuning. The goal of back-propagation fine-tuning is to maximize the log-probability of the entire dataset based on the cross-entropy criterion as follows:

$$F = \sum_{i=1}^{U} \log P(c|\mathbf{v}^i) \tag{4.21}$$

where U is the total number of training cases, v^i is the i-th training example, and c is the corresponding class label for v^i. Suppose we have a DBN with M hidden layers and a softmax layer with N units. By taking partial derivatives, the gradients for the weights are

$$\frac{\partial F}{\partial \mathbf{W}_l} = \sum_{i=1}^{U} \mathbf{v}_l^i \left[\mathbb{Z}_l(i)\mathbb{E}_l(i)\right]^T \tag{4.22}$$

and the gradients for the bias are

$$\frac{\partial F}{\partial \mathbf{b}_l} = \sum_{i=1}^{U} [\mathbb{Z}_l(i)\mathbb{E}_l(i)]^T \tag{4.23}$$

where $\mathbb{Z}$ is the transfer function and $\mathbb{E}$ is the error function. For the softmax layer, the transfer function $\mathbb{Z}_s$ can be written as

$$\mathbb{Z}_s(i) = \mathbf{1} \tag{4.24}$$

where $\mathbf{1}$ is a unit vector. The error function $\mathbb{E}_s$ can be written as

$$\mathbb{E}_s(i) = \delta_{c,i} - \frac{[\exp(\mathbf{W}_s \mathbf{v}_M^i + \mathbf{b}_s)]_c}{\sum_{j \in N} [\exp(\mathbf{W}_s \mathbf{v}_M^i + \mathbf{b}_s)]_j} \tag{4.25}$$

where $\mathbf{v}_M^i$ is the output of last hidden layer (the input of the softmax layer) and $\delta_{c,i}$ is the Kronecker function

$$\delta_{c,i} = \begin{cases} 1 & \text{if } c = i \\ 0 & \text{otherwise} \end{cases} \tag{4.26}$$

For the l-th hidden layer ($l \in [1..M]$), the transfer function $\mathbb{Z}_l$ can be written as

$$\mathbb{Z}_l(i) = (\mathbf{W}_l \mathbf{v}_l^i + \mathbf{b}_l) \cdot (\mathbf{1} - \mathbf{W}_l \mathbf{v}_l^i + \mathbf{b}_l) \tag{4.27}$$

which is known as the element-wise derivative of the i-th input. $\mathbf{v}_l^i$ represent the output of the l-th hidden layer for the i-th input. The error function $\mathbb{E}_l$ can be written as

$$\mathbb{E}_l(i) = \mathbf{W}_l \mathbb{Z}_l(i) \mathbb{E}_{l+1}(i) \tag{4.28}$$

where $\mathbb{E}_{l+1}$ is the errors back-propagated from the following hidden layer. Similar to the pre-training, a momentum based SGD approach is used to update $\mathbf{W}$ and $\mathbf{b}$ in the fine-tuning [49, 50].

$$(\mathbf{W}_l, \mathbf{b}_l)^{t+1} = m\,(\mathbf{W}_l, \mathbf{b}_l)^t + \frac{\alpha}{B} \cdot \frac{\partial F}{\partial\,(\mathbf{W}_l, \mathbf{b}_l)} \tag{4.29}$$

where t is the step variable, α is the learning rate, m is the momentum, and B is the batch size.

4.3.3 DBN Training in Practice

There are several papers about how to train a good DBN in practice [72, 49, 8]. We will summarize some techniques we used in our DBN training toolkit.

- Weight Initialization. For small tasks such as the TIMIT recognition task, visible-hidden weights are initialized to values drawn from a Gaussian distribution with zero mean and 0.01 standard deviation. For large datasets, we followed the method proposed in [8] in which each hidden unit is initialized to values drawn from a uniform distribution with range $[0, 1/\sqrt{\text{FanIn} + \text{FanOut}}]$. FanIn and FanOut denotes the number of incoming and outgoing connections of a hidden unit.

- Mini-batches. Since the SGD algorithm is used for both pre-training and back-propagation, mini-batches are used throughout the training process. The mini-batch size is set from 256 to 1024 according to the results in [19]. During the training, mini-batches are generated on the fly by randomly permutating all training samples after each iteration.

- Pre-training. Gaussian-Bernoulli RBMs often need more iterations and smaller learning rates than Bernoulli-Bernoulli RBMs [83]. A weight decay for visible-hidden weights is used to reduce the number of hidden units with very large weights [49].

- Back-propagation. We found that the momentum based update is a very important factor for efficient back-propagation. The momentum is set to 0.5 for the first few iterations and changed to 0.9 for the remaining iterations. A good way of preventing overfitting is to monitor the cross-entropy on a held-out set. At the end of each iteration, if the cross-entropy on the held-out set increases,

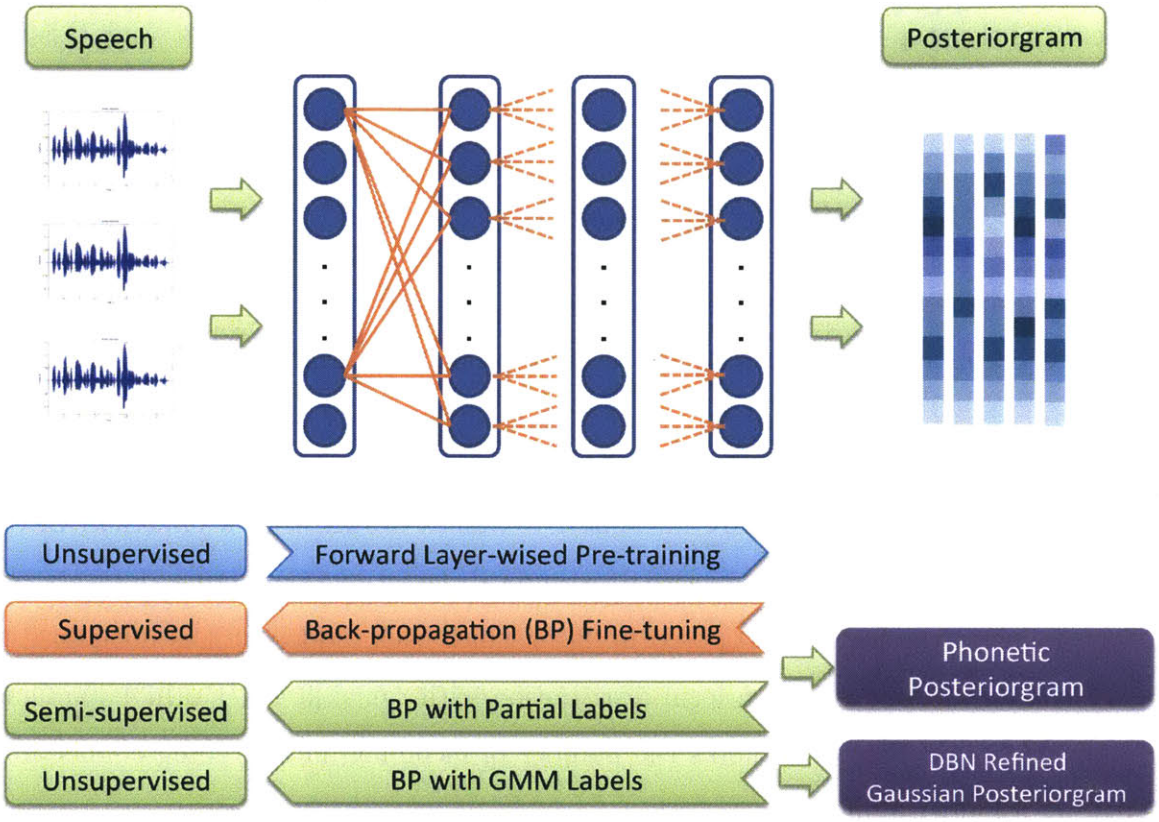


Figure 4-2: System work flow for generating posteriorgrams using DBN in both semi-supervised and unsupervised configurations.

the weights are set to their original values before the iteration and the learning rate is halved [83].

4.4 DBN Posteriorgrams

In this section, we present two DBN posteriorgrams which can be used in the semi-supervised and unsupervised training conditions respectively. Figure 4-2 illustrates a summary of generating posteriorgrams using DBN in both semi-supervised and unsupervised configurations. Details will be presented in the following sub-sections.

4.4.1 Semi-supervised DBN Phonetic Posteriorgram

Like the supervised phonetic posteriorgrams [46, 68] discussed in Section 3.1, a supervised or semi-supervised DBN posteriorgram is a probability vector representing

the posterior probabilities of a set of labeled phonetic units for a speech frame. Formally, if we denotes N speech frames as $\vec{x}_1, \ldots, \vec{x}_N$ and their corresponding phonetic labels $ph_1, \ldots, ph_N$, a posterior probability, $p_i^j = P(ph_j|\vec{x}_i;\theta)$, can then be calculated for any speech frame, $\vec{x}_i$, for each phonetic label ph_j, given DBN model parameters θ and using a softmax activation function. If there are V phonetic labels, a speech frame $\vec{x}_i$ can then be represented by a V-dimensional probability vector, $\vec{p}_i = \{p_i^1, \ldots, p_i^V\}$, where $\sum_j p_i^j = 1 \quad \forall i$.

Compared with the Gaussian posteriorgrams which can be generated by a GMM trained without any supervised information, DBN posteriorgrams require some annotated data for training. In the semi-supervised training procedure we use in this work, we first train the DBN model using all data without labels (i.e., unsupervised), followed by the fine-tuning step that requires some amount of labeled data.

4.4.2 Unsupervised DBN Refined Gaussian Posteriorgram

In machine learning, a weak classifier can be used to initialize a strong classifier to accelerate the training process. For example, in conventional Expectation-Maximization (EM) training of a GMM, K-means clustering is often used to initialize the target GMM. Inspired by this idea, we investigate a fully unsupervised DBN posteriorgram by training a DBN from labels generated from an unsupervised GMM. Given a set of N speech frames with an MFCC representation, $\vec{x}_1, \ldots, \vec{x}_N$, a D-mixture GMM G is trained on all frames without using any labels. For each frame $\vec{x}_i$, we provide a labeler function L as

$$L(\vec{x}_i) = \arg\max_j P(g_j|\vec{x}_i) \tag{4.30}$$

where g_j is the j-th Gaussian component in G. In other words, each speech frame is labeled by the index of the Gaussian component which maximizes the posterior probability given $\vec{x}_i$. Then a DBN is trained on those "artificial" labels. This DBN posteriorgram generation is similar to the semi-supervised case except that the human produced phonetic label ph_j for each frame is replaced by the GMM produced "arti-

ficial" label j. Through this two-stage training process, we leverage the DBN's rich model structure to produce better posteriorgrams than a GMM, while still keeping the entire training framework compatible with the unsupervised setting.

4.4.3 Evaluation

We performed four different evaluations of the DBN based posteriorgram representation. In the first evaluation, we investigated how different layer configurations of the DBN would affect the quality of the generated posteriorgram as well as the spoken term detection performance. The DBN for this experiment was trained in a fully supervised setting. In the second evaluation, we examined how spoken term detection performance is affected when using partially labeled data for DBN training. In the third evaluation, we compared the spoken detection performance of the fully unsupervised DBN posteriorgram with our previous Gaussian posteriorgram baseline. In the fourth evaluation, we compare the Gaussian posteriorgram base QE-STD with the DBN posteriorgram based QE-STD on the Babel Cantonese dataset.

TIMIT Supervised Results

The spoken term detection task performed on the TIMIT corpus used the same setting as in Section 3.6.1. The spoken term list was the same as the previous experiment. Performance was measured by the average equal error rate (EER): the average rate at which the false acceptance rate is equal to the false rejection rate. Besides the EER, we also used the Maximum Term-Weighted Value (MTWV) measure provided by NIST from the 2006 STD contest [34]. MTWV is a biased average between miss detection rate and false alarm rate [100]. For each spoken term query, an individual MTWV score is computed first and the final MTWV score is averaged over all spoken term queries. Formally, the MTWV can be written as

$$\text{MTWV} = \frac{1}{Q} \sum_{q} \max_{\theta} (1 - (P_{miss}(q, \theta) + \beta P_{fa}(q, \theta))) \tag{4.31}$$

where $P_{miss}(q, \theta)$ and $P_{fa}(q, \theta)$ are the miss detection and false alarm rate given the

operating point θ for query q, $\beta = 1000$ is the prior knowledge factor used to bias the average and Q is the total number of spoken term queries. The miss detection and false alarm rate for the query q are defined as follows:

$$P_{miss}(q, \theta) = 1 - \frac{N_{corr}(q)}{N_{true}(q)} \tag{4.32}$$

$$P_{fa}(q, \theta) = \frac{N_{fa}(q)}{T - N_{true}(q)} \tag{4.33}$$

where for the query q, $N_{corr}(q)$ is the number of correct detections, $N_{fa}(q)$ is the number of incorrect detections, $N_{true}(q)$ is the number of true occurrences and T is the total amount of speech (in seconds) in the test test.

In the supervised experiments, we used all labeled training data (3,696 utterances) in order to maximize the DBN posteriorgram performance while changing different DBN layer configurations. For DBN training, each training utterance was segmented into a series of 25ms windowed frames with a 10ms shift (i.e., centisecond analysis). Each frame was represented by 39 dimensional MFCC stacked with the neighboring 10 frames (5 on each side). In total, the feature dimension for each frame is 429 (39 x 11). All 61 phonetic labels were used for training. After training, each frame in the training and test set was decoded by the DBN, producing a posteriorgram vector of 61 dimensions. Spoken term detection was done by comparing the keyword example posteriorgrams with the test set posteriorgrams using the DTW method described in Section 3.4.

Table 4.1 presents the results for different DBN configurations and their resulting average EER and MTWV. In the first column, 500 indicates a DBN that has one hidden layer with 500 hidden units, while 500x500 denotes a DBN with two hidden layers each of which has 500 hidden units. All DBN configurations have a softmax output layer with 61 units. In initialization, the visible-hidden weights in each DBN were randomly drawn from a Gaussian distribution with zero mean and 0.01 standard deviation. The visible and hidden bias weights were set to zero. To cope with real valued input for the stacked MFCC input, the first hidden layer in all DBNs is set

Table 4.1: Average ERR for different DBN layer configurations for supervised DBN posteriorgram based QE-STD on the TIMIT corpus. In the first column, 500 indicates a DBN that has one hidden layer with 500 hidden units, while 500x500 denotes a DBN with two hidden layers each of which has 500 hidden units. All DBN configurations have a softmax output layer with 61 units.

DBNs	Avg. EER(%)	MTWV
500	10.6	0.557
300x300	10.3	0.553
500x500	9.8	0.543
1000x1000	10.4	0.556
500x500x500	10.1	0.546

to a Gaussian-Bernoulli RBM and the remaining hidden layers are set to Bernoulli-Bernoulli RBMs. The forward layer training in each configuration was set to stop at the 100^{th} iteration with a learning rate of 0.004 and a batch size of 256. The fine-tuning using back-propagation was set to stop at the 50^{th} iteration using the line search stochastic gradient decent method with a batch size of 256.

The results indicate that spoken term detection performance was not overly sensitive to DBNs with different layer settings. For both EER and MTWV, the difference among five configurations is less than 1%. This implies that we need not be overly concerned about the DBN layer configurations in subsequent experiments.

Semi-supervised Results

In the second experiment, we used a two-layer DBN with 500 hidden units for each layer and a softmax output layer with 61 units. The DBN initialization and learning parameters were the same as the supervised experiments. We first trained our model on all 3,696 unlabeled utterances, followed by the fine-tuning stage that only used partially labeled data. Figure 4-3 demonstrates the results. On the x-axis, a training ratio of 0.1 indicates that only 10% of the labeled data were used in the fine-tuning stage, while a training ratio of 1.0 means all labeled data were used. It can be observed that the average EER curve drops dramatically from 0.01 to 0.2 and becomes steady between 0.3 to 0.8. This is an interesting result because in scenarios where fully labeled data are not cost effective to obtain, 20% to 30% of labeled data

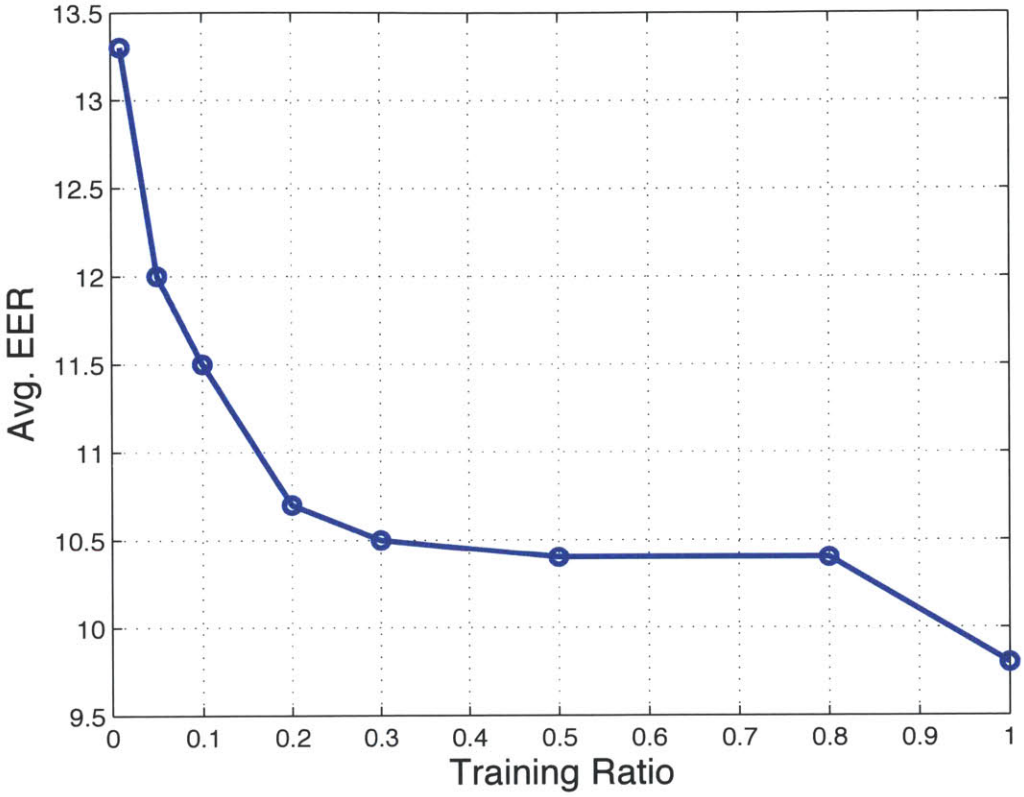


Figure 4-3: Average EER against different training ratios for semi-supervised DBN posteriorgram based QE-STD on the TIMIT corpus. On the x-axis, a training ratio of 0.1 indicates that only 10% of the labeled data were used in the fine-tuning stage, while a training ratio of 1.0 means all labeled data were used. It can be observed that the average EER curve drops dramatically from 0.01 to 0.2 and becomes steady between 0.3 to 0.8.

are enough to produce a system that is only slightly worse than the system trained on all labeled data. Moreover, since the back-propagation algorithm has to go through each data point for each iteration in the fine-tuning step, using a smaller portion of labeled data also saves a significant amount of training time.

TIMIT Unsupervised Results

In the unsupervised training experiment, a 500x500 DBN was trained by using labels generated from a GMM with 61 Gaussian mixtures. Specifically, a GMM was first trained on frame-based MFCCs without using any labels. To be consistent with

our prior work, only 13 MFCCs per frame were used to train the GMM. Once the unsupervised GMM had been created, each frame was subsequently labeled by the most likely GMM component (Eq. 4.30). A DBN was then trained on 429 MFCCs per frame using the GMM-generated labels. We then compared the unsupervised posteriorgram detection performance between the GMM and the DBN-based posteriorgrams, as shown in Table 4.2. As we have reported in Section 3.6.1, the Gaussian posteriorgrams produced an average EER of 16.4% and an MTWV score of 0.229. The unsupervised DBN-based posteriorgrams improved upon this result by over 10% to achieve an average EER of 14.7% and an MTWV score of 0.281. We believe the improvement is due to the DBN's explicit hierarchical model structure that provides a finer-grained posterior representation of potential phonetic units than those that can be obtained by the Gaussian posteriorgram. Note that in an attempt to make a comparison using the same larger feature set, we also trained an unsupervised GMM using the 429 dimensional MFCC vectors that were used to train the DBN. In this case, however, the average EER degraded to over 60%, which we attribute to a weaker ability of the GMM to cope with higher dimensional spaces.

The third row in Table 4.2, highlights one final advantage of the DBN framework in that it is able to incorporate partially labeled data. When we included only 1% of labeled data, we see that the average EER is further reduced to 13.3% (as also shown in the first data point in Figure 4-3). This reduction corresponds to another 9.5% performance gain over the unsupervised case. The MTWV score shows a significant improvement from 0.281 to 0.441. We think the reason is that by introducing supervised information, the false alarm rate can be significantly reduced so that the heavily false alarm rate biased MTWV score improves. Figure 4-4 illustrates Table 4.2 and also shows the average Detection Error Tradeoff (DET) curves for each row in Table 4.2. The false positive rate on the x axis is in log-scale. In the legend, GP stands for Gaussian posteriorgrams. Numbers in parentheses are MTWV scores. We can see that DET curves for semi-supervised and supervised configurations are clearly below the DET curves for unsupervised configurations.

Table 4.2: Comparison of Gaussian and DBN posteriorgram based QE-STD on the TIMIT corpus. The Gaussian posteriorgrams produced an average EER of 16.4% and an MTWV score of 0.229. The unsupervised DBN-based posteriorgrams improved upon this result by over 10% to achieve an average EER of 14.7% and an MTWV score of 0.281. When including 1%, 30% and 100% of labeled data, the average EER is further reduced to 13.3%, 10.5% and 9.5% respectively (as also shown in the first data point in Figure 4-3). The MTWV scores show similar improvement from 0.441 to 0.543.

Posteriorgram	Avg. EER (%)	MTWV
Gaussian	16.4	0.229
Unsupervised DBN	14.7	0.281
DBN (1%)	13.3	0.441
DBN (30%)	10.5	0.526
DBN (100%)	9.8	0.543

Table 4.3: Babel Cantonese 30 spoken term list. The spoken term selection rule makes sure that for each term there are at least 10 occurrences in the training set and less than 100 occurrences in the test set, similar to the configuration we used in the TIMIT experiment.

而家	果	好似	即系	真系	如果
觉得	但系	几多	跟住	果边	我地
打电话	可以	睇下	老公	自己	因为
屋企	之前	听日	时间	星期六	一直
咩野	之后	其实	结婚	电话	到时

Babel Cantonese Results

To demonstrate the language independent aspect of the proposed DBN posteriorgram spoken term detection framework, we applied our system to a large vocabulary Cantonese speech dataset. A set of 30 spoken terms was selected from the provided keyword list and 10 examples of each term were extracted from the training set, shown in Table 4.3. The spoken term selection criterion were that each term occurred at least 10 times in the training set, and less than 100 times in the test set (similar to the configuration we used in the TIMIT experiment). For each spoken term, the spoken term detection task was to rank all test utterances according to the utterance's possibility of containing that term. As in the TIMIT experiment, performance was measured by the EER and the MTWV score.

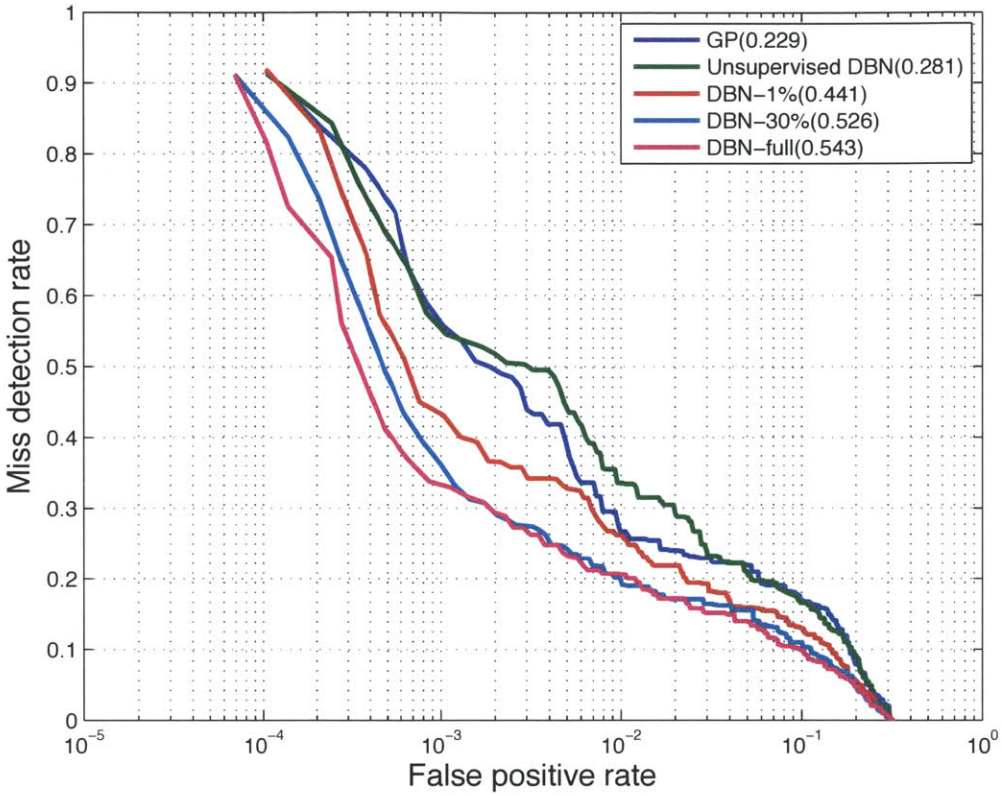


Figure 4-4: DET curve comparison of Gaussian and DBN posteriorgram based QE-STD on the TIMIT corpus. The false positive rate on the x axis is in log-scale. In the legend, GP stands for Gaussian posteriorgrams. Numbers in parentheses are MTWV scores.

Table 4.4 shows the comparison of QE-STD using Gaussian and DBN posteriorgrams on the Babel Cantonese corpus. In the table, Gaussian-i denotes using i example(s) for each term to perform spoken term detection on Gaussian posteriorgrams. To generate Gaussian posteriorgrams, a GMM with 50 components was trained on all the training data. Compared to the TIMIT English dataset, a similar EER decrease and MTWV increase can be seen if increasing the number of spoken term examples used for detection. The Gaussian-10 configuration showed an EER of 21.4% which is higher than the EER of 16.4% on the TIMIT English dataset using the same configuration. MTWV scores are low due to high false alarm rates. We think the reason is that the Babel task is harder than the TIMIT task. The TIMIT dataset is clean read speech recorded with 16kHz sampling rate, while the Babel dataset is

Table 4.4: Comparison of Gaussian and DBN posteriorgram based QE-STD on the Babel Cantonese corpus. Gaussian-i denotes using i example(s) for each term to perform spoken term detection on Gaussian posteriorgrams. Compared to the TIMIT English dataset, a similar EER decrease and MTWV increase can be seen if increasing the number of spoken term examples used for detection. MTWV scores are low due to high false alarm rates.

Posteriorgram	Avg. EER	MTWV
Gaussian-1	33.8%	0.008
Gaussian-5	23.1%	0.089
Gaussian-10	21.4%	0.098
Unsupervised DBN	19.9%	0.099

spontaneous speech recorded through telephone and cell-phone lines with an 8kHz sampling rate.

The DBN posteriorgrams in Table 4.4 were generated from an unsupervised DBN with 5 hidden layers and a softmax layer with 50 units. Each hidden layer contains 2048 hidden units. The pre-training was done on the entire training set and the back-propagation was done by using GMM generated labels. The DBN posteriorgrams outperformed the Gaussian posteriorgrams when using 10 examples for each term. However, the EER and MTWV gain here is smaller than the previous TIMIT experiment. We think the reason is also due to the noisy conditions in the Babel corpus, making it harder for DBN to generate high quality posteriorgrams.

To further demonstrate the performance of the proposed QE-STD system, we used a 606 spoken term list provided by NIST to run a spoken term detection experiment on the standard Babel Cantonese test set. An unsupervised DBN with 5 hidden layers (2048 hidden units each) and a softmax layer with 50 units was trained to generate posteriorgrams. All spoken term examples in the training set were used. The MTWV score obtained was 0.206 compared to an MTWV score of 0.45 from a fully supervised STD system.

4.5 Denoising DBN Posteriorgrams

As we have seen in the Babel Cantonese QE-STD experiment, when the speech data is noisy, the gain from using DBN posteriorgrams become smaller compared to the gain on clean speech. To overcome this problem, in this section, the DBN posteriorgrams are extended to noisy speech conditions. By assuming that parallel clean and noisy speech recordings are available for DBN training, we demonstrate that denoising DBN posteriorgrams can significantly improve the QE-STD performance on noisy speech data.

4.5.1 System Design

The classic back-propagation for fine-tuning a DBN is to use 0/1 as error indicators for class labels shown in Eq. 4.25. The goal is to maximize the log-probability of the entire dataset given the correct class labels. Considering all class labels for an input feature vector, the objective function in Eq. 4.21 can be rewritten as

$$F = \sum_{i=1}^{U} \sum_{j=1}^{N} \delta_{C(i),j} \log P(j|\mathbf{v}^i) \tag{4.34}$$

where N is the total number of classes and $\delta_{C(i),j}$ is the Kronecker function as

$$\delta_{C(i),j} = \begin{cases} 1 & \text{if } C(i) = j \\ 0 & \text{otherwise} \end{cases} \tag{4.35}$$

The classic back-propagation algorithm based on the cross-entropy criterion only collects log-probabilities when the prediction $C(i)$ is the same as the label j. From another point of view, $\delta_{C(i),j}$ can be considered as a probability distribution which assigns probability 1 to the correct label for the i-th input and probability 0 elsewhere. The back-propagation fine-tuning essentially pushes $\log P(j|\mathbf{v}^i)$ towards the δ distribution by maximizing the total log-probability. If we change the δ from a 0/1 distribution to a continuous distribution γ, by running back-propagation, the DBN model can be used to approximate the continuous distribution γ, where in our case,

the continuous distribution γ is a posteriorgram. Formally, the new objective function can be written as

$$F = \sum_{i=1}^{U} \sum_{j=1}^{N} \gamma_{C(i),j} \log P(j|\mathbf{v}^i) \tag{4.36}$$

where γ is a continuous distribution which assigns probabilities across N dimensions.

By using the new objective function in Eq. 4.36, the DBN can be trained on noisy speech data but fine-tuned by clean posteriorgrams. The system work flow is demonstrated in Figure 4-5. Specifically, with parallel clean and noisy speech data available, a clean DBN can be pre-trained on the clean speech data and fine-tuned by the GMM generated labels (unsupervised configuration). Then, the second DBN is pre-trained on the noisy speech data. For each noisy speech utterance, the second DBN is fine-tuned by the posteriorgrams generated from the first clean DBN on the same parallel clean speech utterance, by using the new objective function in Eq. 4.36. Therefore, after the back-propagation fine-tuning, the second DBN takes noisy speech as input and approximates the posteriorgrams generated by the clean DBN on the corresponding clean speech. We hope that the rich nonlinear structure in the DBN can be used to learn an efficient transfer function which removes noise in speech while keeping enough phonetically discriminative information to generate good posteriorgrams.

The limitation of the denoising DBN posteriorgram framework is that speech data must be presented in both noisy and clean conditions. Oftentimes, a speech corpus is recorded in only one condition, either noisy or clean. Given only a clean speech corpus, this framework might still work since noise can be added into the clean speech so that the parallel noisy and clean speech corpus can be created artificially. Given only a noisy speech corpus, there are two possible directions to explore. One direction is to use noise reduction algorithms to create the parallel corpus for DBN training. Another possibility is to pre-train the denoising DBN with all noisy speech data but only fine-tune it with available parallel corpora. In the latter case, the pre-training stage can be viewed as an unsupervised noise-independent learning process, while

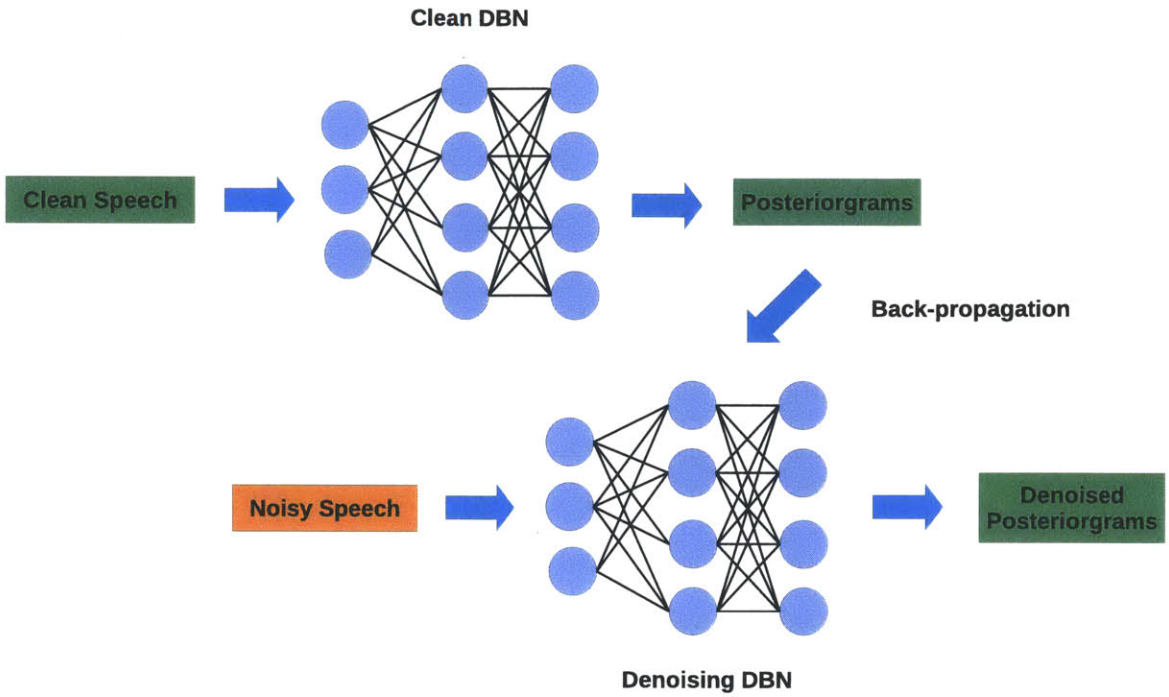


Figure 4-5: System work flow for training a denoising DBN. A clean DBN is trained on the clean speech and is used to generate clean posteriorgrams. The DBN trained on the noisy speech is fined-tuned with the clean posteriorgrams so that for future data, the denoising DBN can accept noisy speech input and output denoised posteriorgrams.

the semi-supervised fine-tuning stage can be expected to learn how to remove unseen noise by only using partial labeled information.

4.5.2 Evaluation

To evaluate the QE-STD performance, the NTIMIT speech corpus was used for the experiment. The NTIMIT speech corpus is the noisy version of the original clean TIMIT speech corpus. Specifically, for each clean TIMIT utterance, a noisy version was recorded from broadcasting the clean utterance through a telephone line. As a result, channel noise was added in each TIMIT utterance for the NTIMIT corpus. Moreover, the recording sampling rate of NTIMIT was lowered to 8kHz due to the telephone signal transmission. The training set and test set used were the same as the original TIMIT dataset. To compare with the previous TIMIT experiment, the same 10 spoken terms and the same 10 examples of each term were extracted from

Table 4.5: Comparison of Gaussian and DBN posteriorgram based QE-STD on the NTIMIT and TIMIT corpus.

Posteriorgram	Avg. EER	MTWV
Gaussian	19.9%	0.006
Unsupervised DBN	18.2%	0.04
Denoising DBN	16.2%	0.232
Gaussian-TIMIT	16.4%	0.229
Unsupervised DBN-TIMIT	14.7%	0.281

the training set. As before, performance was measured by the average EER and the MTWV score.

In order to build a baseline system, Gaussian posteriorgrams and unsupervised DBN posteriorgrams were generated for the NTIMIT dataset. Specifically, to generate Gaussian posteriorgrams, a GMM with 50 components was trained on all the training data. The input feature was 13 MFCCs with 25ms window length and 10ms window shift exactly as what we did for the TIMIT dataset. For the unsupervised DBN posteriorgrams, to be consistent with the previous TIMIT experiment, a DBN with two hidden layers and a softmax layer was pre-trained on 429 dimensional stacked MFCC features (39 x 11). Each hidden layer consists of 512 hidden units and the softmax layer has 50 units. The DBN was then fine-tuned using labels generated from the 50-component GMM.

To generate denoising DBN posteriorgrams, we took the unsupervised DBN posteriorgrams from the clean TIMIT data and used them to perform frame-wise back-propagation on the DBN pre-trained on the noisy NTIMIT data. After the back-propagation was done, unseen test utterances from the NTIMIT corpus were decoded by the denoising DBN.

Table 4.5 shows the results on the NTIMIT dataset with different posteriorgrams. Due to the channel noise and low sampling rate of the NTIMIT speech, the baseline QE-STD based on Gaussian posteriorgrams and unsupervised DBN posteriorgrams are both worse than the performance on the clean TIMIT corpus in terms of EER and MTWV. As for the Babel Cantonese experiment, the EER gain for using unsupervised DBN posteriorgrams is smaller on the noisy NTIMIT corpus than the TIMIT corpus

(NTIMIT: 19.9% $\rightarrow$ 18.2% vs. TIMIT: 16.4% $\rightarrow$ 14.7%). However, the MTWV gain is significant from 0.006 to 0.04. By applying denoising DBN, the MTWV can be further boosted to 0.232. It is interesting that the denoising DBN has a lower EER 16.2% and a higher MTWV 0.232 on the noisy NTIMIT corpus with only 8kHz sampling rate than the EER of 16.4% and the MTWV of 0.229 for the system using Gaussian posteriorgrams on the clean TIMIT corpus with 16kHz sampling rate. The EER of 14.7% and the MTWV of 0.281 for the unsupervised DBN on the TIMIT corpus can be viewed as an upper bound for the denoising DBN on the NTIMIT corpus since it was used to fine-tune the denoising DBN.

4.6 Summary

In this chapter, we presented a spoken term detection method based on posteriorgrams generated from Deep Belief Networks (DBNs). The proposed representation can be easily adapted to work in both a semi-supervised and an unsupervised training condition. Spoken term detection experiments on the TIMIT corpus showed a 10% relative improvement compared to our previous Gaussian posteriorgram framework in the unsupervised condition. In the semi-supervised setting, the detection performance using the DBN posteriorgram can achieve a comparable performance to fully supervised training when using only 30% of the labeled data. The experiments on the Babel Cantonese corpus demonstrated the language independent nature of the proposed framework.

At the end of this chapter, the DBN posteriorgram framework was extended to noisy speech conditions. By refining the noisy DBN with the corresponding clean DBN posteriorgrams through the back-propagation on continuously density probability functions, on the noisy NTIMIT dataset, we were able to achieve almost the same spoken term detection performance as on the clean data.

Chapter 5

Fast Search Algorithms for Matching Posteriorgrams

In this chapter, we present two fast search algorithms for matching posteriorgrams. In Chapter 3 and Chapter 4, both the spoken term discovery and detection systems described use the standard Dynamic Time Warping (DTW) algorithm on posteriorgram vectors. Due to its intrinsic $O(n^2)$ complexity, DTW is too slow to handle big speech datasets. For instance, spoken term detection experiments in previous chapters were performed on a computer cluster with around 200 CPU cores, which might not be a common computing setup. In order to reduce the computational cost, we propose two lower-bounding based fast matching approaches to speed up the original DTW algorithm on posteriorgrams. The first method is an exact lower-bound based algorithm which does not sacrifice any matching accuracy. The second method is a relaxed lower-bound based algorithm which is less tight compared to the exact lower-bound, but provides additional speed-up. Experiments show that the proposed algorithms are able to speed-up the original DTW by a factor of 320, which indicates that for a fairly large speech corpus, the proposed spoken term detection system can be reasonably deployed on a single desktop computer.

5.1 Introduction

One attractive property of DTW is that it makes no assumptions about the underlying linguistic units. If we have an example speech query pattern, we can find the top K nearest-neighbor (KNN) matches in some corpus of speech utterances. DTW is a natural search mechanism for this application, however, depending on the size of the corpus, there can be a significant amount of computation involved in the alignment process. For aligning two time-series consisting of M and N elements or frames, DTW conservatively takes $O(MN)$ time to compute a match. If one pattern sequence is much longer than the other, $M \ll N$, (e.g., searching for a word in a long recording), then any individual match will be $O(M^2)$, but we will need to initiate multiple DTW searches (in the extreme, starting a new search at every frame in the N frame sequence), which makes the overall computation closer to $O(M^2N)$. For very large N, (e.g., $N > 10^7$) this can be a considerable burden. To solve this computational problem, several lower-bound algorithms have been proposed for DTW search in large databases by Keogh et al. [63, 104, 138]. The basic idea behind lower-bounded DTW is similar in concept to the use of the future underestimate incorporated into an A^* graph search. To start, N lower-bound DTW estimates are computed between the query pattern, Q and every possible speech segment, S, in the corpus of utterances. These lower-bound estimates are sorted into a queue. Then, the lowest lower-bound estimate is incrementally popped off the queue and the actual DTW alignment score is computed for that particular match. This step is repeated until the lowest estimate remaining in the queue is greater than the K^{th}-best DTW score. The K-best search can then be terminated.

In both Chapter 3 and Chapter 4, we have been using a speech representation based on a sequence of posterior probability vectors – posteriorgrams, because of their superior generalization across speakers compared to spectral-based representations [146, 147]. In using the posteriorgram representation, it has been found that the inner product between posteriorgram vectors produces superior results on a variety of DTW-based tasks reported by Asaei et al. in [147, 4]. Thus, if we are to

leverage the lower- bound concept for reducing the DTW computation, we need to derive a lower-bound estimate method for inner product distances. In the first part of this chapter, we describe a lower-bound estimate method that we have developed for inner-product distances, and prove that it is admissible for KNN-based DTW search. We then perform spoken term detection experiments and compare the result to previously reported results from Chapter 3, and show that we can eliminate 89% of the DTW calculations without affecting performance.

In the second part of this chapter, we propose another improved lower-bound estimate using piecewise aggregate approximation (PAA) [64]. PAA can be viewed as a down-sampling approach which can make a short but representative abstraction for a long time series. When comparing two time series, using their corresponding PAA representation saves computation time in exchange for a slightly weaker lower-bound estimate. In our reported spoken term detection experiments on the TIMIT corpus, we consider the total calculation needed for both the lower-bound estimate and DTW-KNN search. The results showed that the proposed PAA lower-bound estimate reduced the computational requirements for DTW-KNN search by 28% compared with the tight lower-bound estimate [149].

5.2 Related Work

Over the years, using DTW to match two time series has attracted much interest. Since DTW allows minor local misalignment, it has been widely used in computer science, biology, and the finance industry [54]. However, to align two time series with n and m lengths, the time complexity of DTW is $O(nm)$, which is not suitable for applications with a large number of data entries. Therefore, ever since DTW was developed, the problem of how to reduce its computational cost has been addressed. In general, previous work about how to speed-up DTW search for large databases can be divided into two categories. The first category is a lower-bound estimate based on DTW indexing methods. The second category is distance matrix approximation methods. We will review these two categories in the following two sections.

5.2.1 Lower-bound Estimate Based Methods

The core idea of prior lower-bound based DTW search methods is to produce a reliable lower-bound approximation of the true DTW distance in linear time. The lower-bound approximation can be then used to build an index for subsequent search.

The first lower-bound estimate was proposed by Yi et al. in 1997 [143]. Yi's lower-bound estimate operated by comparing two one-dimensional time series by using the Euclidean distance as the local distance metric. The basic concept of Yi's lower-bound is that points in one time series that are larger/smaller than the maximum/minimum of the other time series must contribute at least the squared difference of their value, and the maximum/minimum value to the final DTW distance [142]. The limitations of Yi's lower-bound estimate are 1) it is not an exact lower-bound of the true DTW distance, which results in false dismissals in some cases; 2) the computational overhead is large since it needs to scan the whole series to obtain the maximum/minimum value.

Kim et al. improved Yi's results and proposed an exact lower-bound estimate which does not allow false dismissals [66]. Kim's lower-bound calculation is based on four values from the two time series. The values are the first, the last, the maximum and the minimum of two time series. Since Kim's lower-bound estimate also requires the maximum and minimum of two time series, it still has a fairly large computational overhead. Moreover, the definition of maximum and minimum values dose not allow the lower-bound estimate to work on multi-dimensional time series. Subsequent experiments also found that Kim's lower-bound was very loose in some datasets [63].

Keogh et al. [16, 63] proposed another exact lower-bound estimate for DTW using local warping constraints such as the Sakoe-Chiba band [112] or Itakura parallelogram [56]. The local path warping constraint prevents the DTW path from going into one direction too much. To compare two time series Q and C using DTW with local path warping constraint r, Keogh's lower-bound first calculates two envelopes called the Upper envelope U and Lower envelope L respectively. Formally, the Upper envelope U can be written as

$$U = (u_1, \cdots, u_n) = (\max(q_{1-r}, q_{1+r}), \cdots, \max(q_{i-r}, q_{i+r}), \cdots, \max(q_{n-r}, q_{n+r})) \tag{5.1}$$

where n is the length of the time series Q, u_i is the i-th entry in U, and q_i is the i-th element in Q. The Lower envelope L can be written as

$$L = (l_1, \cdots, l_n) = (\min(q_{1-r}, q_{1+r}), \cdots, \min(q_{i-r}, q_{i+r}), \cdots, \min(q_{n-r}, q_{n+r})) \tag{5.2}$$

If Q and C are one-dimensional time series and the local distance metric is Euclidean distance, by using U and L, the lower-bound estimate for DTW on Q and C can be written as

$$\text{LB\_Keogh}(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (c_i - U_i)^2 & \text{if } c_i > U_i \\ (c_i - L_i)^2 & \text{if } c_i < L_i \\ 0 & \text{otherwise} \end{cases}} \tag{5.3}$$

where U_i is the i-th entry of U and c_i is the i-th entry in the time series C. The core idea behind this lower-bound can be interpreted as all points not falling within two envelopes contribute at least the squared difference of their values and the envelope values to the final DTW distance. Keogh et al. proved that this lower-bound estimate is an exact lower-bound, and guarantees no false dismissals [63].

The proposed lower-bound estimate in this chapter is close to Keogh's lower-bound idea but differs in three key aspects. First, the proposed lower-bound is able to handle multi-dimensional time series. Second, the proposed lower-bound uses an inner-product distance measure instead of the Euclidean distance. Third, the proposed lower-bound only requires the construction of the Upper envelope, ignoring the Lower envelope, while still maintaining no false dismissals.

5.2.2 Distance Matrix Approximation Based Methods

There has also been some research concentrating on fast approximation of the DTW distance matrix by using a sparse representation of the data. Both approaches essentially reduce the size of the DTW distance matrix so that DTW is able to run much faster on those approximated distance matrices.

Keogh et al. proposed a piecewise linear approximation method which downsamples the time series with linear prediction functions [64]. The reduced number of points in the DTW distance matrix provides significant speed-up at the expense of some false dismissals. Chan et al. proposed another data downsampling approach using wavelet filters [17]. The core idea behind both approaches is to represent the original data with fewer points by using predictive functions. Since both approaches depend on prediction functions, it is clear that the prediction functions have to be carefully chosen based on specific applications, resulting in flexibility issues when changing domains.

More recently, Jansen et al. proposed a two stage distance matrix approximation method for fast DTW matching [59, 60, 61]. The first stage is to use locality sensitive hashing (LSH) to represent the original time series data. LSH can be viewed as a projection function which maps high dimensional data to low dimensional bit vectors [2]. After the mapping, the relative similarity between data points in the original space can be approximated in the low-dimensional space by simple and very fast distance metrics such as the Hamming distance. LSH inherently handles the high dimensional time series, and the computational overhead is relatively low. After representing the original data by using LSH, the DTW distance matrix is calculated based on LSH vectors. In the second stage, instead of running DTW directly on the distance matrix, an image-processing based approach is used to locate regions which are highly likely to contain a match. The real DTW alignment is only then performed on those regions, ignoring the remaining matrix, which results in a huge reduction in the total computing time in large databases. Several spoken term detection and spoken term discovery experiments based on this two stage method have showed promising results.

In sum, all inexact approximation based methods suffer from the same drawback which is the introduction of false dismissals. Often the approximation thresholds need to be carefully tuned to prevent too many false dismissals; these thresholds can vary widely for different applications. Since our proposed lower-bound estimates still require a certain amount of real DTW calculations to determine the final KNN list, these approximation-based methods could be built upon our proposed lower-bound estimate to improve the real DTW calculation in the following KNN search.

5.3 DTW on Posteriorgrams

The posteriorgram is a feature representation of speech frames generated from a GMM or a DBN, as described in Section 3.2 and Section 4.4. To match posteriorgrams, consider the two posteriorgram sequences for a speech query,

$$Q = \{\vec{q}_1, \ldots, \vec{q}_M\}$$

and a speech segment,

$$S = \{\vec{s}_1, \ldots, \vec{s}_N\}$$

where $\vec{q}_i$ and $\vec{s}_j$ are D-dimensional posterior probability vectors. The local distance between $\vec{q}_i$ and $\vec{s}_j$ can be defined by their inner product as

$$d(\vec{q}_i, \vec{s}_j) = -\log(\vec{q}_i \cdot \vec{s}_j)$$

Using the DTW algorithm, the overall best alignment between Q and S is

$$\text{DTW}(Q, S) = \min_{\phi} A_{\phi}(Q, S)$$

where ϕ is the warping function and A is the scoring function for each alignment according to a specific warping function ϕ. Detailed description can be found in Section 3.4.

5.4 Lower-Bounded DTW on Posteriorgrams

The idea of the lower-bounded DTW is that instead of calculating the exact DTW distance, an underestimated distance can be first computed to obtain an approximation of the real distance. The following search mechanism can use the underestimated distance and calculate the real distance when necessary.

5.4.1 Lower-bound Definition

Given two posteriorgram sequences, Q, and S, we can determine a lower-bound of their actual DTW score by first deriving an upper-bound envelope sequence,

$$U = \{\vec{u}_1, \cdots, \vec{u}_M\}$$

where

$$\vec{u}_i = \{u_i^1, \cdots, u_i^D\}$$

and

$$u_i^j = \max(q_{i-r}^j, \cdots, q_{i+r}^j)$$

Note that the variable r is the same as is used for the DTW global path constraint, and that, in general, $\sum_{j=1}^{D} u_i^j \geq 1$. U can thus be viewed as a D-dimensional windowed maximum envelope derived from Q. Figure 5-1 illustrates an example of U on one dimension of a posteriorgram. A lower-bound DTW score between two posteriorgrams, Q and S, can then be defined as

$$L(Q, S) = \sum_{i=1}^{l} d(\vec{u}_i, \vec{s}_i) \tag{5.4}$$

where $l = \min(M, N)$. Note that the computational complexity of computing $L(Q, S)$ is only $O(l)$.

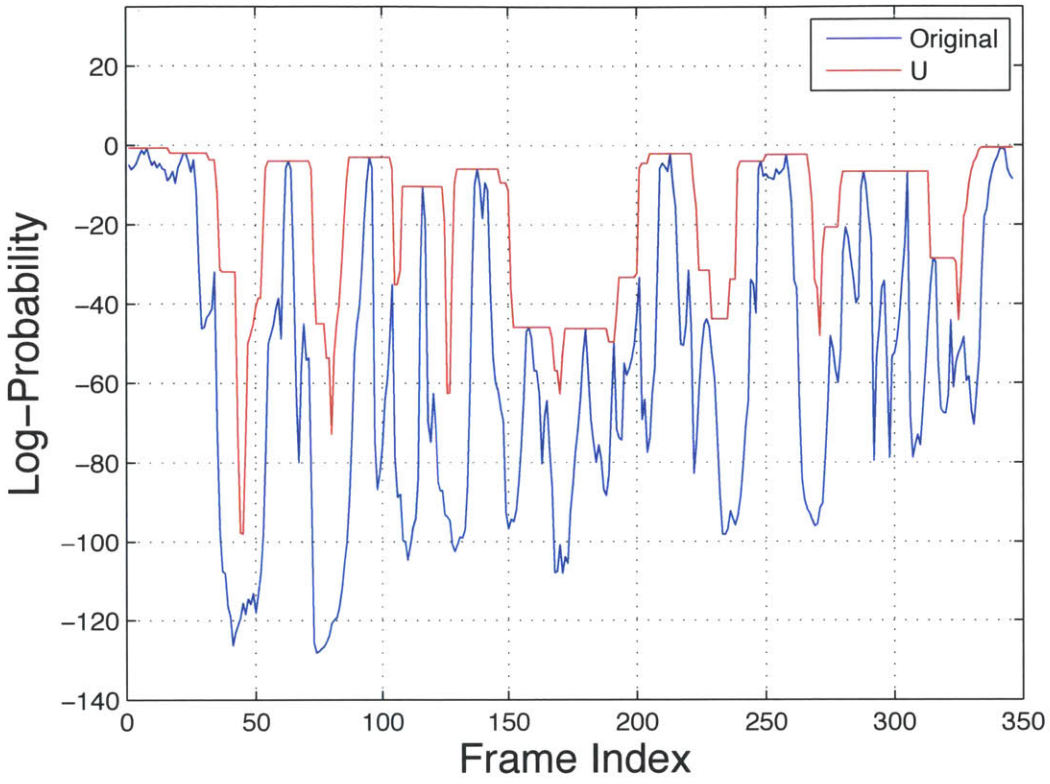


Figure 5-1: Example of a 1-dimensional upper-bound envelope sequence (red) compared to the original posteriorgram (blue) for $r = 8$.

5.4.2 Lower-bound Proof

To prove that $L(Q, S) \leq \text{DTW}(Q, S)$ for posteriorgram sequences Q and S, we follow the strategies that are used by Keogh et al. and Rath et al. in [63, 104]. By expanding both terms, we wish to show that

$$\sum_{i=1}^{l} d(\vec{u}_i, \vec{s}_i) \leq \sum_{k=1}^{K_\phi} d(\vec{q}_{\phi_q(k)}, \vec{s}_{\phi_s(k)})$$

where ϕ is the best warping path that produces $\text{DTW}(Q, C)$. The right hand side (RHS) can be further split into two parts

$$\sum_{i=1}^{l} d(\vec{u}_i, \vec{s}_i) \leq \sum_{k \in \text{MA}} d(\vec{q}_{\phi_q(k)}, \vec{s}_{\phi_s(k)}) + \sum_{k \in \text{UM}} d(\vec{q}_{\phi_q(k)}, \vec{s}_{\phi_s(k)})$$

where MA denotes a matched set containing exactly l warping pairs, while UM corresponds to an unmatched set that includes all remaining warping pairs. We construct the matched set as follows. For the i^{th} term on the left hand side (LHS), a warping pair $(\phi_q(k), \phi_s(k))$ from the RHS is placed into MA if $\phi_s(k) = i$. If there are multiple warping pairs from the RHS with $\phi_s(k) = i$, we select the pair with smallest $\phi_q(k)$ (although it only matters that one is selected). Note that there are always enough pairs to select into the matched set since $l \leq K_\phi$. By following this construction rule we ensure that the size of the matched set is exactly l so that each term on the LHS is matched exactly once by an element in the matched set. Based on the definition of the inner-product distance, all terms on the RHS are positive. Thus, all terms in UM can be eliminated if we can prove that the LHS is less than the terms in MA.

Consider an individual warping pair in MA, $(\phi_q(k), \phi_s(k))$, as it relates to the i^{th} term on the LHS, $d(\vec{u}_i, \vec{s}_i)$. By expanding the distance function back into the negative log inner-product, the inequality we need to prove becomes

$$\sum_{i=1}^{l} -\log(\vec{u}_i \cdot \vec{s}_i) \leq \sum_{i \in \text{MA}} -\log(\vec{q}_{\phi_q(i)} \cdot \vec{s}_{\phi_s(i)})$$

Since both sides now have the same number of terms, the inequality holds if each term on the LHS is less than or equal to the corresponding term on the RHS. By eliminating the log and examining only the individual dot product terms, we therefore need to show

$$\vec{u}_i \cdot \vec{s}_i \geq \vec{q}_{\phi_q(i)} \cdot \vec{s}_{\phi_s(i)}$$

Note that because of the way the matched set was selected, $\phi_s(i) = i$ so that $\vec{s}_i = \vec{s}_{\phi_s(i)}$. Since the DTW global path constraint r limits the warping pair so that $|\phi_q(i) - \phi_s(i)| \leq r$ we can also say $|\phi_q(i) - i| \leq r$, or $i - r \leq \phi_q(i) \leq i + r$. We can therefore see from the definition of u_i^j, that $u_i^j \geq q_{\phi_q(i)}^j$ so that our inequality holds:

$$L(Q, S) \leq \text{DTW}(Q, S)$$

5.4.3 Nontrivialness Check for Exact Lower-bound

One special property of the posteriorgram is that the summation of posterior probabilities from all dimensions should be one. Thus, the lower-bound could be trivial if $\vec{u}_i \cdot \vec{s}_i \geq 1$ because on the RHS $\vec{q}_{\phi_q(i)} \cdot \vec{s}_{\phi_s(i)} \leq 1$. However, if we let

$$u_{max} = \max(u_i^1, \ldots, u_i^D)$$

the LHS can be written as

$$\vec{u}_i \cdot \vec{s}_i \leq u_{max} \cdot \sum_{j=1}^{D} s_i^j = u_{max}$$

since $\sum_{j=1}^{D} s_i^j = 1$. Since we also know that $u_{max} \leq 1$ we can see that the lower-bound estimate is not the trivial case.

5.5 Piecewise Aggregate Approximation for DTW on Posteriorgrams

The time consumed for the lower-bound estimate is $O(M)$ for a spoken term query with M frames. For a database with N entries however, calculating the lower-bound estimate for a spoken term query and every candidate speech segment in the database would take $O(MN)$ time, which could still be a considerable burden when N is very large (e.g., $N > 10^7$). To address this disadvantage and further improve the efficiency of the search, we apply the concept of piecewise aggregate approximation (PAA) to reduce the length of posteriorgrams into B blocks, and estimate a slightly weaker lower-bound in exchange for a faster lower-bound calculation.

Prior research with PAA has also focused on using the Euclidean distance as the similarity metric between one dimensional time series [63, 142]. If we therefore hope to leverage the PAA concept for reducing lower-bound calculations, we need to develop a new PAA lower-bound estimate for posteriorgrams. Therefore, in the following sections, we describe a PAA lower-bound estimation approach, and prove

that it is admissible for DTW-KNN search. Through experiments, we show that using a PAA representation of posteriorgrams might lead to a weaker lower-bound estimate which would increase the necessary DTW calculations in the KNN search. However, considering the total time cost, the PAA lower-bound estimate combined with the DTW-KNN search improves the spoken term detection efficiency by 28%.

5.5.1 PAA Definition

Given two posteriorgrams Q and S, without loss of generality, we assume they have the same length $M = N$. Define two approximation posteriorgrams

$$\hat{U} = \{\hat{U}_1, \cdots, \hat{U}_B\}$$

and

$$\hat{S} = \{\hat{S}_1, \cdots, \hat{S}_B\}$$

$\hat{U}_i$ denotes the i^{th} block of the approximated upper-bound envelope U, and is defined as

$$\hat{U}_i = \{\hat{u}_i^1, \cdots, \hat{u}_i^D\}$$

where each dimension

$$\hat{u}_i^p = \max\left(u^p_{\frac{M}{B}(i-1)+1}, \cdots, u^p_{\frac{M}{B}i}\right) \tag{5.5}$$

$\hat{S}_i$ denotes the i^{th} block of the approximated S and can be defined as $\hat{S}_i = \{\hat{s}_i^1, \cdots, \hat{s}_i^D\}$ where each dimension

$$\hat{s}_i^p = \frac{B}{M} \sum_{j=\frac{M}{B}(i-1)+1}^{\frac{M}{B}i} s_j^p \tag{5.6}$$

Note that if M is not divisible by B, $\frac{M}{B}$ is floored and the remaining frames form an additional block. It is clear that the PAA block reduction process is similar to a

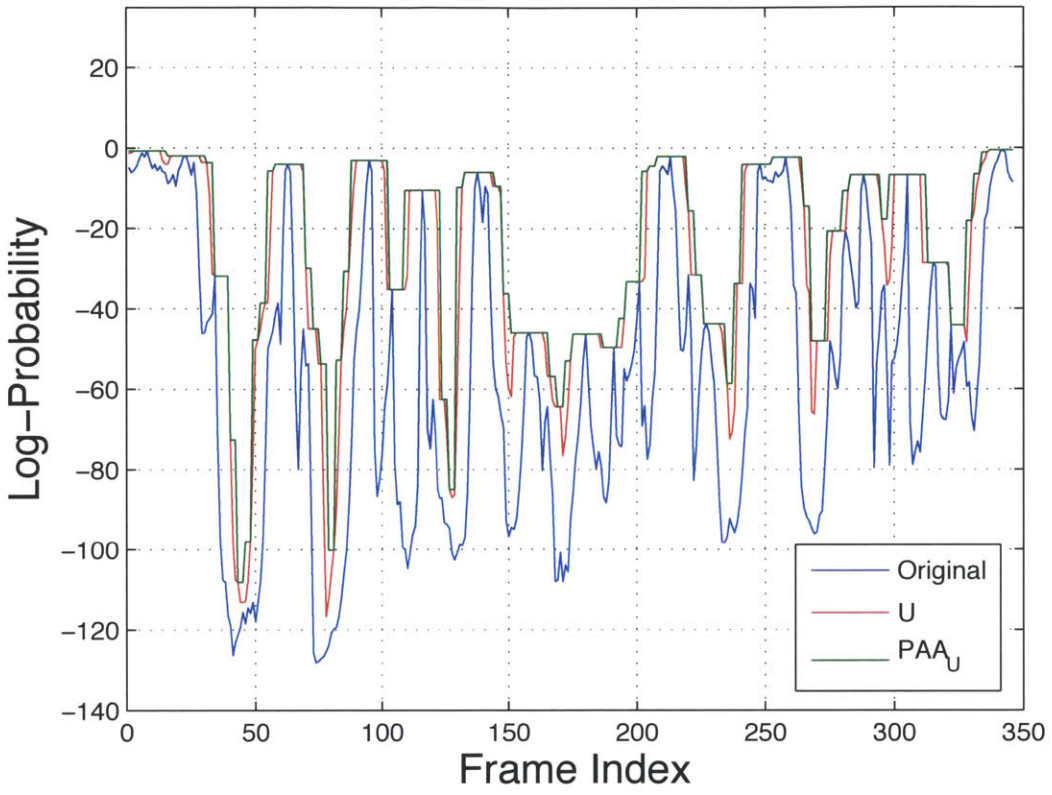


Figure 5-2: Example of a one-dimensional PAA sequence (3 frames per block) (green), an upper-bound envelope sequence (red) and an original posteriorgram (blue) for r = 5

down-sampling process. For a speech query, for each dimension, the maximum value of the frames within a block is used to represent the block, while for a speech segment, the average value of the frames within a block is used. Figure 5-2 demonstrates an example of the approximated $\hat{Q}$ and the upper-bound envelope U on one dimension of a posteriorgram.

Using $\hat{U}$ and $\hat{S}$, the PAA lower-bound estimate for DTW on posteriorgrams can be defined as

$$\text{PAA}(Q, S) = \sum_{i=1}^{B} \frac{M}{B} \cdot d(\hat{U}_i, \hat{S}_i) \tag{5.7}$$

where $d(\cdot)$ is the inner product function.

5.5.2 PAA Lower-bound Proof

To prove $\text{PAA}(Q,S) \leq L(Q,S)$, without loss of generality, we first assume that $B = 1$ which indicates the entire posteriorgram sequence is considered as one block. If under this assumption the inequality holds, it is clear that the same proof can be applied to each block when $B \geq 1$. (Note that if $B = M$ then $\text{PAA}(Q,S) = L(Q,S)$.)

Since $B = 1$, Eq. 5.7 can be simplified as

$$\text{PAA}(Q,S) = M \cdot \left(-\log \sum_{p=1}^{D} \hat{u}_1^p \cdot \hat{s}_1^p \right)$$

According to the definition of the original lower-bound estimate in Eq. 5.4, the inequality becomes

$$M \cdot \left(-\log \sum_{p=1}^{D} \hat{u}_1^p \cdot \hat{s}_1^p \right) \leq \text{L}(Q,S) = \sum_{i=1}^{M} -\log \sum_{p=1}^{D} u_i^p \cdot s_i^p$$

After absorbing the summation into the log and negating both sides, the inequality becomes

$$\log \left(\sum_{p=1}^{D} \hat{u}_1^p \cdot \hat{s}_1^p \right)^M \geq \log \prod_{i=1}^{M} \sum_{p=1}^{D} u_i^p \cdot s_i^p$$

which is equivalent to prove

$$\left(\sum_{p=1}^{D} \hat{u}_1^p \cdot \hat{s}_1^p \right)^M \geq \prod_{i=1}^{M} \sum_{p=1}^{D} u_i^p \cdot s_i^p \tag{5.8}$$

Note that since $B = 1$, according to the definition of the block reduction process in Eq. 5.5 and Eq. 5.6, it is clear that

$$\hat{u}_1^p = \max\left(u_1^p, u_2^p, \cdots, u_M^p\right)$$

$$\hat{s}_1^p = \frac{1}{M} \sum_{i=1}^{M} s_i^p$$

Therefore, the left hand side of Eq. 5.8 can be written as

$$\begin{aligned}\left(\sum_{p=1}^{D}\hat{u}_1^p\cdot\hat{s}_1^p\right)^M &= \left(\frac{1}{M}\sum_{p=1}^{D}\sum_{i=1}^{M}\hat{u}_1^p\cdot s_i^p\right)^M \\ &\geq \left(\frac{1}{M}\sum_{p=1}^{D}\sum_{i=1}^{M}u_i^p\cdot s_i^p\right)^M\end{aligned} \tag{5.9}$$

where $\hat{u}_1^p \geq u_i^p, \forall i \in [1, M]$ based on Eq. 5.5. Interchanging the summation in Eq. 5.9, the inequality we need to prove becomes

$$\left(\frac{1}{M}\sum_{i=1}^{M}\sum_{p=1}^{D}u_i^p\cdot s_i^p\right)^M \geq \prod_{i=1}^{M}\sum_{p=1}^{D}u_i^p\cdot s_i^p \tag{5.10}$$

Let $a_i = \sum_{p=1}^{D} u_i^p \cdot s_i^p$, the inequality becomes

$$\left(\frac{1}{M}\sum_{i=1}^{M}a_i\right)^M \geq \prod_{i=1}^{M}a_i$$

Since it is clear that $a_i \geq 0$, the arithmetic mean is always greater than or equal to the geometric mean. Combining with the proof in Section 5.4.2, the following inequality holds

$$\mathrm{PAA}(Q,S) \leq \mathrm{L}(Q,S) \leq \mathrm{DTW}(Q,S) \tag{5.11}$$

which indicates the PAA lower-bound estimate is admissible to DTW-KNN search.

5.5.3 Nontrivialness Check for PAA Lower-bound

Since the sum of posterior probabilities in a posteriorgram should be one, in order to avoid trivialness we should prove that the approximated posteriorgram has the property that

$$\left(\sum_{p=1}^{D}\hat{u}_1^p\cdot\hat{s}_1^p\right)^M \leq 1$$

From Eq. 5.9 and Eq. 5.10, it is clear that

$$\sum_{p=1}^{D} \hat{u}_1^p \cdot \hat{s}_1^p = \frac{1}{M} \sum_{i=1}^{M} \sum_{p=1}^{D} \hat{u}_1^p \cdot s_i^p$$

Let $\hat{u}_{max} = \max(\hat{u}_1^1, \hat{u}_1^2, \cdots, \hat{u}_1^D)$. We have

$$\begin{aligned} \frac{1}{M} \sum_{i=1}^{M} \sum_{p=1}^{D} \hat{u}_1^p \cdot s_i^p &\leq \frac{\hat{u}_{max}}{M} \cdot \sum_{i=1}^{M} \sum_{p=1}^{D} s_i^p = \frac{\hat{u}_{max}}{M} \cdot \sum_{i=1}^{M} \cdot 1 \\ &= \frac{\hat{u}_{max}}{M} \cdot M \cdot 1 = \hat{u}_{max} \leq 1 \end{aligned}$$

where $\sum_{p=1}^{D} s_i^p = 1$ based on the posteriorgram definition.

5.6 KNN Search with Lower-Bound Estimate

In order to determine the K nearest neighbor (KNN) segmental matches to a query, the default approach is to consider every possible match for all possible segments in each corpus utterance. By incorporating the lower-bound estimate described previously, we can find the top KNN matches much more efficiently, as shown in the pseudo-code in Algorithm 1.

The basic idea of the algorithm is to use the lower-bound DTW estimate to prune out segments whose lower-bound DTW estimates are greater than the K^{th} best DTW score. In Algorithm 1, the function *ComputeLB* calculates the lower-bound DTW estimate between the spoken term query, Q, and every possible utterance segment, S, using the upper envelope U. All utterance segments and their associated lower-bound estimates are stored in a priority queue ranked by the lower-bound estimate.

During KNN search, the algorithm begins from the top (smallest estimate) of the priority queue and calculates the actual DTW distance between the spoken term query and the associated segment. After using the function *FindC* to locate the utterance containing the current segment, the function *UpdateC* updates the best DTW distance in that utterance. Then, the function *UpdateRL* updates the result

Algorithm 1: KNN Search with Lower-Bound

Data: Q,U and C
Result: RL containing k most possible utterances having the spoken term Q
begin
  for *each utterance* $c \in C$ **do**
    for *each segment* $s \in c$ **do**
      $lb = \text{ComputeLB}(U, s)$
      $PQ.\text{push}([lb, s])$
  $KthBest = MaxFloat$
  while $PQ \neq \emptyset$ *AND* $(|RL| < k$ *OR* $PQ.top.lb < KthBest)$ **do**
    $[lb, s] = PQ.top$
    $v = \text{DTW}(Q, s)$
    $c = \text{FindC}(s)$
    $\text{UpdateC}(c, s, v)$
    if $c \in RL$ **then** $\text{UpdateRL}(RL)$
    else $RL.\text{add}(c)$
    if $|RL| \leq k$ **then** $KthBest = \text{FindMax(RL)}$
    else $KthBest = \text{FindKthMax(RL)}$
    $PQ.\text{pop}()$

list if the best DTW score in the current utterance changes. If the result list does not contain the current utterance, the current utterance is added into the result list. Finally, if the size of the result list is less than or equal to K, the K^{th} best is set to the maximum value of the associated DTW score of the utterances in RL. If the size of the result list is greater than K, then the K^{th} best is set to the K^{th} maximum value in RL. The search algorithm ends if K possible utterances are in the result list and the lower-bound estimates of all remaining segments in the priority queue are greater than the K^{th} best value. Figure 5-3 demonstrates the Algorithm 1. In the figure, the left list represents the priority queue PQ that contains lower-bound estimates, while the right list denotes the output result list RL.

5.7 Experiments and Results

Since we have shown that the lower-bound estimate results in a KNN-DTW search that is admissible, we wish to measure how much computation can be saved with the

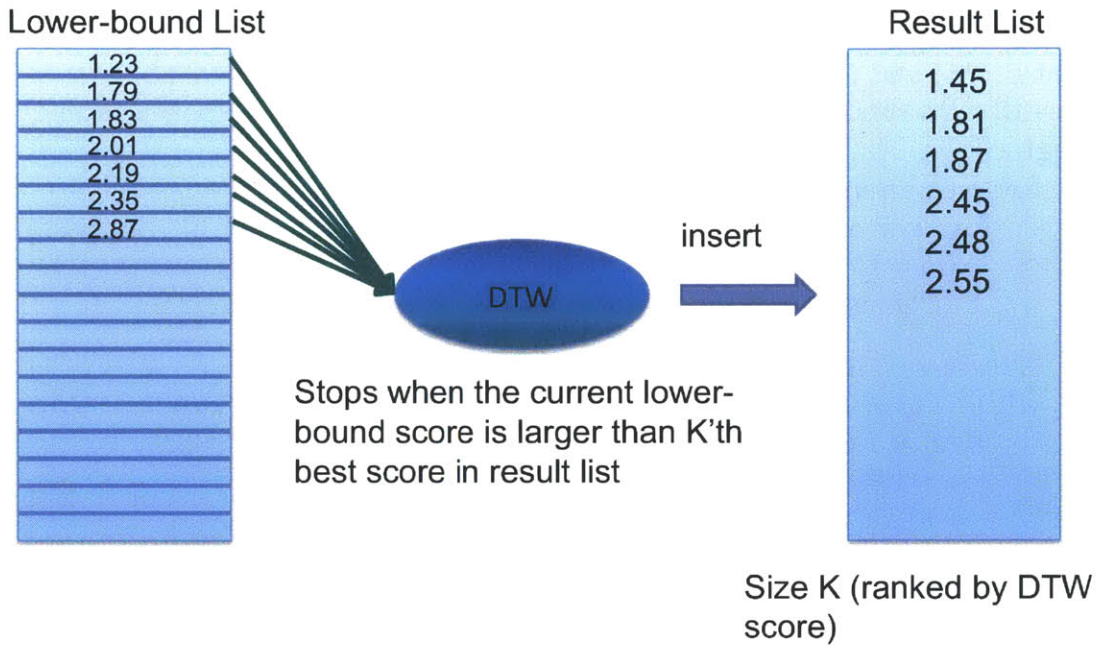


Figure 5-3: An illustration of KNN search with lower-bound estimate (presented in Algorithm 1). The left list represents the priority queue PQ that contains lower-bound estimates, while the right list denotes the output result list RL.

lower-bound estimate as well as the PAA lower-bound estimate. In order to do this, we have chosen to duplicate the spoken term detection experiments that we have performed on the TIMIT corpus using posteriorgram-based DTW search in Chapter 3. The same experiment configurations were used and the spoken term list was fixed as in Section 3.6.1. Performance was still measured by the average equal error rate (EER): the average rate at which the false acceptance rate is equal to the false rejection rate.

5.7.1 The Exact Lower-Bound Results

Since the lower-bound estimate is admissible, the results obtained by the new KNN method are the same as have been reported earlier where we achieved 16.4% equal error rate (EER) as in Chapter 3. Of greater interest, however, is the amount of computation we can eliminate with the lower-bound estimate.

Figure 5-4 summarizes the amount of computational savings which can be achieved with the lower-bound estimate. The figure plots the average DTW ratio (scaled by

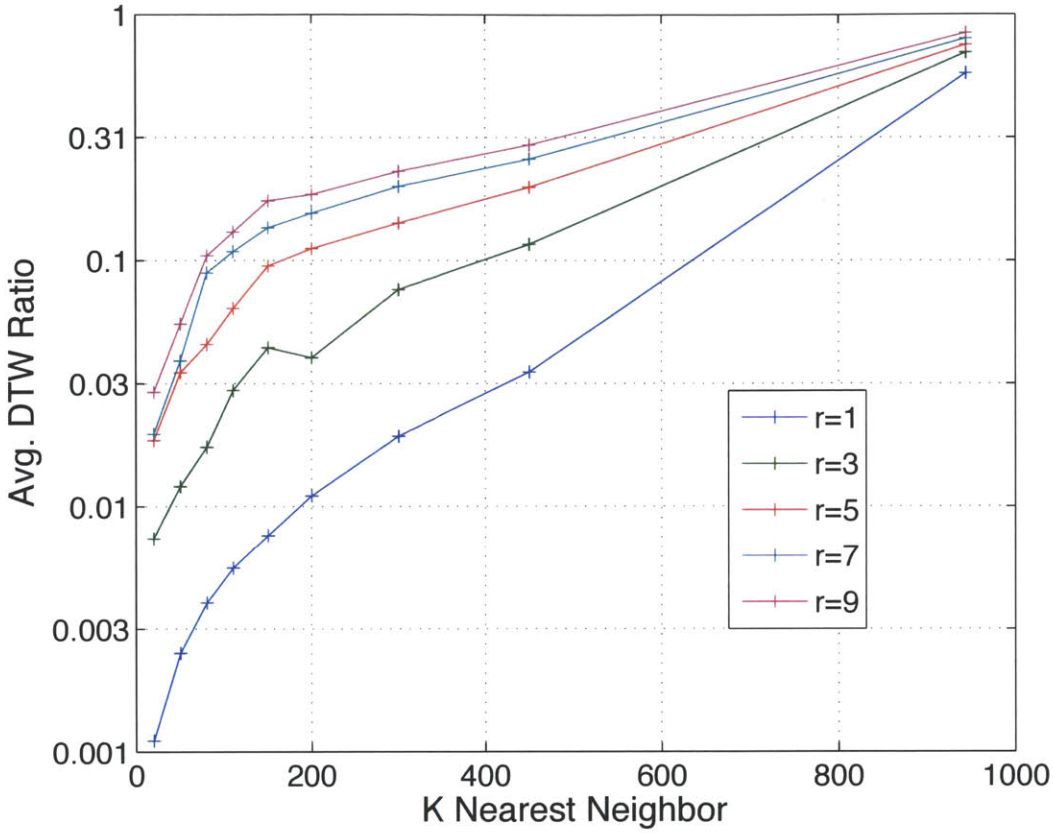


Figure 5-4: Average DTW ratio against KNN size for different global path constraints, r. The average DTW ratio is the ratio of the number of utterance segments that require a DTW score to be computed divided by the total number of segments for a particular spoken term search, averaged across all 10 spoken term queries. For $r = 1$, an exact DTW computation is needed for only 0.11% of all possible segments when K=10.

log base 10) against the size of K in the KNN search for several different DTW path constraint settings ($r = 1, 3, 5, 7, 9$). The average DTW ratio is the ratio of the number of utterance segments that require a DTW score to be computed divided by the total number of segments for a particular spoken term search, averaged across all 10 spoken term queries. For $r = 1$, for example, an exact DTW computation is needed for only 0.11% of all possible segments when K=10. In our prior experiments, we achieved a minimum EER for $r = 5$ [146]. With the current framework we achieve the same EER for K=200, and require only 11.2% of exact DTW scores to be computed compared to our previous results. Finally, it is interesting to note that even when K=944

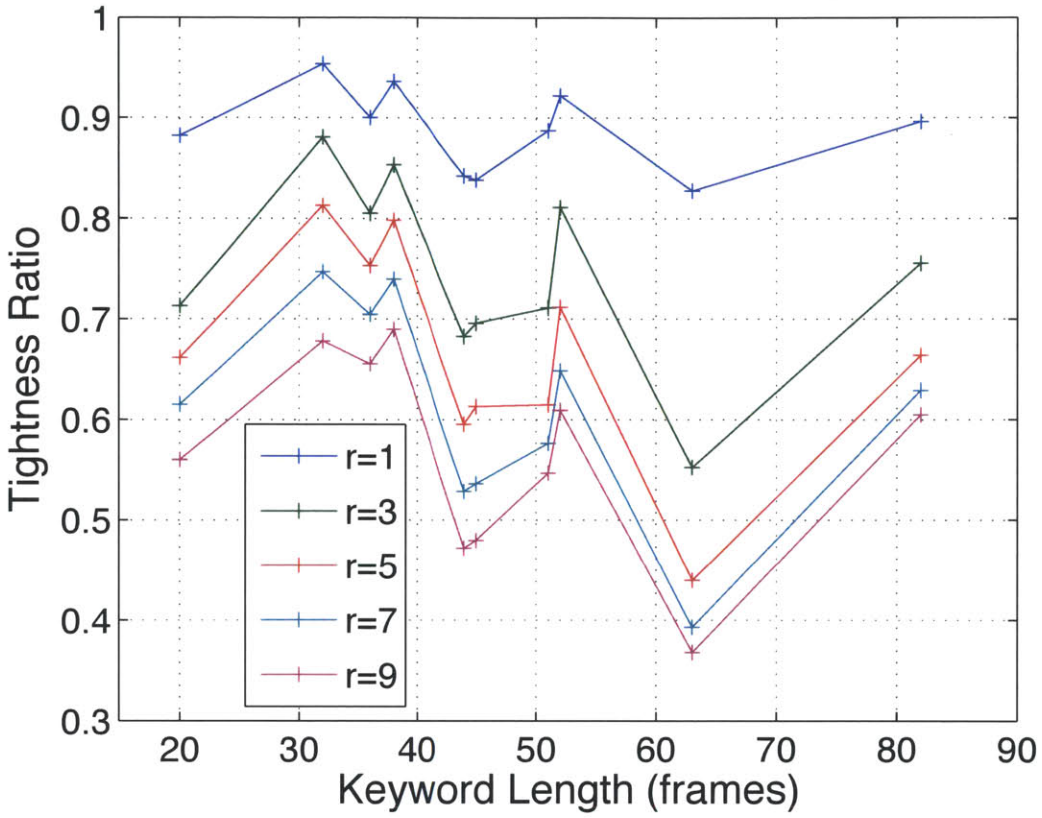


Figure 5-5: Tightness ratio against different query lengths. The tightness ratio is defined as the lower bound value divided by the actual DTW distance. 944 nearest neighbors were computed for each spoken term in order to obtain the tightness ratio for every segment. Five settings of the path constraint are also included.

(i.e., which finds all utterances in the test set), only 75% of the DTW calculations are needed since the lower-bound estimate prunes away undesirable segments in each utterance.

Figure 5-5 illustrates the tightness ratio against different spoken term lengths. The tightness ratio is defined as the lower bound value divided by the actual DTW distance. 944 nearest neighbors were computed for each spoken term in order to obtain the tightness ratio for every segment. Five settings of the path constraint are also included. Based on the figure, smaller path constraints lead to tighter estimation of the actual DTW distance, indicating more pruning power. Another observation is that in general, the tightness ratio decreases if the spoken term length increases. The reason is probably because long warping paths ($\max(n, m) \leq K \leq n + m + 1$) might

bring more terms into the unmatched set, which increases the margin of the lower bound estimation.

In terms of computation time, the results are quite dramatic. While our original DTW experiments required approximately 2 minutes to search for matches for the 10 spoken terms on a 200 CPU compute cluster, the new DTW-KNN method takes approximately 2 minutes on a single CPU or about 12s/query. Since the test set corpus corresponds to approximately 48 minutes of speech, this translates to approximately 14 seconds/query/corpus hour/CPU.

5.7.2 PAA Lower-Bound Results

Since the PAA lower-bound estimate is admissible (Eq. 5.11), the spoken term detection accuracy is the same as the previously reported result which was 16.4% equal error rate as in Chapter 3.

Figure 5-6 illustrates why the proposed PAA lower-bound estimate can speed up the overall calculation compared to the original lower-bound method. The green dotted curve (LB) represents the actual inner product calculations needed for lower-bound estimate only. Each solid curve (DTW) represents the actual inner product calculations needed for the DTW calculation with different r, where r is the global path constraint in DTW. Each dashed curve (TOL) is the sum of the solid curve (in the same color) and the green dotted curve, which indicates the total inner product calculations needed. Note that, as mentioned earlier, when there is only one frame per block, the PAA lower-bound estimate degrades to the original lower-bound estimate. For example, when $F = 1, r = 5$, the original approach requires 1.12×10^8 inner product calculations as well as 5.45×10^7 calculations for the DTW. In this case, the time consumed on the lower-bound estimate is more than the time consumed on the DTW calculations. However, if we increase the number of frames per block, the number of inner product calculations required for the lower-bound estimates decreases. At the same time, since the lower-bound estimates become weaker, the number of inner product calculations required for the DTW increases. Considering the total inner product calculations, it can be seen that a large amount of inner product calculations

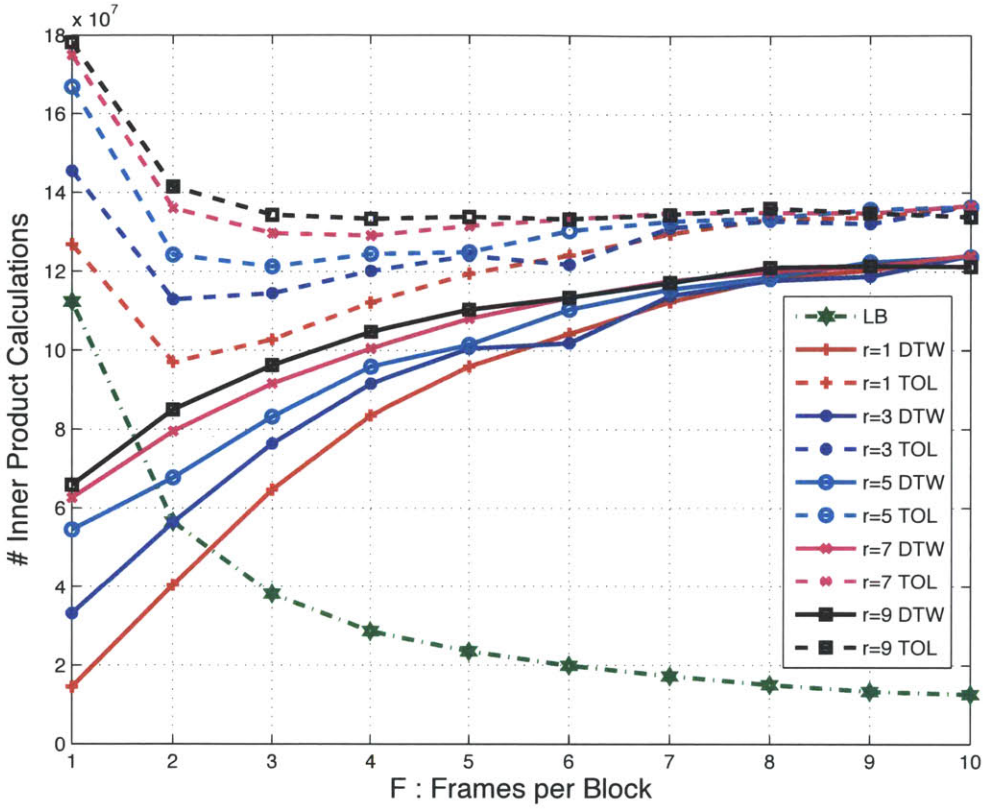


Figure 5-6: Actual inner product calculation against different number of frames per block. The green dotted curve (LB) represents the actual inner product calculations needed for lower-bound estimate only. Each solid curve (DTW) represents the actual inner product calculations needed for the DTW calculation with different r. r is the global path constraint in DTW. Each dashed curve (TOL) is the sum of the solid curve (in the same color) and the green dotted curve, which indicates the total inner product calculations needed.

can be saved compared with the original approach ($F = 1$). Since according to the results in Section 3.6.1, the minimum EER was achieved when $r = 5$, the PAA lower-bound estimate for this r value can save 28% of the inner product calculations when $F = 3$ and $K = 200$.

Figure 5-7 compares the average inner product save ratio against different K nearest neighborhoods when $r = 5$. The inner product save ratio is defined as the percentage of total inner product calculations saved comparing with the original lower-bound estimate. As seen in the figure, for this task both small and large values of K achieve greater overall computational savings compared to values of K between 100 and 450.

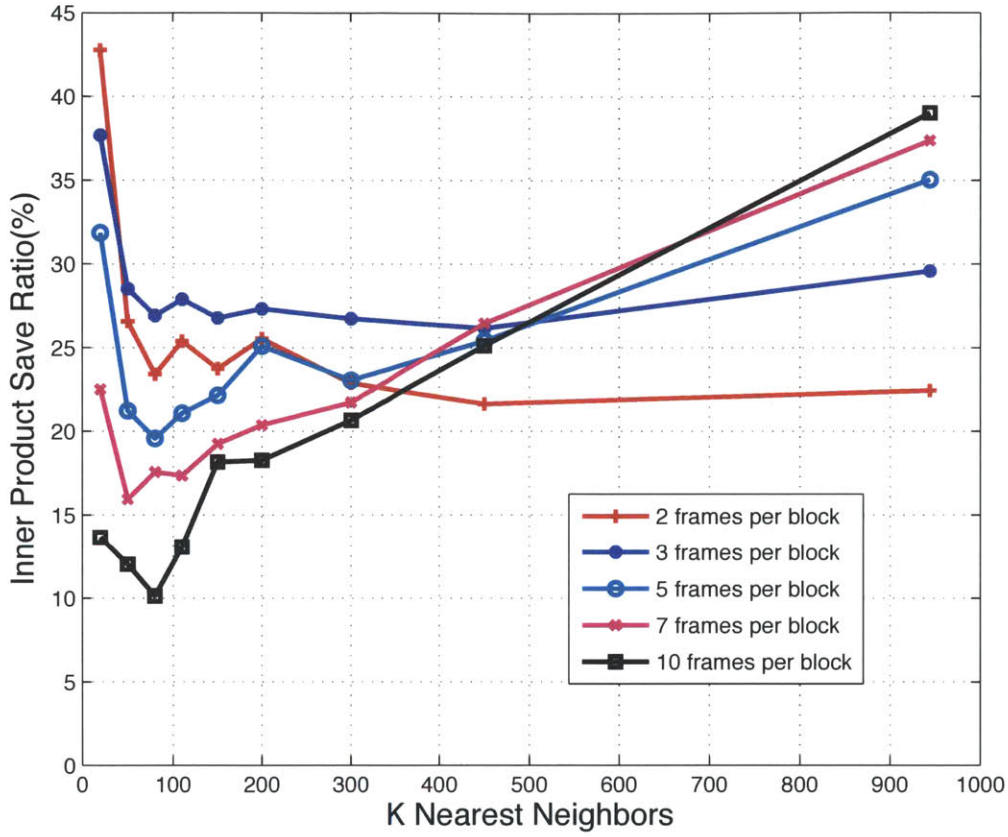


Figure 5-7: Average inner product calculation save ratio against different K nearest neighbors. The inner product save ratio is defined as the percentage of total inner product calculations saved comparing with the original lower-bound estimate. For this task both small and large values of K achieve greater overall computational savings compared to values of K between 100 and 450.

We believe the reason for this behavior is because when K is small, searching K best matches is a highly competitive process. A slightly weaker lower-bound estimate will increase DTW calculations dramatically so that the inner product save ratio is dominated by the inner product calculations needed on the DTW side. As a result, the save ratio is inversely proportional to the number of frames per block because having more frames in a block results in a greater underestimation of the lower-bound. In contrast, for large K the KNN searches almost all speech segments. The inner product save ratio largely depends on the number of inner product calculations in the lower-bound estimate. Thus, the PAA lower-bound estimates with large block sizes

achieve greater overall savings.

In terms of computation time, the original lower-bound approach takes 120 seconds on a single desktop CPU on average, while the PAA lower-bound method needs 87 seconds. Since the TIMIT test corpus contains 48 minutes of speech, each spoken term search takes approximately 10 seconds/query/corpus hour/CPU, compared with 14 seconds/query/corpus hour/CPU achieved with the original lower-bound estimate.

5.8 Summary

In this chapter, we presented a lower-bound estimate and its corresponding fast approximation for DTW-based methods that uses an inner-product distance metric such as for a posteriorgram representation. Given a query posteriorgram and a test posteriorgram, the lower-bound is obtained by calculating the inner-product distance of the upper envelope of the query posteriorgram against the test posteriorgram. The lower-bound underestimates the actual DTW score between the query and test posteriorgrams, which provides an efficient pruning mechanism for KNN search. Based on the experimental results in the spoken term detection task, the KNN-DTW search can eliminate 89% of DTW calculations for a tight lower-bound estimate and achieve another 28% speedup for a PAA approximated lower-bound estimate. Since both lower-bound estimates guarantee no false dismissals, the system achieves the same spoken term detection rate as the baseline system without pruning.

Finally, since the lower-bound calculation can be easily parallelized, we will show how to use other computing architectures such as GPU computing to further speed-up the entire algorithm in the next chapter.

Chapter 6

GPU Accelerated Spoken Term Detection

In this chapter, we will present a fast spoken term detection system based on Graphical Processing Units (GPUs). In the previous two chapters, we discussed about how to generate posteriorgrams and how to speed-up spoken term search on posteriorgrams. Although the lower-bound based approaches accelerate the spoken term search by a factor of 300, there is still room for improvement. The lower-bounding idea is on the algorithmic level, while on the computational hardware level, we believe multi-threading dedicated platforms such as GPUs could bring more speed improvement. In the first part of this chapter, we will describe a carefully designed lower-bound estimate, and the K nearest neighbor DTW search algorithm on GPU parallel computing architecture[1]. In the second part of this chapter, we will present a parallel implementation of training Deep Belief Networks (DBNs), which could accelerate the DBN training significantly[2].

[1] Joint work with Kiarash Adl [145]
[2] Joint work with Nate Chodosh

6.1 Introduction

GPUs represent a category of specially designed graphical processors. The original purpose of GPUs is to facilitate fast construction and manipulation of images intended for output to a display, i.e. for video games. Since images can be decomposed into blocks of pixels, and each block can be processed independently, GPUs are designed to contain a highly parallel computing architecture. The result is more effective than CPUs that are designed for general purpose computing. An additional advantage of modern GPUs connected via a PCI Express interface is that they can access main memory faster than the CPUs themselves [133].

Graphical Processing Units (GPU) have been increasingly used for general purpose computing in the speech research community [14, 28, 133]. As a parallel computing architecture, GPUs are especially designed for dealing with intensive, highly parallel computations that are often encountered in speech tasks. Inspired by their success, in this chapter we propose a fast unsupervised spoken term detection system based on lower-bound K nearest neighbor (KNN) DTW search on GPUs. The lower-bound estimate as well as the KNN-DTW search are carefully designed to utilize GPU's parallel computing architecture. On the TIMIT corpus, a 55x speed-up of a spoken term detection task is achieved when compared to our previous CPU implementation [148], without affecting the detection performance. On a large, artificially created corpus, experiments indicate that the total computation time of the spoken term detection system is linear with corpus size. On average, searching a keyword on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour [145].

In Chapter 3, we used Deep Belief Networks (DBNs) to generate posteriorgrams for speech frames. Although DBN-based posteriorgrams have shown promising results on the spoken term detection task compared to the Gaussian posteriorgrams, the training of DBNs is much slower than the training of Gaussian Mixture Models (GMMs), especially for large speech corpora. For instance, Dahl et al. reported that it took 100 hours to train a DBN on 24 hours of speech data. For a larger corpus with 2,000 hours, it would take 50 days to train a context-dependent DBN [25]. To address this

problem, in this section, we introduce GPU computing to the training of DBNs. The phonetic classification experiment on the TIMIT corpus showed a speed-up of 36 for the pre-training and 45 for the back-propagation for a two-layer DBN trained on the GPU platform compared to the CPU platform.

6.2 Related Work

In order to facilitate the fast-growing application of GPUs to non-image processing tasks, NVidia has released the Compute Unified Device Architecture (CUDA) as a library providing APIs for using GPUs to perform general purpose computing [129]. When programming on GPUs with CUDA, CUDA defines a kernel as an abstraction of instances that can be run in parallel. Within each kernel instance, multiple threads can be issued to finish the task assigned to that instance. Therefore, CUDA can be viewed as a two-layer parallel computing architecture. In each CUDA run, one GPU core is assigned to run several kernel instances, and within each kernel instance, multiple threads are running simultaneously [9]. In order to achieve maximum benefit, it is important to take this architecture into consideration when adapting speech processing algorithms to the GPU/CUDA framework.

For example, Chong et al. in [20] proposed a large vocabulary speech recognition engine implemented on the CUDA/GPU framework with a significant speed-up observed. Their idea was to parallelize the Viterbi search on a linear lexicon-based recognition network. The linear lexicon-based network can be highly optimized for simple language models but cannot easily incorporate advanced language models. Phillips et al. proposed another parallel large vocabulary speech recognition system [99] based on parallel weighted finite state transducers (WFSTs). They used WFSTs to incorporate advanced language models but still suffered from buffer-synchronization issues which are not scalable with an increasing number of cores.

In speech acoustic modeling, Cardinal et al., Dixon et al. and Vanek et al. [14, 28, 133] have demonstrated how GPUs can be used to rapidly calculate acoustic likelihoods for large mixture models. Given a speech frame, their idea was to use

each GPU core to calculate the likelihood of the frame on one Gaussian component in the mixture model. A vectorized summation was then performed to obtain the weighted total log-likelihood. This approach significantly improves the efficiency of Gaussian likelihood calculation which would account for 60% of the total calculation in a Gaussian mixture model based speech system.

In Dynamic Time Warping (DTW) based speech processing, Sart et al. provided a DTW search algorithm for GPUs in [116], claiming that they could speed up the run time by up to two orders of magnitude. Their main contribution was to use each core to calculate one DTW alignment if multiple segments were given. Although the speed-up is quite notable, their aggressive use of each core imposes significant core synchronization overhead and is not well-suited to the recent design philosophy of the CUDA/GPU framework. In contrast, our parallel DTW implementation breaks down the entire algorithm into several parallel stages. Each stage is not very computationally intensive but can be fully parallelized. Therefore, by avoiding a large amount of core synchronization, a larger speed-up can be observed and the comparison results will be reported in the following sections.

Scanzio et al. explored an implementation of a complete Artificial Neural Network (ANN) training procedure for speech recognition [117]. Their main idea was to parallelize all the matrix/vector related operations in both the feed-forward and the back-propagation steps. Our parallel implementation of the Deep Belief Networks (DBNs) is inspired mainly from their work but differs in two important aspects. First, the design of our implementation considered the deep structure of the network. For instance, our implementation is able to handle 10+ layers with thousands of hidden units per each layer. Second, we implemented a completely parallel pre-training algorithm which is specifically designed for DBNs.

In addition to speech processing, the CUDA/GPU framework has been used in the computer vision and signal processing research communities as well. In computer vision, several image feature extraction and compression algorithm are implemented on GPUs and have obtained a speed-up of up to 280x over the CPU based algorithm [33]. In applications of video processing, the CUDA/GPU has been successfully used in

discrete Fourier transform on images [44], particle tracing, ray tracing [53], collision/object detection [94], etc. In signal processing, the FFT has been efficiently implemented by several research groups by using CUDA [24]. A recent report showed that the CUDA FFT implementation obtained a gain of 2.7x speed-up compared to a highly optimized CPU implementation running on the fastest CPU at the time [80].

All of these results have motivated us the use of CUDA/GPU framework in speeding up the spoken term detection system as well as the deep learning algorithms.

6.3 GPU Accelerated Lower-Bounded DTW Search

To avoid any notation problems, we first briefly review the spoken term detection framework discussed in previous chapters.

6.3.1 Spoken Term Detection using KNN-DTW

Given a spoken term Q with N frames, let $\vec{x}_1, \ldots, \vec{x}_N$ represent MFCCs for each speech frame. A D-mixture GMM G is trained on all N frames without using any labels. Subsequently, for each speech frame, $\vec{x}_i$, a posterior probability, $p_i^j = P(g_j|\vec{x}_i)$, can be calculated where g_j denotes j-th Gaussian component in GMM G. Collecting D posterior probabilities, each speech frame $\vec{x}_i$ is then represented by a posterior probability vector $\vec{p}_i = \{p_i^1, \ldots, p_i^D\}$ called a Gaussian posteriorgram. After representing the spoken term and speech documents using Gaussian posteriorgrams, an efficient KNN-DTW is used to find the top K nearest neighbor document matches. Specifically, if $Q = \{\vec{q}_1, \ldots, \vec{q}_M\}$ denotes the spoken term posteriorgram and $S = \{\vec{s}_1, \ldots, \vec{s}_N\}$ denotes a speech segment, an upper-bound envelope sequence U is calculated on Q, where $U = \{\vec{u}_1, \cdots, \vec{u}_M\}$, $\vec{u}_i = \{u_i^1, \cdots, u_i^D\}$ and $u_i^p = \max(q_{i-r}^p, \cdots, q_{i+r}^p)$. U can be viewed as a sliding-maximum on Q with window size r. Then, a lower-bound estimate of the DTW distance $\mathrm{DTW}(Q, S)$ is computed by

$$L(Q, S) = \sum_{i=1}^{l} d(\vec{u}_i, \vec{s}_i) \tag{6.1}$$

where $l = \min(M, N)$ and $d(\cdot)$ represents an inner product distance. If there are C speech segments, C lower-bound estimates are calculated and ranked. The KNN search starts from the segment with the smallest lower-bound estimate and performs DTW alignment. In Section 5.4, we proved that $L(Q, S) \leq DTW(Q, S)$, guaranteeing that the KNN search can stop when the current lower-bound estimate is greater than the DTW distortion score of the K^{th} best match [148].

6.3.2 System Design

In this section, we describe the proposed parallel spoken term detection system. The entire implementation consists of four CUDA kernels. The first two kernels correspond to the implementation of a parallel lower-bound estimate, while the last two kernels are used for parallel DTW calculations. Since we believe that the proposed method can be implemented in parallel architectures [123] other than CUDA, our focus will be on describing how to decompose the framework into parallel modules rather than using too many CUDA specific terms (e.g. grid, block). Moreover, since CUDA can run multiple kernel instances on one GPU core, in order to avoid confusion and without loss of generality, in the following description, we assume that each kernel instance runs on one GPU core. Figure 6-1 illustrates the parallel implementation.

Parallel Lower-bound Estimate

The parallel lower-bound estimate computation consists of two kernels. The first kernel computes the frame-wise inner-product distance, while the second kernel sums the inner-product distances. Specifically, suppose the upper bound envelope of the spoken term Q is U with length l and one of the speech segments is S_j with length l. By applying Eq. (6.1), the lower-bound estimate of Q and S_j is essentially the summation of the frame-by-frame inner-product distance of U and S_j. The summation calculation can be divided into two independent steps. In the first step, the inner-product distance d_i for each frame pair $\vec{u}_i$ and $\vec{s}_i$ is computed. The second step sums over l distances to obtain the lower-bound estimate.

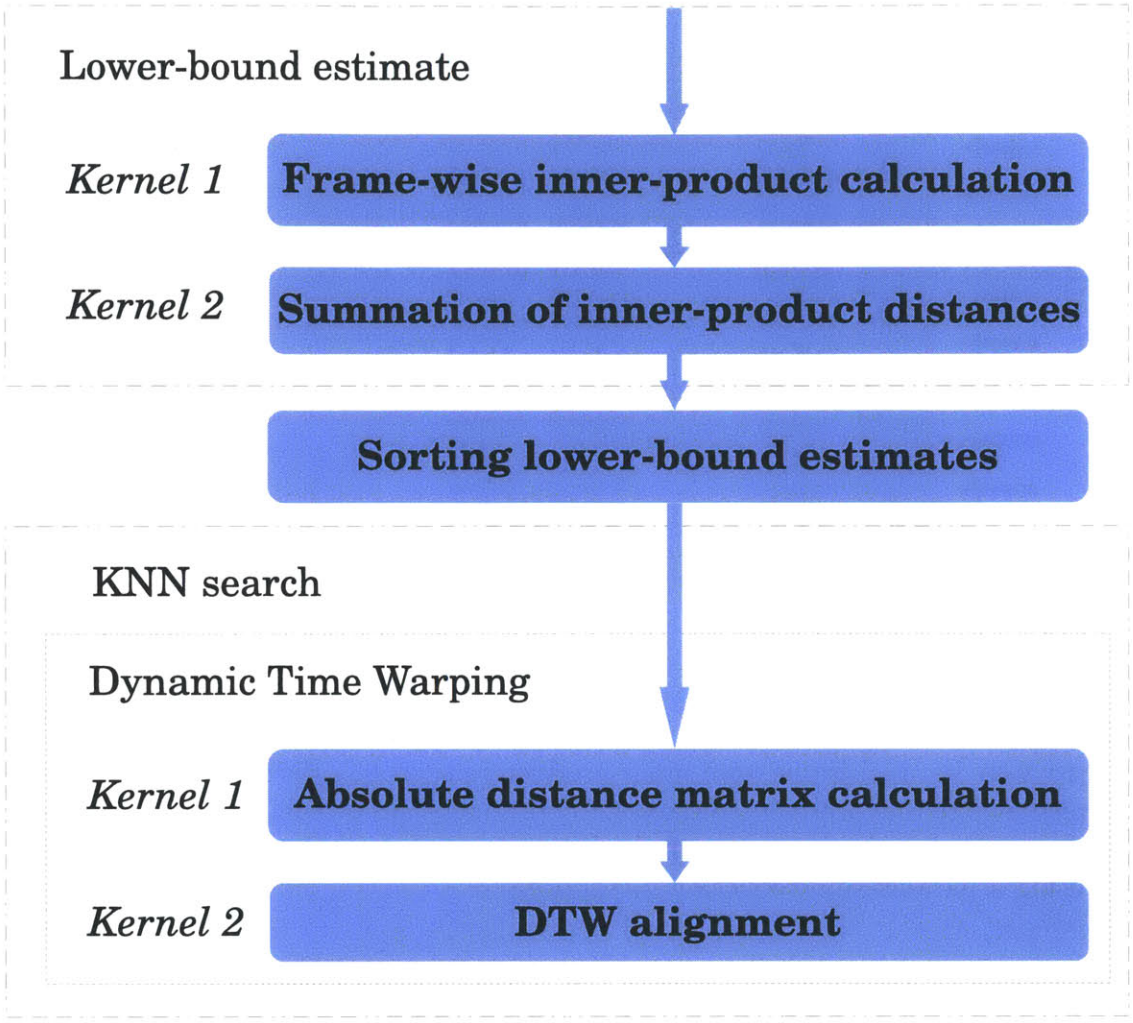


Figure 6-1: System flowchart of the parallel implementation of the lower-bound calculation and KNN-DTW search. The entire implementation consists of four CUDA kernels. The first two kernels correspond to the implementation of a parallel lower-bound estimate, while the last two kernels are used for parallel DTW calculations.

In order to maximize GPU efficiency, two separate kernels are designed for each step respectively. The first kernel takes two posteriorgram vectors as input, and outputs the inner-product distance of these two vectors. Since each kernel can run multiple threads simultaneously, the vector-wise inner-product calculation is further decomposed into element-wise products. Since each posteriorgram is a D-dimensional vector, the inner-product can be written as

$$d(\vec{u}_i, \vec{s}_i) = \sum_{k=1}^{D} u_i^k \cdot s_i^k. \tag{6.2}$$

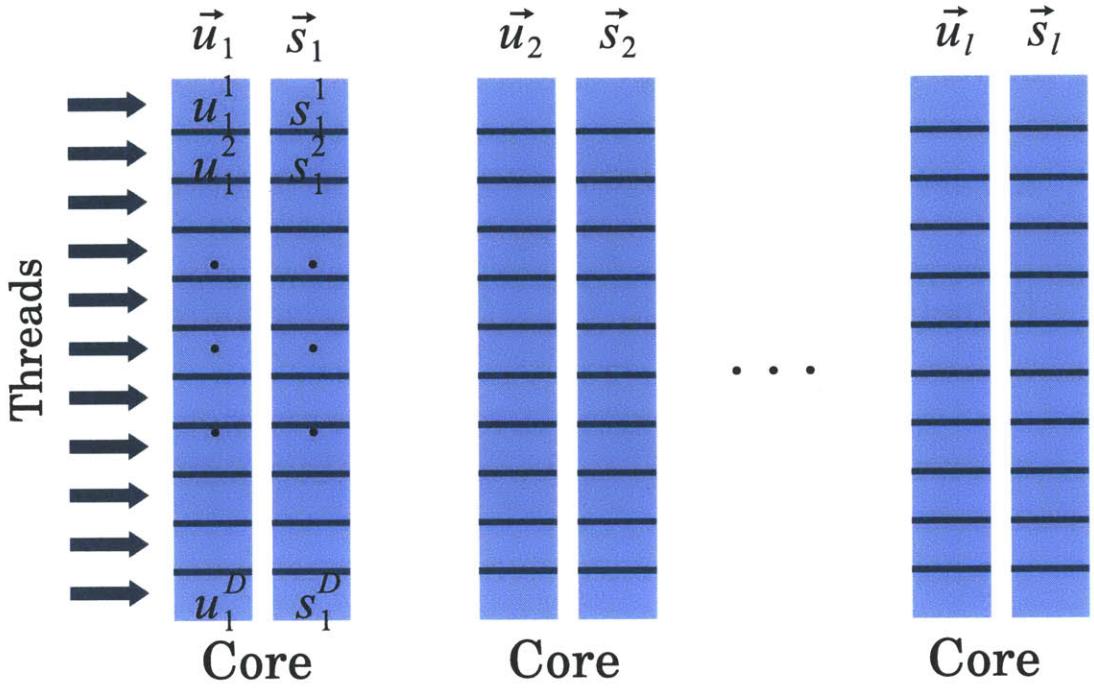


Figure 6-2: Parallel frame-wise inner-product calculation. Each thread in the kernel corresponds to one product, so that the D products can be computed simultaneously. After the D threads finish computing, the summation of D products is parallelized by using a parallel-sum-reduction method [9].

Therefore, as illustrated in Figure 6-2, each thread in the kernel corresponds to one product, so that the D products can be computed simultaneously. After the D threads finish computing, the summation of D products is parallelized by using a parallel-sum-reduction method [9]. When the first kernel finishes computing, l inner-product distances are stored in on-device memory. The second kernel then launches to sum over l inner-products. This summation is also performed via the parallel-sum-reduction technique.

Sorting and KNN Search

After the lower-bound estimates have been computed, each speech segment S_j is associated with a lower-bound distance estimate LB_j. Since the number of the target speech segments can potentially be very large, a CUDA library called Thrust is used to perform high-performance parallel sorting of these lower-bound estimates [52].

When sorting is complete, the KNN search starts from the speech segment with the smallest lower-bound estimate and calculates the actual DTW distance. In theory, in order to obtain K best matches, at least K DTW alignments need to be performed [148]. Therefore, instead of going through the sorted speech segments one by one in the CPU implementation and calculating one DTW alignment at a time, K speech segments are considered and K DTW alignments are performed simultaneously in the GPU implementation. If the current best DTW matching score is less than the lower-bound value of the $(K+1)^{th}$ speech segment, the KNN search terminates and outputs the best K matches. Otherwise, the next $K/2$ speech segments are considered and $K/2$ new DTW alignments are computed. The search iterates until the termination condition is met. In summary, the parallel KNN search strategy is to aggressively run K DTW alignments in the first round of search, and subsequently to perform $K/2$ DTW alignments to be more conservative and avoid wasting DTW calculations.

Parallel DTW

In order to maximize the use of the parallel computing architecture, DTW calculations are divided into two steps, illustrated in Figure 6-3. In the first step, given two posteriorgrams Q and S_j, an absolute distance matrix $A(Q, S_j)$ (which uses the inner-product as local distance metric) is computed. The second step performs the DTW alignment on A and outputs the distortion score. A kernel is designed for each respective step.

Specifically, if Q and S_j have the same length l, the size of the absolute distance matrix $A(Q, S_j)$ is $l \times l$. Each instance of the first kernel computes a single element in A. Note that because of the DTW beam width constraint, only valid elements are computed. Since the inputs to the first kernel are two D-dimensional posteriorgram vectors, in a manner similar to the first kernel in the parallel lower-bound estimate step, each thread in each kernel instance computes a scalar product so that D products can be performed simultaneously. Since the DTW alignment is a 2-D sequential dynamic search, it cannot be completely parallelized. Therefore, in the second kernel,

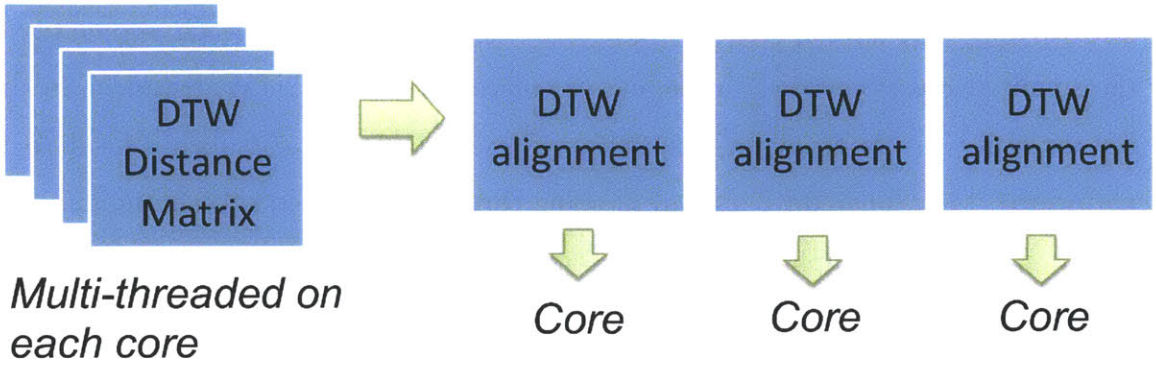


Figure 6-3: Parallel DTW. The first step performs multi-threaded distance matrix calculation. The second step computes multiple DTW alignments simultaneously.

each thread in each kernel instance corresponds to the entire warping process given an absolute distance matrix A.

In Sart et al. [116], the DTW GPU implementation was to use each thread in each kernel instance to compute the absolute distance matrix A as well as the DTW alignment. In order to compare the two approaches, we implemented both methods and report the results of the comparison in the next section.

6.3.3 Evaluation

Spoken Term Detection Task

We chose to duplicate previously reported spoken term detection experiments that we have performed on the TIMIT corpus using posteriorgram-based DTW search in Chapter 3. The same experiment configurations were used and the spoken term list was fixed as in Section 3.6.1. Performance was still measured by the average equal error rate (EER): the average rate at which the false acceptance rate is equal to the false rejection rate. All spoken term detection experiments were performed on a computer equipped with an Intel® Quad 6600 Processor (2.66 GHz, 4 cores), 8 GB RAM and a NVidia® GTX 580 graphic card (512 GPU cores with 1.5 GB on-board RAM). The graphic card cost 500 USD in June, 2011.

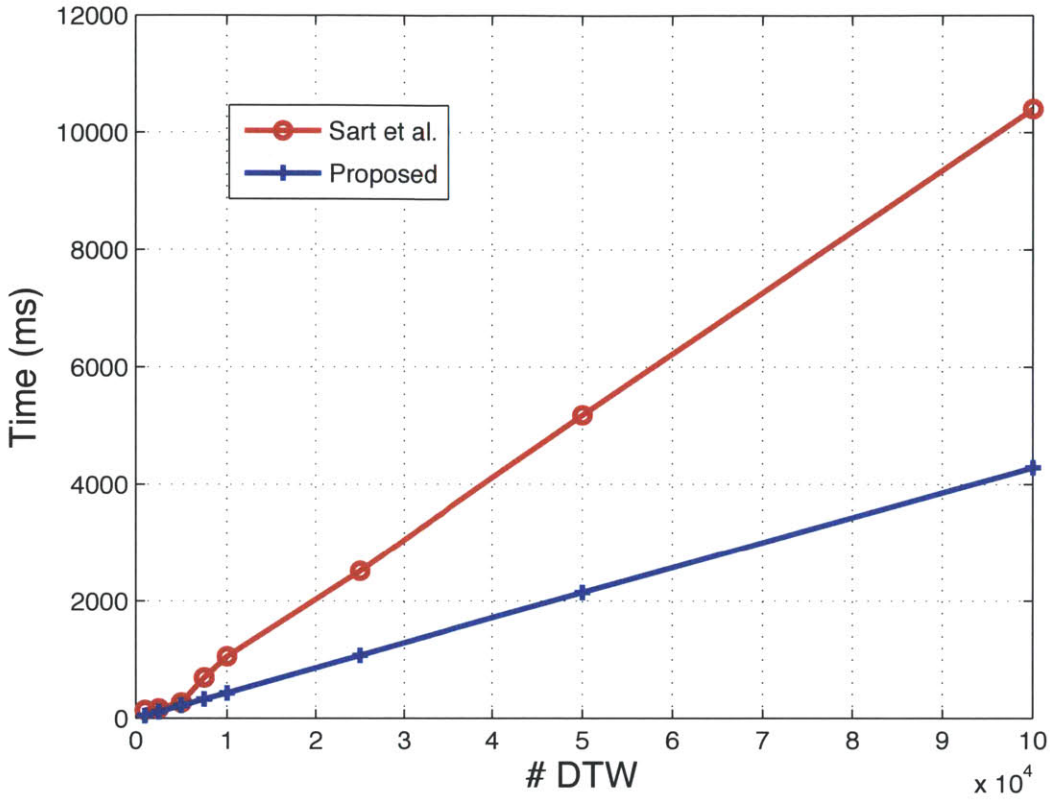


Figure 6-4: Comparison of the computation time for the two different parallel DTW implementations. The x-axis shows the number of DTWs computed, the red curve represents the time consumed for the method proposed by Sart et al. [116], while the blue curve shows the method used in the current implementation.

Experimental Results

In order to validate the correctness of the GPU implementation of the spoken term detection algorithms, we first compared the detection results with the corresponding CPU implementation. In addition to the intermediate outputs from the four kernels being the same, the overall detection EER remained the same as has been previously reported in Section 3.6.1, with an EER equal to 16.4%.

Figure 6-4 shows a comparison of the computation time for the two different aforementioned parallel DTW implementations. In the figure, the x-axis shows the number of DTWs computed, the red curve represents the time consumed for the method proposed by Sart et al. [116], while the blue curve shows the method used in

the current implementation. The figure indicates that both methods consumed similar time when computing less than 5,000 DTWs. However, our method outperformed the previous method by a factor of 2 in terms of the running time when more than 5,000 DTWs computations were required. We believe the reason for the greater efficiency is due to the time saved in the absolute distance matrix calculation. In the current version the entire matrix was decomposed into individual elements for maximum parallelization, while the prior method computed the full matrix in each thread without parallelization. As the number of DTW calculation increases, the amount of computation time saved becomes more apparent.

Figure 6-5 shows the average computation time needed for searching one keyword using the proposed spoken term detection system as a function of corpus size. Since the original TIMIT test corpus only contains 48 minutes of speech, in order to measure the computation time on a larger corpus, we replicated the TIMIT test corpus by adding small amounts of random noise into each speech frame. The replicated TIMIT test corpus contained 1,000 hours of speech. In the figure, the black curve represents the total time consumed, the blue curve represents the time consumed by the lower-bound estimate, the green curve represents the time used by sorting all lower-bound values, and the red curve represents the time consumed by the KNN-DTW search. The results indicate that the KNN-DTW search occupies nearly 80% of the total running time, which we believe is due to the difficulty in parallelizing the DTW algorithm. It is encouraging to observe that the total computation time grows linearly with the corpus size. For 1,000 hours of speech, searching a keyword requires 40 minutes on a single desktop computer, which translates to 2.4 seconds/corpus hour. Note that with multiple desktops and GPUs, the entire process could be further parallelized with each GPU searching a subset of the overall corpus.

In terms of computation time for searching the TIMIT test corpus, the original CPU-based approach took, on average, 120 seconds per keyword, while the GPU-based approach takes, on average, only 2.2 seconds per keyword, which translates to a speed-up factor of 55.

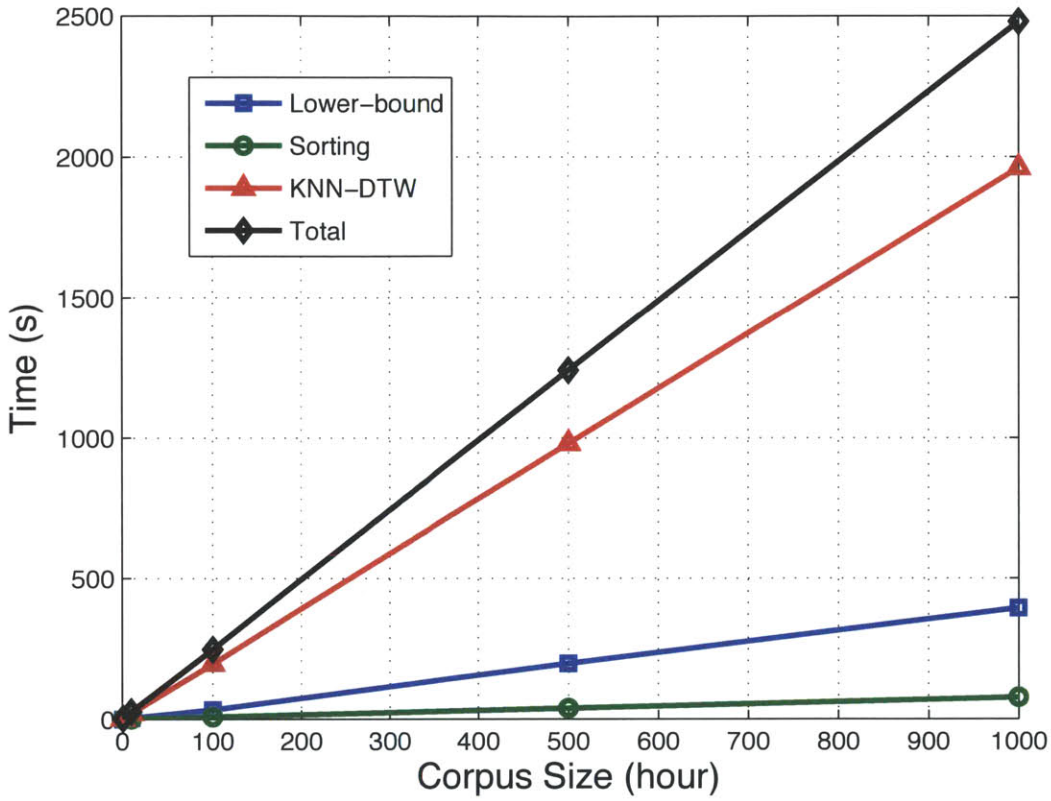


Figure 6-5: The average computation time needed for searching one keyword using the proposed spoken term detection system as a function of corpus size. Since the original TIMIT test corpus only contains 48 minutes of speech, in order to measure the computation time on a larger corpus, we replicated the TIMIT test corpus by adding small amounts of random noise into each speech frame. The replicated TIMIT test corpus contained 1,000 hours of speech. The black curve represents the total time consumed, the blue curve represents the time consumed by the lower-bound estimate, the green curve represents the time used by sorting all lower-bound values, and the red curve represents the time consumed by the KNN-DTW search.

6.4 GPU Accelerated Deep Learning

Since most of the computation in DBN training involves matrix-matrix operations which have a high degree of fine grain parallelism, the use of GPUs is particularly suited for this task. In Chapter 4, we derived the detailed steps for performing the pre-training and the back-propagation for a Restricted Boltzmann Machine (RBM) which is the building block of a DBN. To better fit the GPU computing environment, a matrix form of pre-training and back-propagation needs to be developed. Since the stochastic gradient decent (SGD) update method is used both for the pre-training and the back-propagation, we assume that a n by d data matrix D is the input of our training algorithm, where n is the number of data samples in a batch and d is the number of dimensions of each data vector.

6.4.1 Pre-training

For pre-training, we start from the forward pass of a single RBM with the input D. The forward pass can be written as

$$P_{pos} = \text{sigmoid}\left\{-\begin{bmatrix} D & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} W \\ B_h \end{bmatrix}\right\} \tag{6.3}$$

where P_{pos} is the forward probability given the visible-hidden weight matrix W and the hidden bias vector B_h. In order to convert the forward probability to a binary activation of the hidden units, a random matrix R can be used as

$$H_a = \begin{bmatrix} P_{pos} > R \end{bmatrix} \tag{6.4}$$

where H_a represents the activation of the hidden units and R is a n by h matrix randomly drawn from a uniform distribution with range $[0, 1]$. The hidden unit activates $H_a^{ij} = 1$ if $P_{pos}^{ij} > R^{ij}$. Then, the backward reconstruction can be written as

$$D_r = \text{sigmoid}\left\{-\begin{bmatrix} H_a & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} W^\top \\ B_v \end{bmatrix}\right\} \tag{6.5}$$

where B_v is the visible bias vector. The backward probability P_{neg} can be written as

$$P_{neg} = \text{sigmoid}\left\{-\begin{bmatrix} D_r & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} W \\ B_h \end{bmatrix}\right\} \tag{6.6}$$

With the original data D and the reconstructed data D_r, the reconstruction error is defined by

$$RE_{\text{error}} = \text{CSUM}\left\{\text{CSUM}\left[(D - D_r)^2\right]\right\} \tag{6.7}$$

where CSUM returns a row vector of sums of each column and the square is an element-wise operator. The corresponding derivatives are

$$\Delta W = \frac{\left[D^\top \cdot P_{pos} - D_r^\top \cdot P_{neg}\right]}{n} \tag{6.8}$$

$$\Delta B_v = \frac{\left[\text{CSUM}(D) - \text{CSUM}(D_r)\right]}{n} \tag{6.9}$$

$$\Delta B_h = \frac{\left[\text{CSUM}(P_{pos}) - \text{CSUM}(P_{neg})\right]}{n} \tag{6.10}$$

Note that here we omitted the weight decay on the visible-hidden weights for clarification.

6.4.2 Back-propagation

The back-propagation method can be also written in the matrix form. We assume the use the same data input D and operate a DBN with one hidden layer (RBM) and one softmax layer. First, the forward probability can be calculated as

$$P_{pos} = \text{sigmoid}\left\{-\begin{bmatrix} D & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} W \\ B_h \end{bmatrix}\right\} \tag{6.11}$$

where W denotes the visible-hidden weights and B_h denotes the hidden bias vector

as the pre-training. The prediction matrix can be computed as

$$T = \exp\left(P_{pos} \cdot W_{\text{softmax}}\right) \tag{6.12}$$

where W_{softmax} represents the weights of the softmax layer. Then, we normalize the prediction matrix T to make it act as a probability matrix T_n. Therefore, the error signal for the softmax layer is

$$E_{\text{softmax}} = T_n - L \tag{6.13}$$

where $L_{ij} = 1$ if the class label of the i-th data vector is j and $L_{ij} = 0$ elsewhere. By propagating the errors into the hidden layer, the error signal for the hidden layer is

$$E = (E_{\text{softmax}} \cdot W_{\text{softmax}}^{\top}) \circ P_{pos} \circ (\mathbf{1} - P_{pos}) \tag{6.14}$$

where $\circ$ is the element-wise matrix product. Finally, with the error signals, the derivatives are

$$\Delta W_{\text{softmax}} = P_{pos}^{\top} \cdot E_{\text{softmax}} \tag{6.15}$$

$$\Delta W = D^{\top} \cdot E \tag{6.16}$$

The standard parallel matrix-matrix operations are based on CUBLAS [23] library provided by NVidia. Other operations such as element-wise operations are implemented in the SLS-DBN toolkit.

6.4.3 Evaluation

To demonstrate the speed-up brought by using GPU for training DBNs, we performed a phonetic classification task on the TIMIT corpus. The phonetic classification task is to classify 48 phone classes by using 230k speech segments. The features were produced by applying Neighborhood Component Analysis (NCA) [42] to the

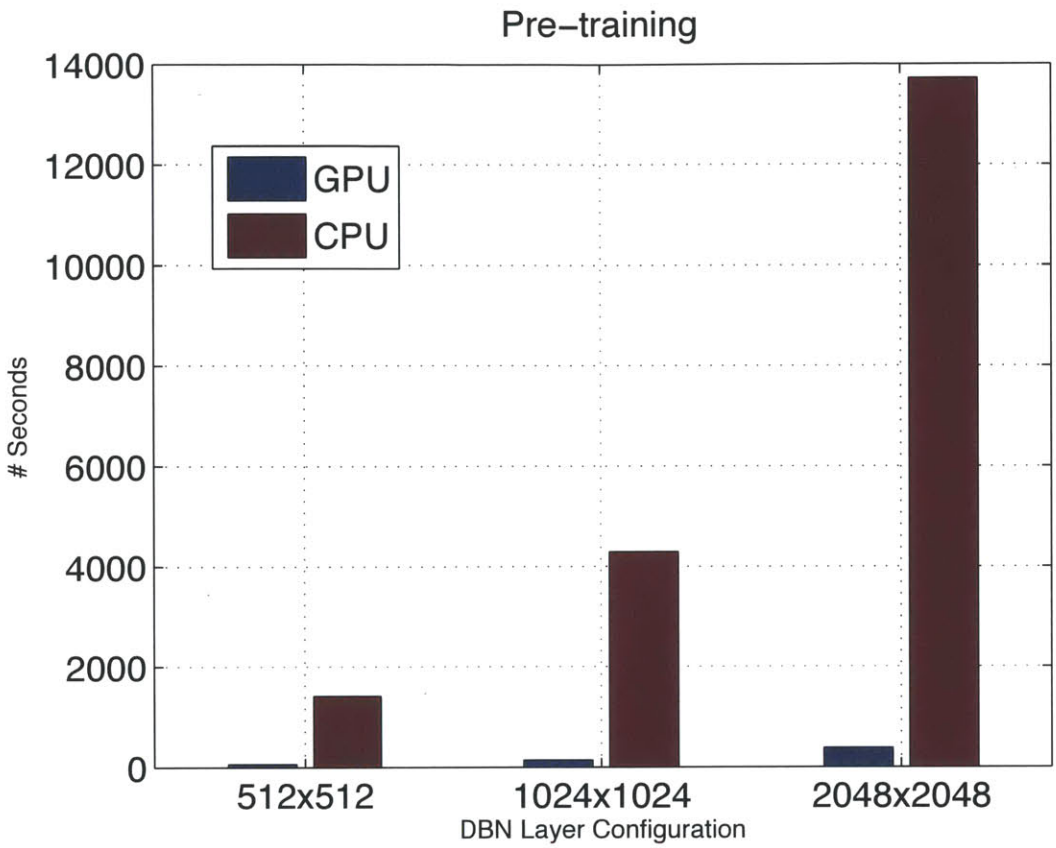


Figure 6-6: Time consumed for the full pre-training on the TIMIT phonetic classification task with different DBN layer configurations. The pre-training was set to stop at the 50^{th} iteration for each layer.

averaged MFCCs within each speech segments, resulting in a 61-dimensional feature vector for each speech segment [18]. DBNs with two hidden layers with different number of hidden units were trained on both CPU and GPU platforms. A softmax layer with 48 output units was attached to the DBNs to perform the classification task. The CPU DBN training toolkit was implemented by using the Intel Math Kernel Library (MKL) to maximize the multi-threading performance. The GPU DBN training toolkit was implemented by using the CUDA 4.2 release. The CPU platform used was E3600 4-core Xeon with Hyper-threading [125] enabled. The GPU platform was NVidia Tesla C2070 with 448 cores and 6GB onboard shared memory. The total size of speech feature vectors is around 60 MB, which can be loaded onto the GPU card at one time.

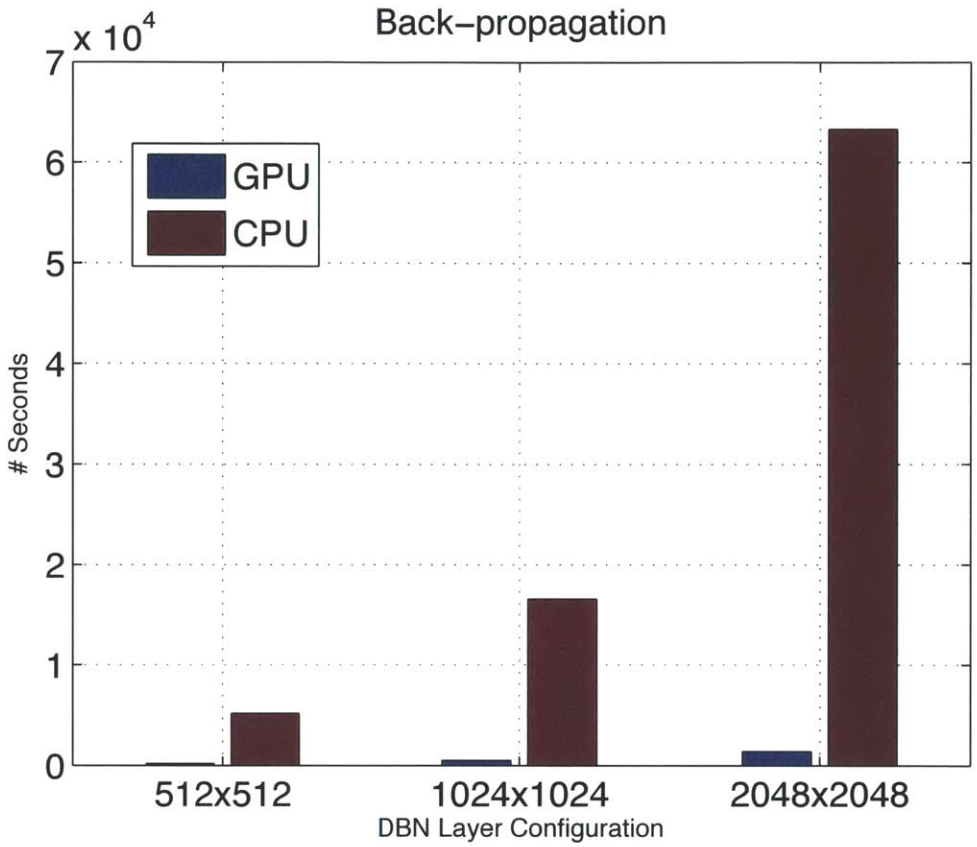


Figure 6-7: Time consumed for the full back-propagation on the TIMIT phonetic classification task with different DBN layer configurations. The back-propagation was set to stop at the 50^{th} iteration.

Figure 6-6 shows the time consumed for the full pre-training with different DBN layer settings. The pre-training was set to stop at the 50-th iteration for each layer. The GPU implementation shows a significant speed-up compared to the CPU implementation. Moreover, with an increasing number of hidden units, the GPU implementation shows larger speed-up compared to the CPU implementation. With 2048 hidden units for each layer, the GPU implementation is able to speed-up the full pre-training by a factor of 36.

Figure 6-7 illustrates the time consumed for the back-propagation with different DBN layer settings. The back-propagation was set to stop at the 50-th iteration. Compared to the pre-training, the GPU implementation is more effective in the back-propagation stage. With 2048 hidden units, the GPU implementation is able to

achieve a speed-up of 45. Similar to the pre-training experiment, the GPU implementation tends to obtain more speed-ups when increasing the number of hidden units.

Besides the TIMIT phonetic classification task, the DBN GPU implementation was used for all DBN related experiments in this thesis. We observed that with deeper layers and more hidden units, the GPU implementation could offer significant speed-ups against the CPU implementation. Depending on different tasks, the speed-ups could range from 20 to 60.

6.5 Summary

In the first part of this chapter, we described a GPU parallelized implementation of an unsupervised spoken term detection system based on the lower-bound KNN-DTW search. The original lower-bounded DTW algorithm was carefully re-designed to fit the GPU's parallel computing architecture. In a spoken term detection task using the TIMIT corpus, a 55x speed-up was achieved compared to our previous CPU-based implementation without affecting the detection performance. On artificially replicated data, experimental results indicated that the total running time of the entire spoken term detection system grows linearly with corpus size. On average, searching a spoken term on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour.

In the second part of this chapter, we presented a GPU parallel implementation for training deep belief networks (DBNs). By deriving update formulas in the matrix form, the DBN pre-training and back-propagation can fully utilize the GPU's parallel programming architecture. The phonetic classification experiment on the TIMIT corpus showed a speed-up of 36 for pre-training and 45 for back-propagation for a two-layer DBN trained on the GPU platform compared to the CPU platform.

Chapter 7

Conclusions and Future Work

In this chapter, we summarize the main contributions of this thesis and discuss future work.

7.1 Summary and Contributions

The work discussed in this thesis is motivated by the challenge of searching and extracting useful information from speech data in a completely unsupervised setting. The research is motivated by the fact that in many real world speech processing problems, obtaining annotated data is not cost and time effective. We therefore ask how much can we learn from speech data without any transcription. To address this question, in this thesis, we chose the query-by-example spoken term detection as a specific scenario to demonstrate that this task can be done in the unsupervised setting without any annotations. To build the unsupervised spoken term detection framework, we contributed three main techniques to form a complete working flow; 1) two posteriorgram-based speech representations which enable speaker-independent, and noisy spoken term matching, 2) two lower-bounding based methods for Dynamic Time Warping (DTW) based pattern matching algorithms and 3) the parallel implementation of the lower-bounded DTW search algorithm, and training of large Deep Belief Networks (DBNs) on Graphical Processing Units (GPUs).

In Chapter 3, we presented the spoken term detection system and spoken term

discovery system using unsupervised Gaussian posteriorgrams. The Gaussian posteriorgram generation method and the associated DTW based search algorithm were described. The core idea is to train a Gaussian mixture model without using any supervised annotation, and represent each speech frame by calculating a posterior distribution over all Gaussian components. A modified DTW matching algorithm can be used to evaluate the similarity between two speech segments represented by Gaussian posteriorgrams in terms of an inner-product distance. The entire process is completely unsupervised and does not depend on speakers or languages. The experimental results demonstrated the viability of using Gaussian posteriorgrams for both query-by-example based spoken term detection and speech pattern discovery in a multi-speaker environment.

In Chapter 4, we presented a spoken term detection method based on posteriorgrams generated from Deep Belief Networks (DBNs). The proposed representation can be easily adapted to work in both semi-supervised and unsupervised training conditions. Spoken term detection experiments on the TIMIT corpus showed a 10.3% relative improvement compared to our previous Gaussian posteriorgram framework in the unsupervised condition. In the semi-supervised setting, the detection performance using the DBN posteriorgram can achieve a comparable performance to fully supervised training when using only 30% of the labeled data.

In Chapter 5, we presented two lower-bound estimates and their corresponding fast approximation for DTW-based methods that use an inner-product distance metric such as for a posteriorgram representation. Given a spoken term posteriorgram and a test posteriorgram, the lower-bound is obtained by calculating the inner-product distance of the upper envelope of the spoken term posteriorgram against the test posteriorgram. The lower-bound underestimates the actual DTW score between the spoken term and test posteriorgrams, which provides an efficient pruning mechanism for KNN search. Based on the experimental results in a spoken term detection task, the KNN-DTW search can eliminate 89% of DTW calculations for the tight lower-bound estimate, and achieve another 28% speedup for the Piecewise Aggregation Approximated lower-bound estimate. Since both lower-bound estimates guarantee

no false dismissals (i.e. admissible search), the system attains the same spoken term detection error rate as the baseline system without pruning.

In the first part of Chapter 6, we described a GPU parallelized implementation of an unsupervised spoken term detection system based on lower-bound KNN-DTW search. The spoken term detection algorithm is carefully re-designed to fit the GPU's parallel computing architecture. In a spoken detection task using the TIMIT corpus, a 55x speed-up is achieved compared to our previous CPU-based implementation without affecting the detection performance. On artificially replicated data, experimental results indicate that the total running time of the entire spoken detection system grows linearly with corpus size. On average, searching a spoken term on a single desktop computer with modern GPUs requires 2.4 seconds/corpus hour. In the second part of Chapter 6, we presented a GPU parallel implementation for training DBNs. By deriving update formulas in the matrix form, the DBN pre-training and back-propagation can fully utilize the GPU's parallel programming architecture. The phonetic classification experiment on the TIMIT corpus showed a speed-up of 36x for the pre-training and 45x for the back-propagation for a two-layer DBN trained on the GPU platform as compared to the CPU platform.

7.2 Future Work

Although experiments in this thesis showed promising results for the query-by-example spoken term detection task, there is still much room for improvement for each of the three main techniques. Detailed discussion are presented in the following sections.

7.2.1 Posteriorgram Generation

In this thesis, we showed that DBNs generate better posteriorgrams compared to GMMs. DBN-based models belong to the category of neural network models which are not fully Bayesian. One future direction is to develop a fully Bayesian probabilistic model to produce posteriorgrams. One candidate is to use the Dirichlet Process

Mixture Model (DPMM). A Dirichlet Process Mixture Model (DPMM) can be viewed as a sequence of finite mixture models where the number of mixture components is taken to infinity [89, 103, 45, 55]. In contrast to the conventional GMM which needs to select a proper number of mixture components, the DPMM assumes an infinite number of mixture components which are distributed according to a Dirichlet process. This soft assignment of the number of mixture components helps automatically learn the best model configuration for the speech data without any supervision.

To be specific, a DPMM can be represented by the following two sampling process.

$$\begin{aligned} \phi_i \mid G &\sim G \\ x_i \mid \phi_i &\sim F(\phi_i) \end{aligned} \tag{7.1}$$

where G is distributed according to a Dirichlet process, ϕ_i represents parameters of i-th Gaussian component, $F(\phi_i)$ is the actual Gaussian distribution function and x_i represents the i-th observation. To model an observation x_i, a Gaussian distribution $F(\phi_i)$ is randomly sampled from the base distribution G. The construction of sampling ϕ_i from G can be done by the stick-breaking process [35, 118] or the Chinese restaurant process [10, 1]. In contrast to the finite case, every time an x is observed, we can select an existing mixture component to model x, but there is also a small chance to create a new mixture component for x with probability inversely proportional to the number of existing mixture components. In practice, a DPMM training procedure can stop when posterior probabilities of all observations on a newly created mixture component are less than a threshold [70]. This stopping criterion represents an approximation of the infinite DPMM for practical use while not losing too much modeling power.

In our case, if we can apply the DPMM for posteriorgram generation, we do not need to set a specific number of mixture components for each speech corpus/task. The learning algorithm itself can determine the best number of mixture components needed. Then, for each speech frame, the posteriorgram can be computed in the conventional way using Eq.3.1. One possible challenge is that the current DPMM learning algorithm requires Markov Chain Monte Carlo (MCMC) sampling which

is not efficient for large speech corpora [126]. We need to derive a good way to organize speech data to speed-up the training process. The recent development of the variational approximation of DPMM learning [11, 70, 127] is also worth investigating.

7.2.2 Letter-to-Posteriorgram HMM

If some word level annotations are provided, we could establish alignments from labeled words to refined posteriorgrams. Since the letter sequences are already embedded in the word level labels, a Letter-to-Posteriorgram Hidden Markov Model (LP-HMM) can be trained to automatically align HMM states to letters. Inspired by the HMM text-to-speech framework [29, 128], using LP-HMM, a typed keyword (a letter sequence) can be represented by a series of posteriorgram frames by sampling the mean from the HMM state sequence. In other words, given a typed keyword input, we synthesize its posteriorgram representation by running a state traverse in LP-HMM according to the letter sequence in the keyword. After this step, since a typed keyword is converted to a series of posteriorgrams, the DTW algorithm can then be used to search the keyword posteriorgrams against the entire test data. If this idea worked, it would be a good extension to our current unsupervised query-by-example keyword detection system when there are some labeled data available. The training of LP-HMM for query-by-typing keyword search is illustrated in Figure 7-1.

7.2.3 Posteriorgram Evaluation

A recent work by M. Carlin et. al [15] presented a method of evaluating the quality of posteriorgrams generated by a supervised phonetic recognizer. It evaluates phonetic posteriorgrams without coupling to any specific task. The results showed that their metrics for posteriorgrams are closely related to phone recognition accuracy.

Specifically, their evaluation method requires word pair alignment on a fully labeled dataset. Consider the following four sets of word pairs in a speech corpus.

- C_1: same word spoken by same speaker
- C_2: same word spoken by different speakers

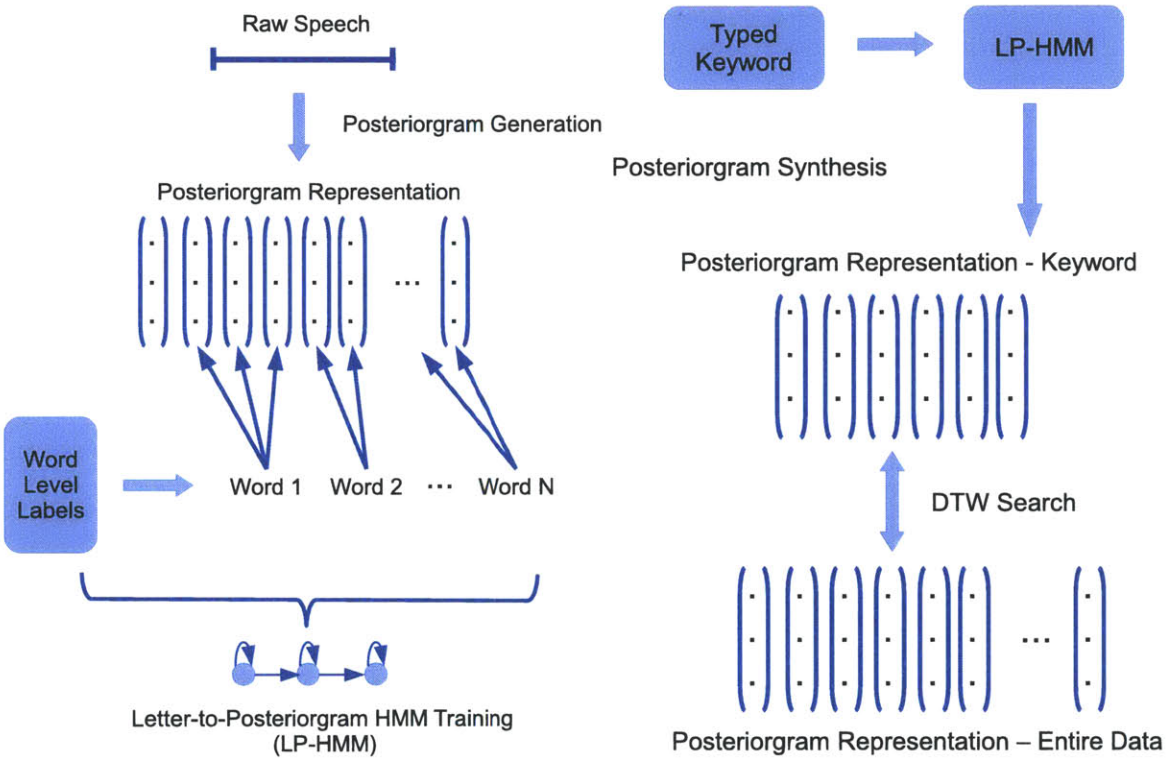


Figure 7-1: The left figure shows the workflow of training LP-HMM. Using a posteriorgram representation. Since the letter sequences are already embedded in the word level labels, a Letter-to-Posteriorgram Hidden Markov Model can be trained to automatically align HMM states to letters. The right figure shows query-by-typing keyword search using trained LP-HMM. A typed keyword (a letter sequence) can be represented by a series of posteriorgram frames by sampling the mean from the HMM state sequence. Since a typed keyword is converted to a series of posteriorgrams, the DTW algorithm can be then used to search the keyword posteriorgrams against the entire test data.

- C_3: different word spoken by same speaker
- C_4: different word spoken by different speakers

A word pair $(w_i, w_j) \in C_k, k = 1 \cdots 4$ represents a basic unit for evaluation. The similarity of any word pair can be measured using DTW alignment. The total similarity of a word pair set is the summation of DTW similarities normalized by the size of the set. If $S(C_k)$ denotes the total similarity of the word pair set k, it is expected that a good posteriorgram representation of speech frames would result in $S(C_1) \ll S(C_2) \ll S(C_3) \ll S(C_4)$. Instead of using normalized summation of similarities for each set, we can also compare the DTW similarity distribution for each word pair set. KL divergence can be applied to calculate the distance between DTW similarity distributions. Using this method, a quantitative measurement could be derived for examining the quality of posteriorgrams. We would like to investigate this evaluation method for our unsupervised Gaussian posteriorgrams as well as posteriorgrams generated by improved models discussed above.

7.2.4 Lower-Bounding for Spoken Term Discovery

Using the lower-bounded DTW search, we presented methods for fast spoken term matching on posteriorgram representation. One future direction is to apply the same lower-bound idea to the spoken term discovery task, which essentially replaces the one versus many spoken term matching with the many versus many spoken term discovery matching. To discover recurring patterns (i.e. frequently used words or short phrases) in two speech utterances, the original Segmental Dynamic Time Warping (S-DTW) approach requires at least $O(MN^2)$ time, where M is the maximum length between two utterances and N is the minimum warping length. As for the spoken term detection task, for a large speech corpus, the original S-DTW algorithm has a considerable computational burden. Therefore, it is worth investigating if a similar lower-bounded estimate idea could apply to the S-DTW search. For instance, if the lower-bound estimate exists, a linear-time fast-match could be used as the first pass to find highly possible regions for recurring patterns. Then, in the second pass, an exact

DTW could be performed to refine the first pass results and output the final recurring pattern candidates. Moreover, it would be interesting to investigate whether the lower-bound idea can be combined with the approximation based method proposed by Jansen et al. in [59, 60].

7.2.5 Hierarchical Parallel Implementation

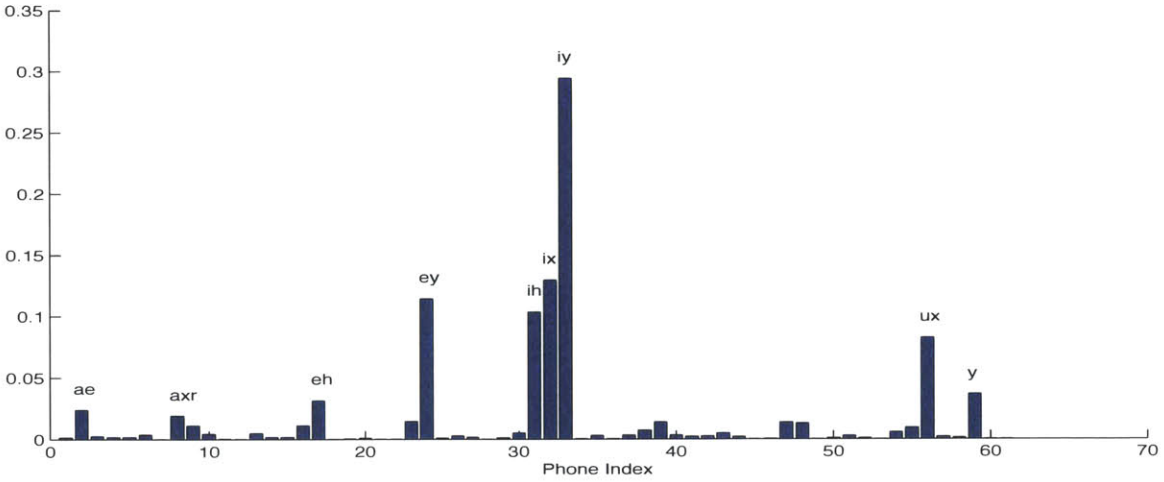
The current parallel implementation for lower-bounded DTW search and DBN training can be viewed as device level parallelization. Both implementations are multi-threaded for a specific device (i.e. a GPU card) on a single computer. One possible extension is to consider a hierarchical multi-CPU and multi-GPU parallel structure. For example, consider a compute farm with 100 CPU cores and 512 x 100 GPU cores (i.e. each CPU core is equipped with a GPU card with 512 cores). An entire speech task can be first divided into 100 sub-tasks. Each sub-task is then multi-threaded on 512 GPU cores. This hierarchical parallel structure can utilize advantages from both host level parallelization and device (instruction) level parallelization, which we believe, can further speed up the proposed spoken term search and DBN training algorithms.

Appendix A

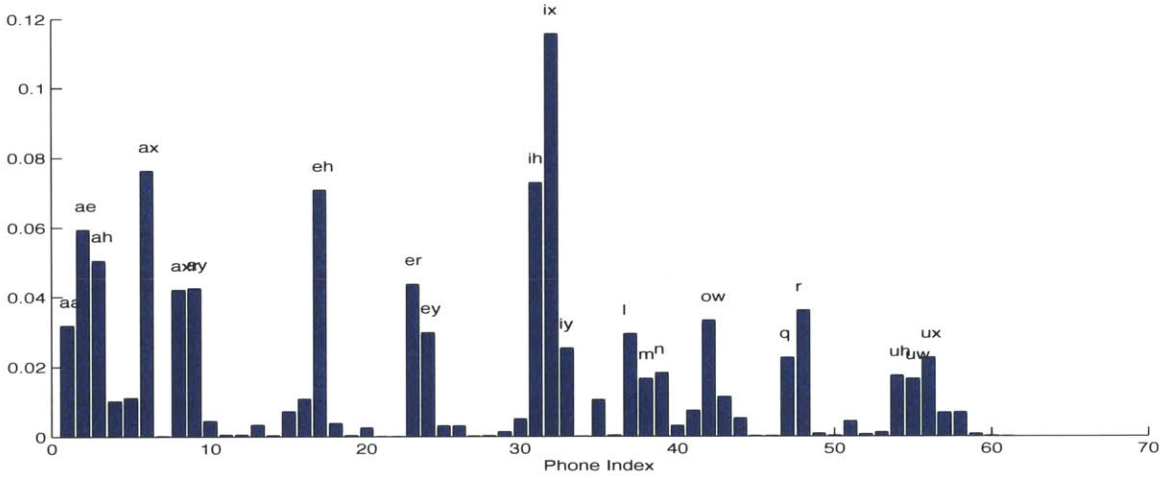
Phonetic Distribution of Gaussian Components

In Gaussian posteriorgram generation, we assume that through unsupervised GMM training, each Gaussian component can be viewed as a self-organizing phonetic class. To validate this assumption, a phonetic histogram analysis is applied to visualize the underlying acoustically meaningful information represented by each Gaussian component. Specifically, after unsupervised GMM training, each speech frame can be decoded and represented by Gaussian posteriorgrams. Since the TIMIT dataset provides a time-aligned phonetic transcription, for each Gaussian component, by summing all posterior probabilities of each phone, we can calculate a distribution over phones belonging to that Gaussian component. By drawing a bar for every TIMIT phone for one Gaussian component, we can have a histogram of the underlying phonetic distribution of that Gaussian component. In the following sections, we group Gaussian components by phone manner classes.

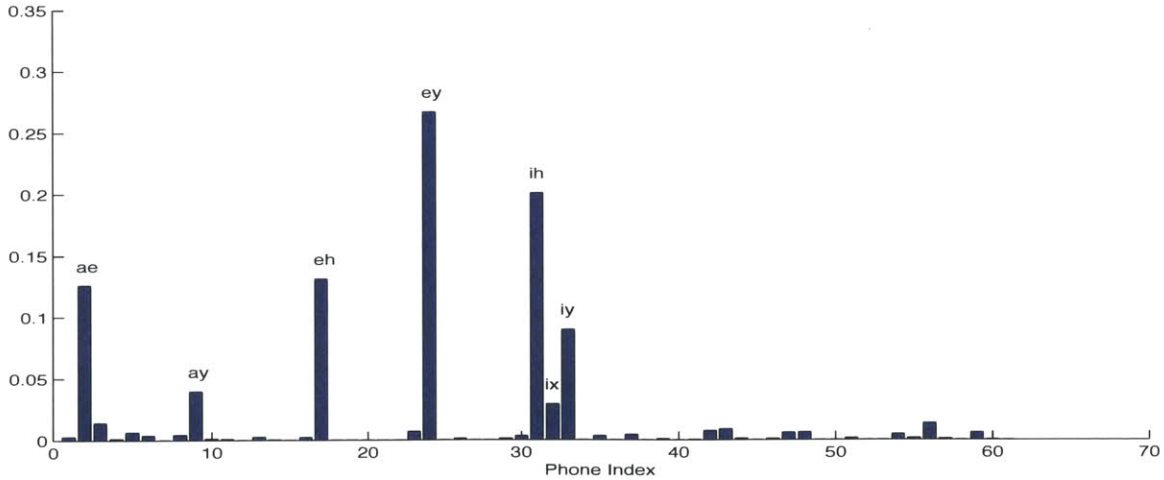
A.1 Vowels



Component 0

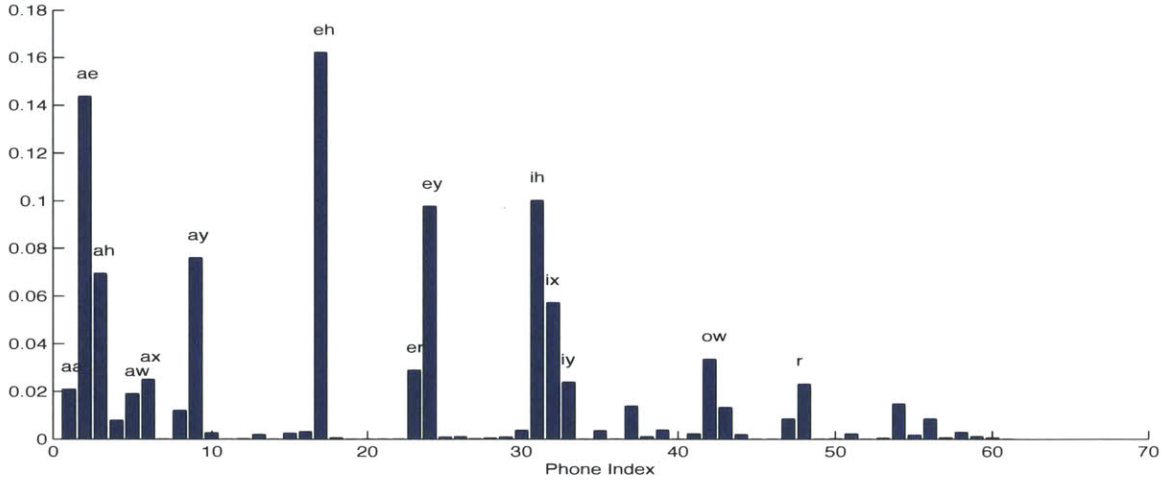


Component 1

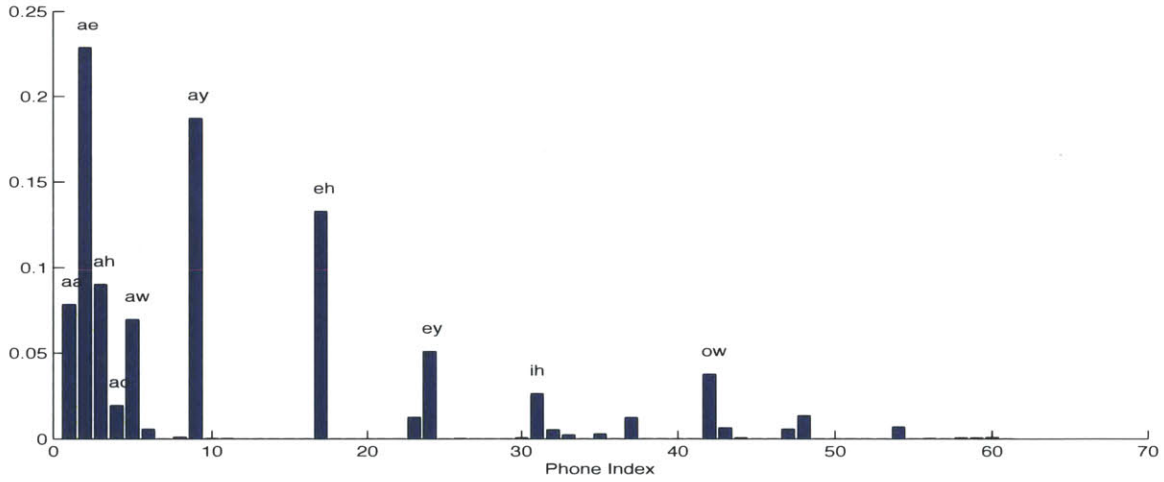


Component 2

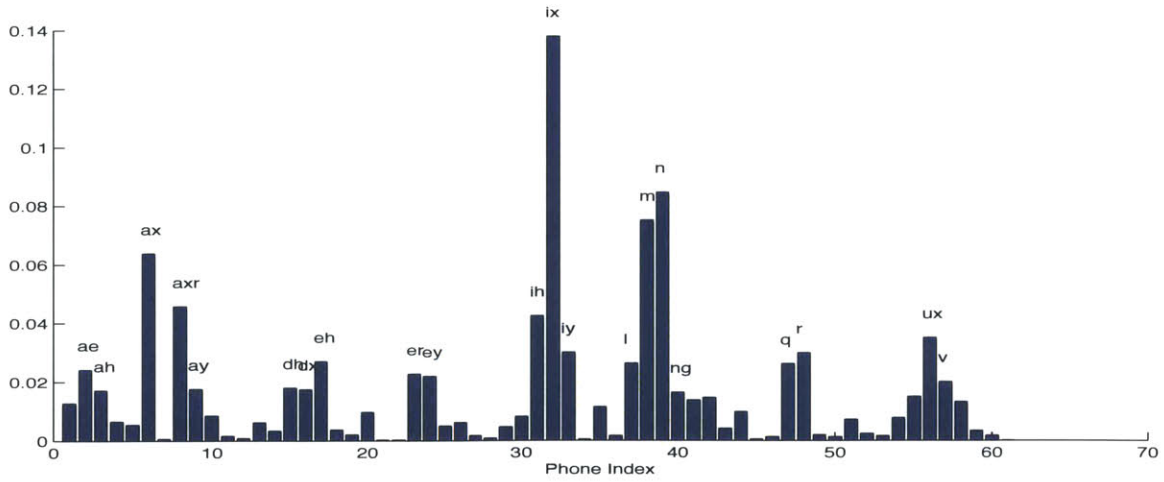
Figure A-1: Vowels 1



Component 5

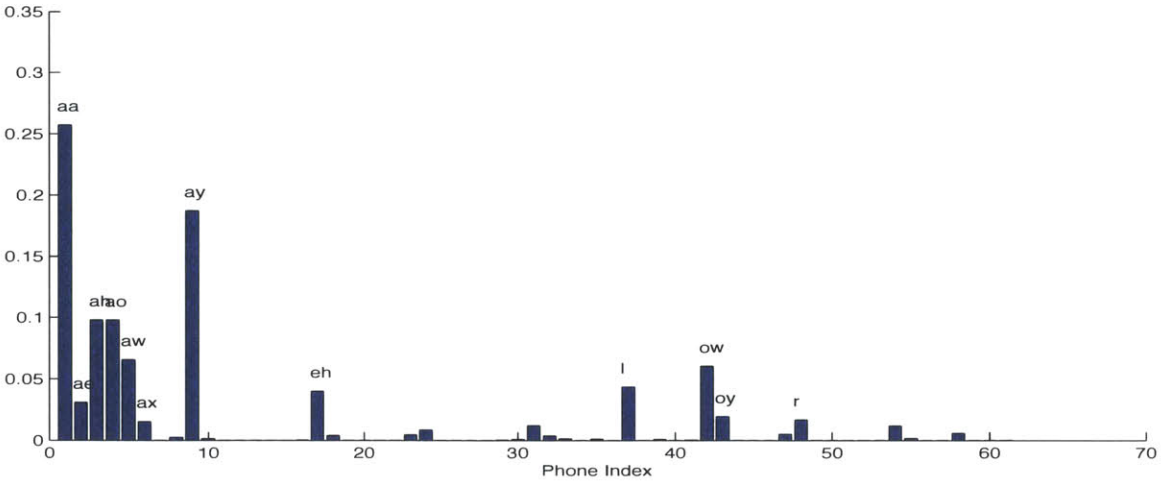


Component 14

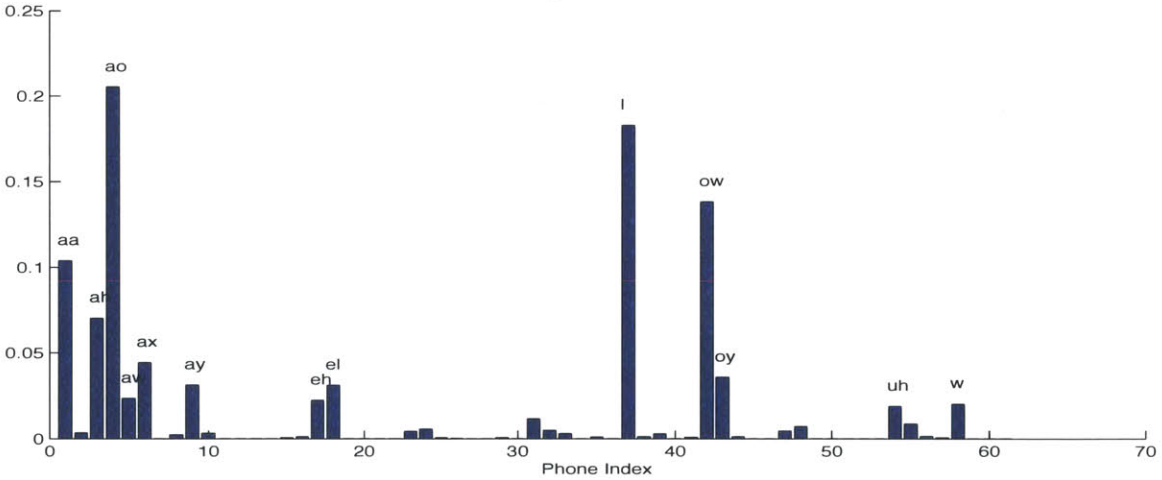


Component 18

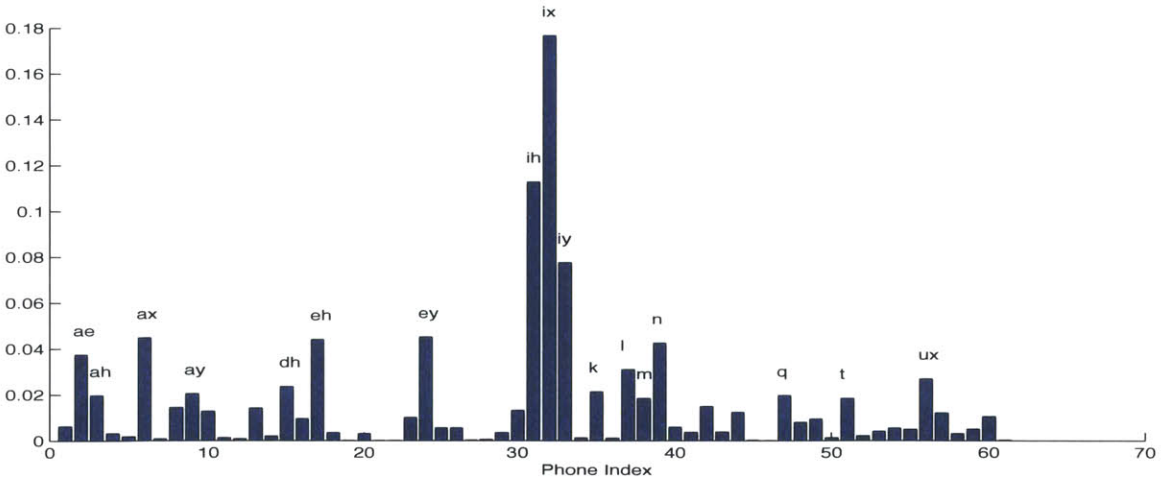
Figure A-2: Vowels 2



Component 25

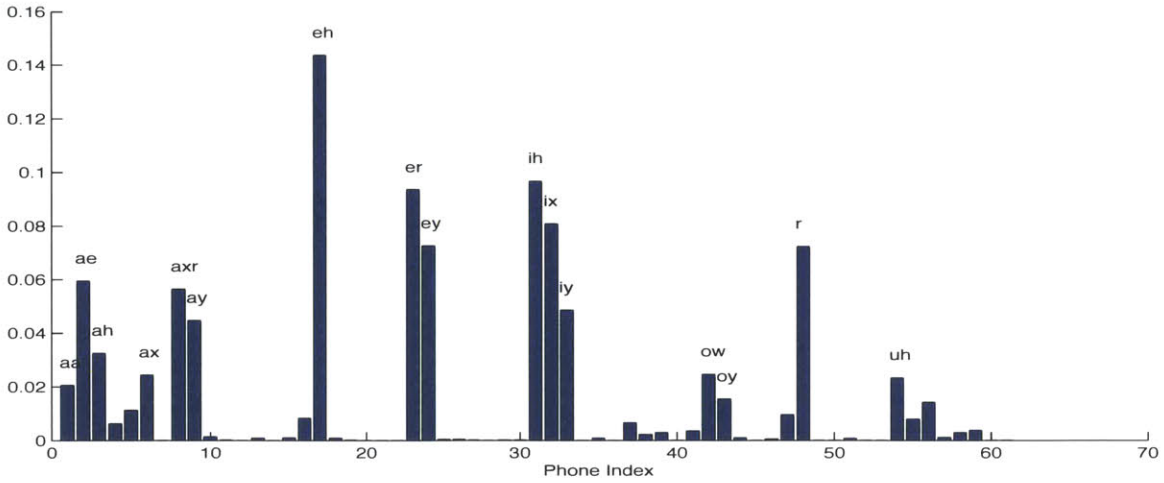


Component 29

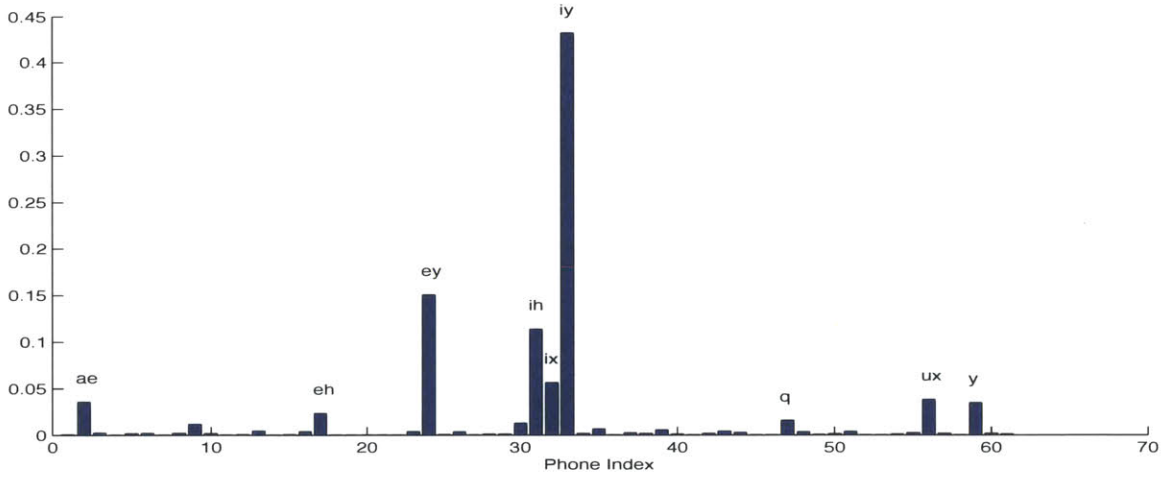


Component 30

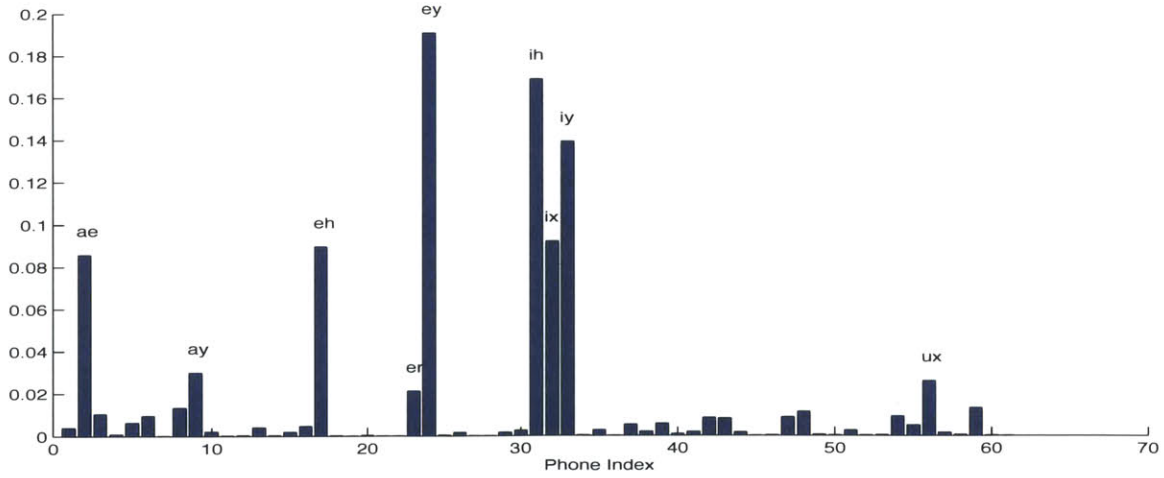
Figure A-3: Vowels 3



Component 31

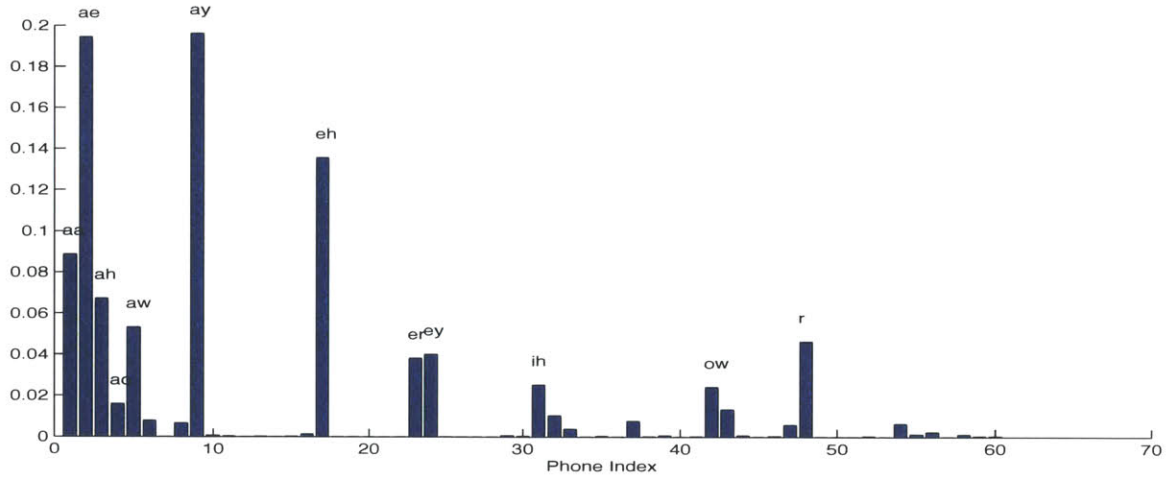


Component 32

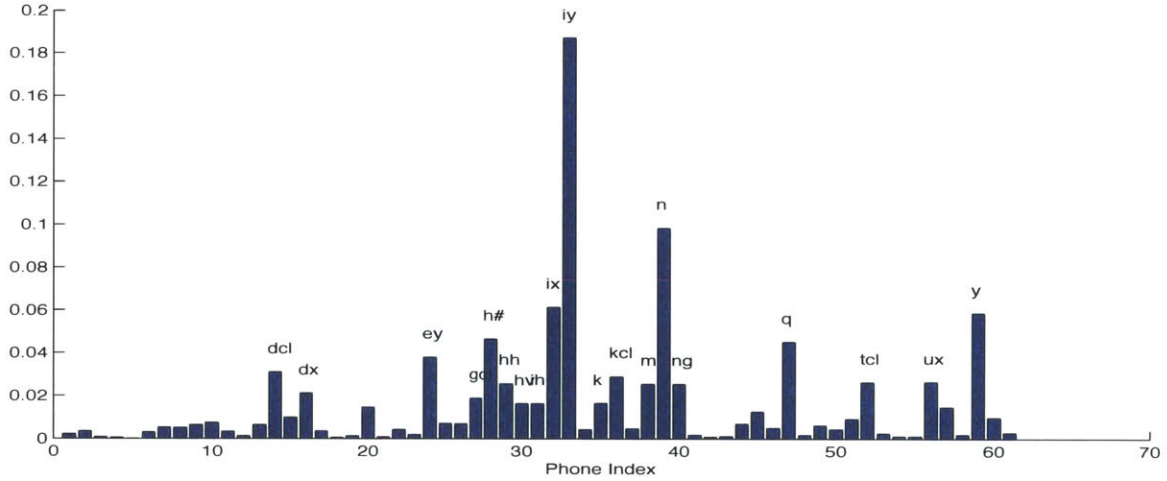


Component 33

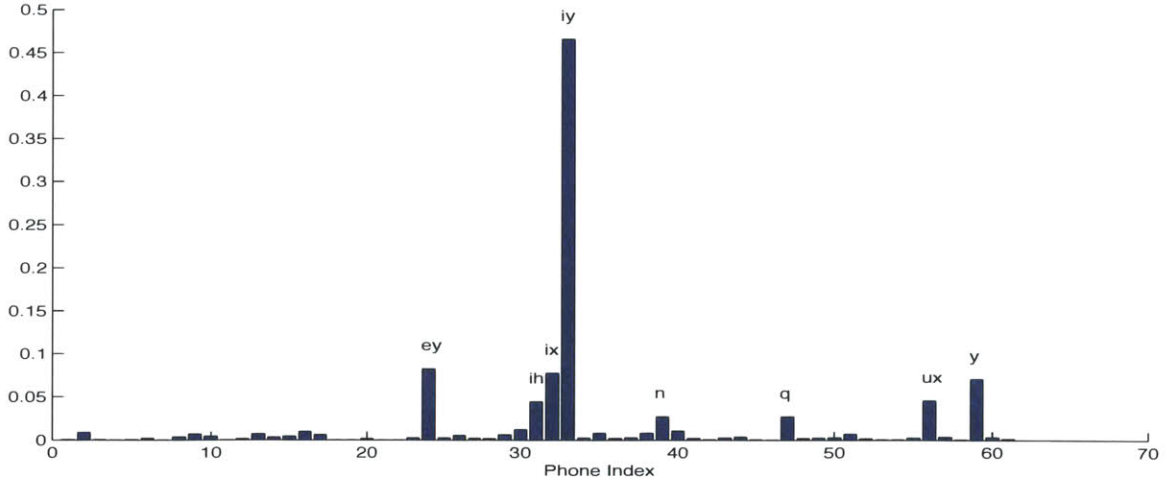
Figure A-4: Vowels 4



Component 34

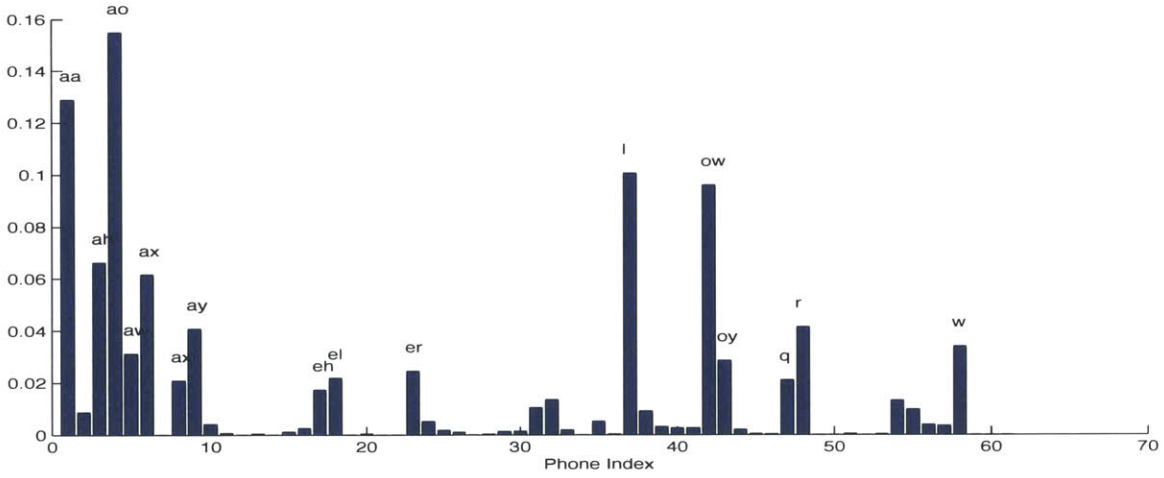


Component 39

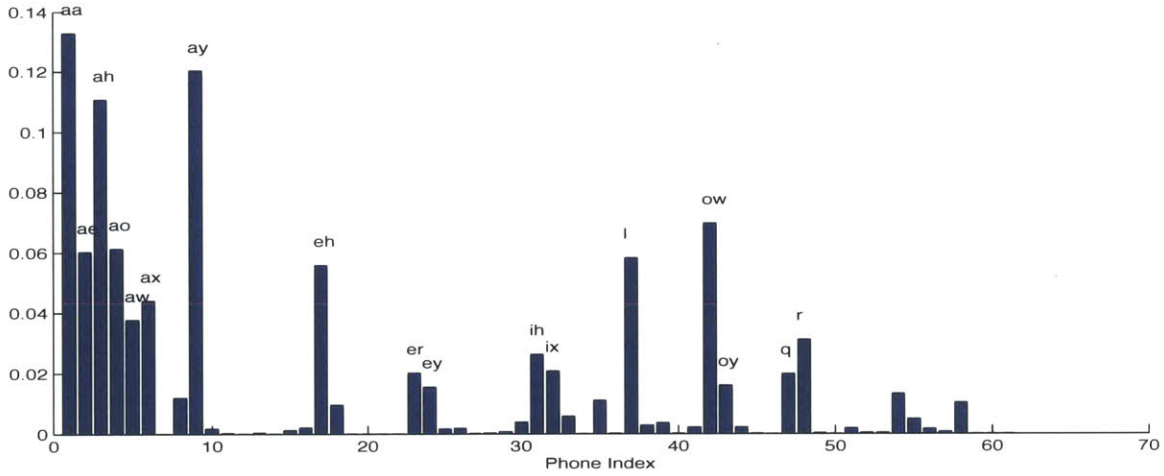


Component 45

Figure A-5: Vowels 5



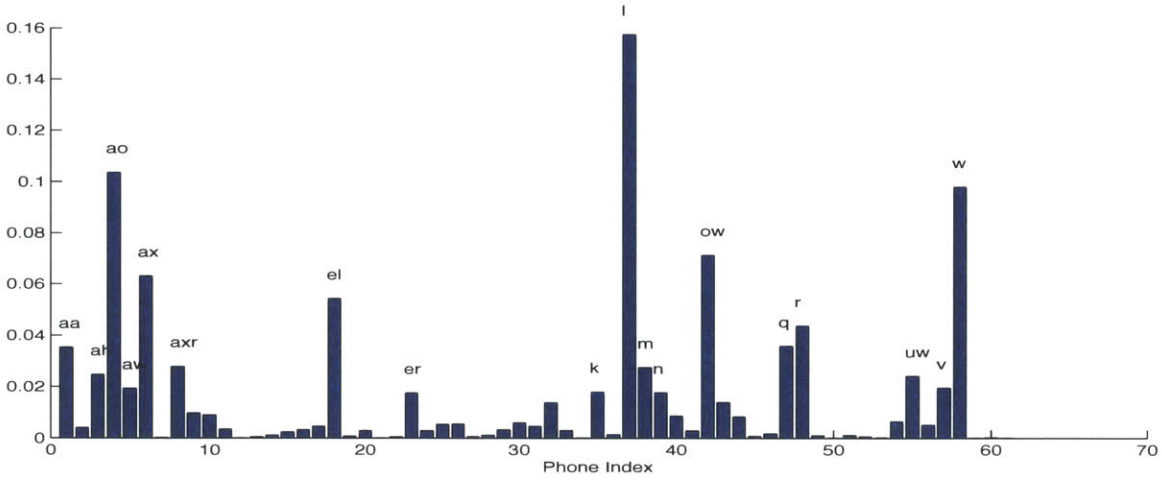
Component 48



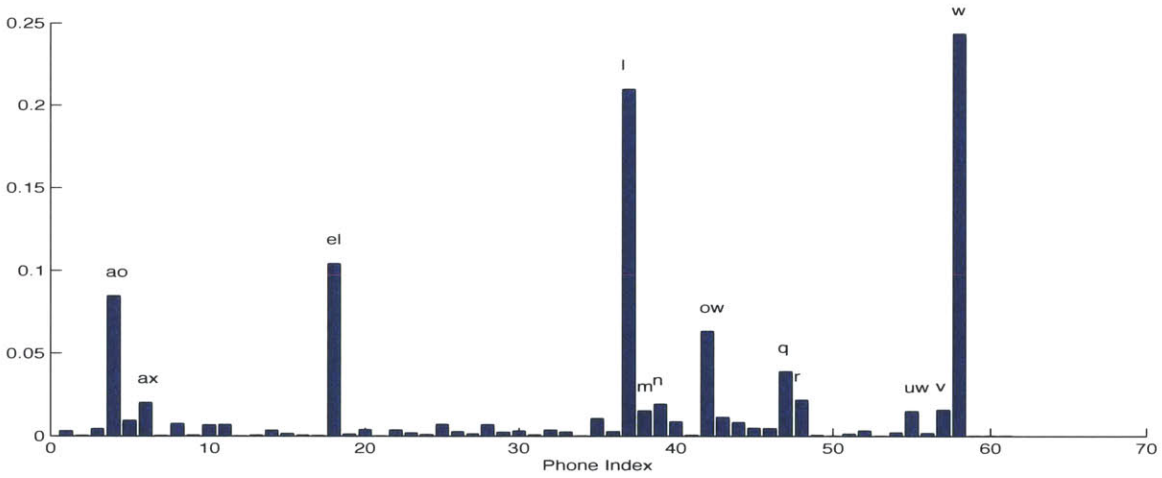
Component 49

Figure A-6: Vowels 6

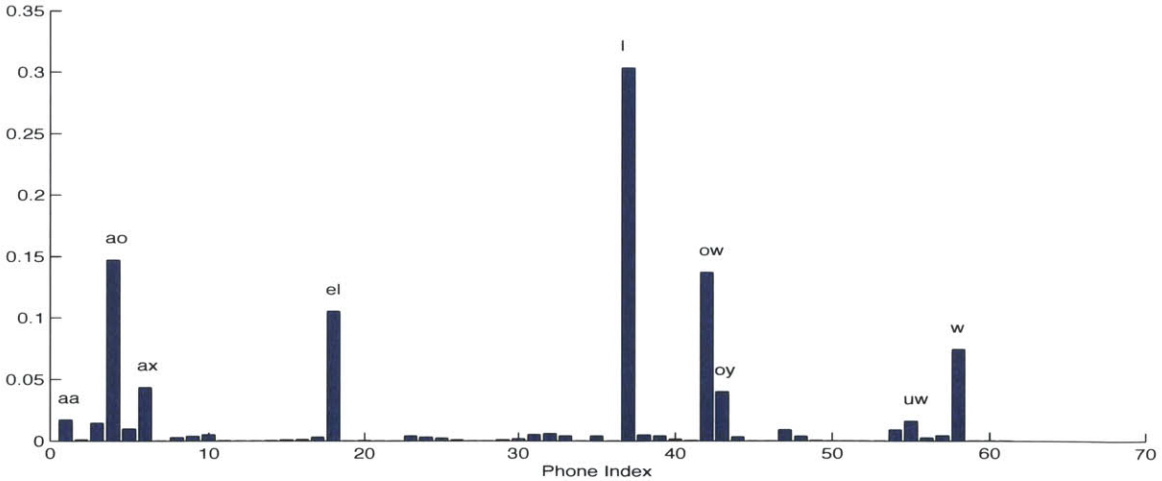
A.2 Semi-vowels and Retroflex



Component 12

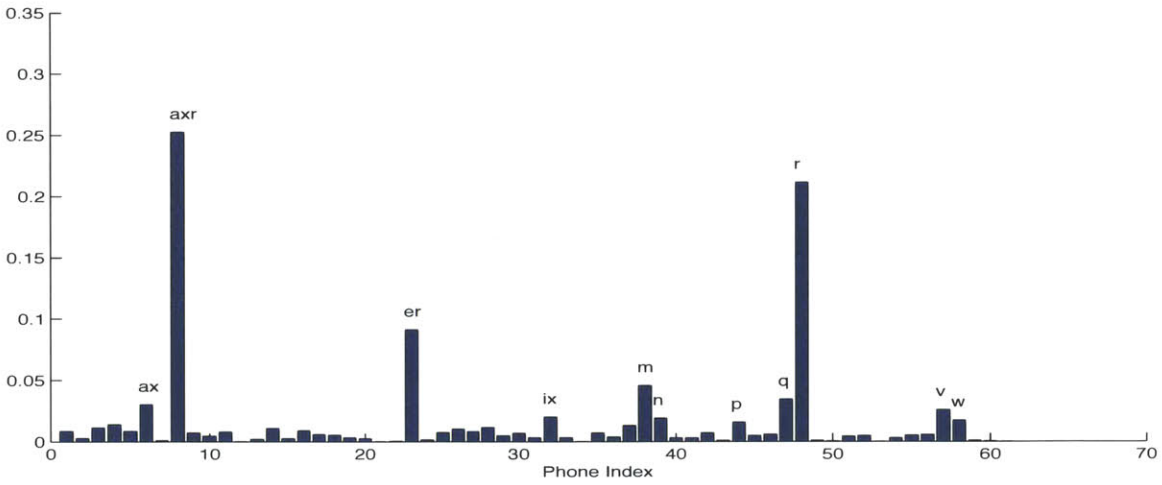


Component 15

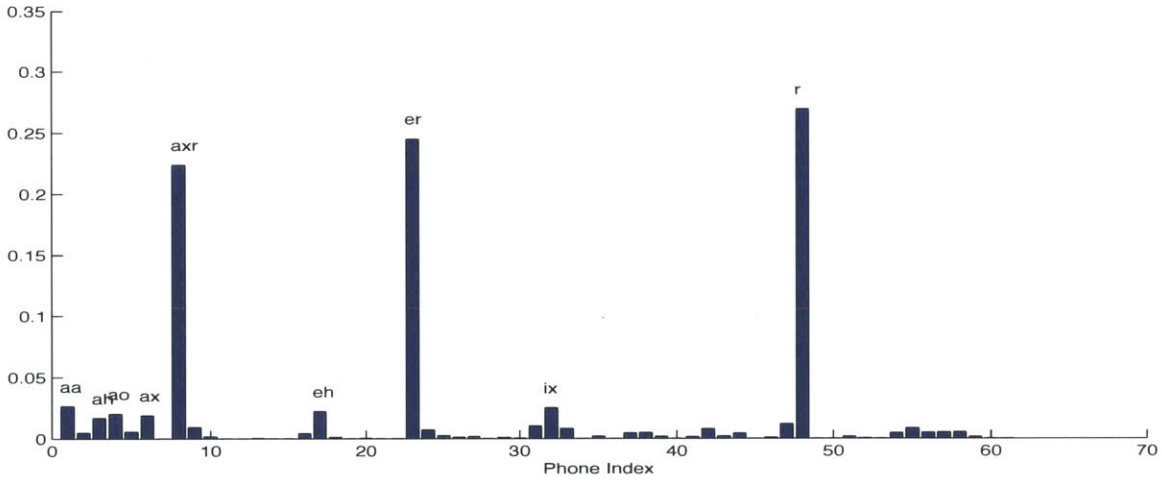


Component 23

Figure A-7: Semi-vowels



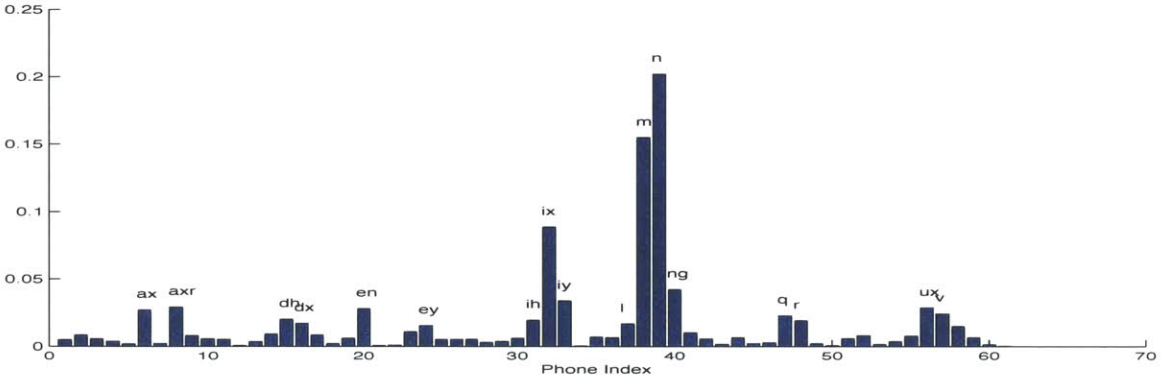
Component 3



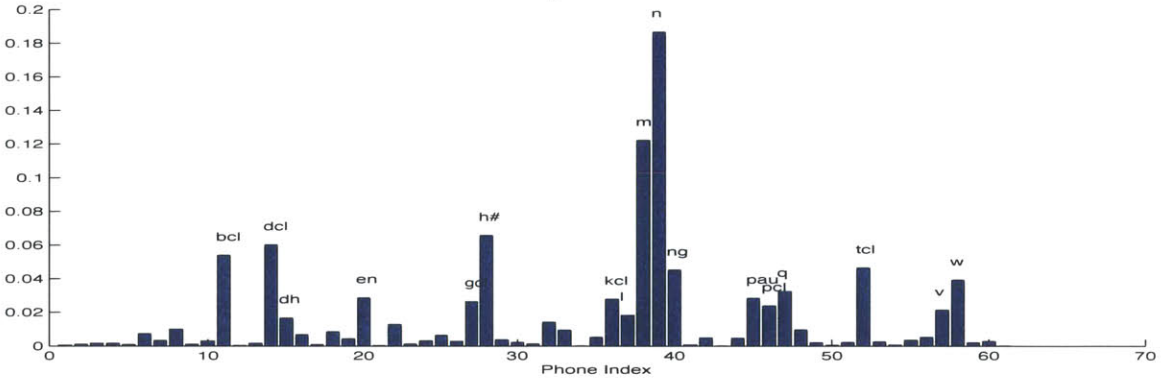
Component 7

Figure A-8: Retroflex

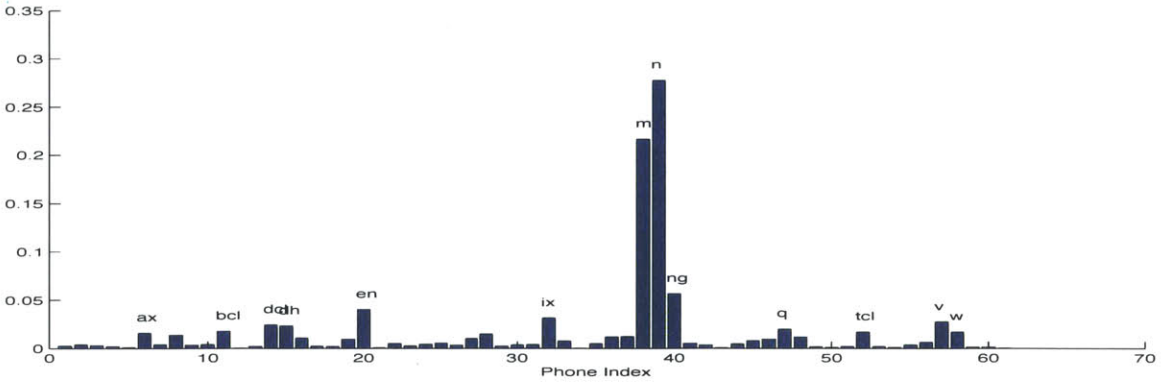
A.3 Nasals



Component 24



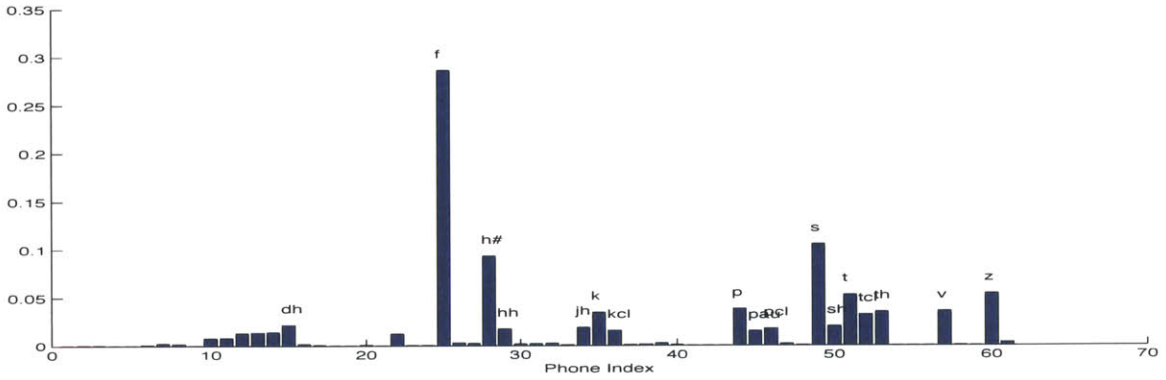
Component 26



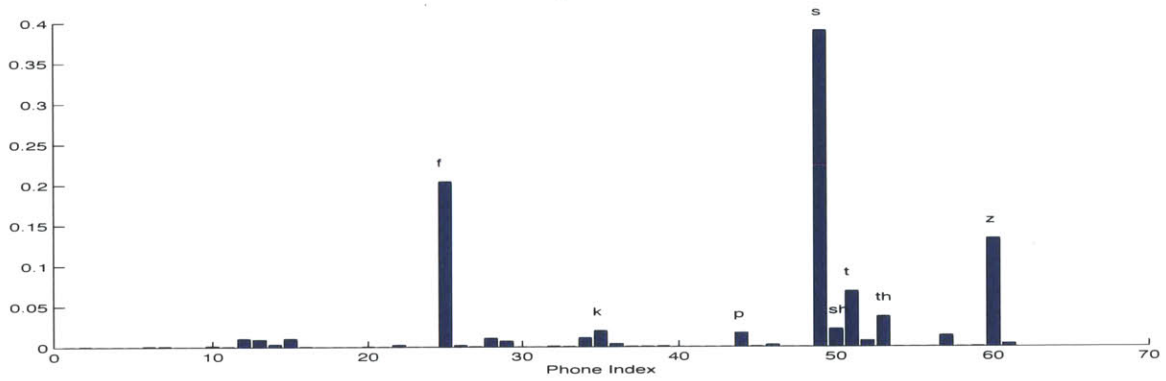
Component 28

Figure A-9: Nasals

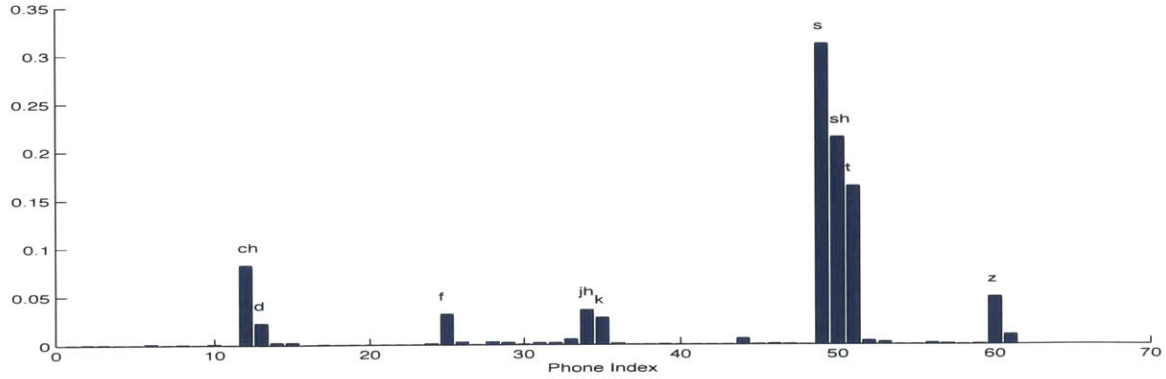
A.4 Fricatives



Component 4

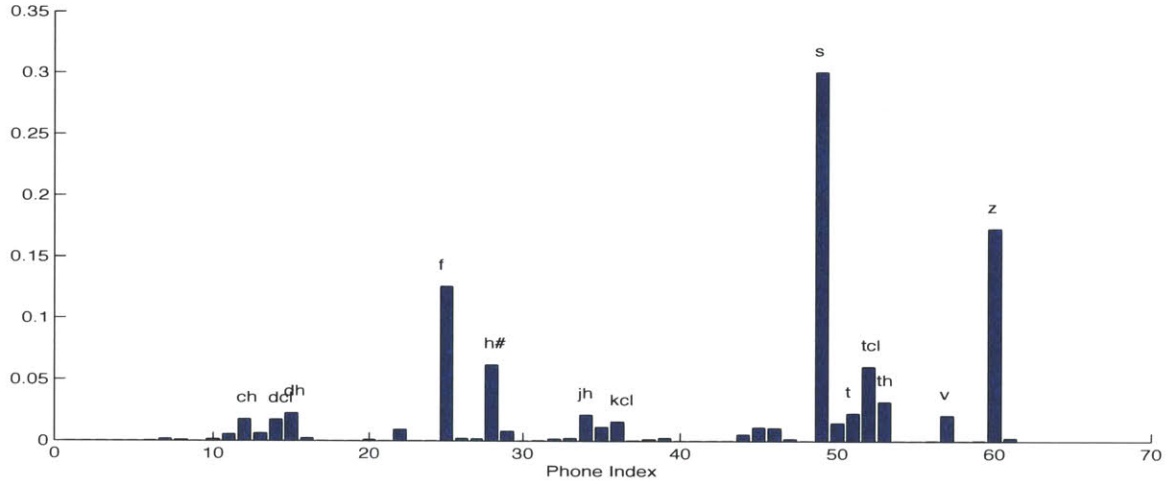


Component 8

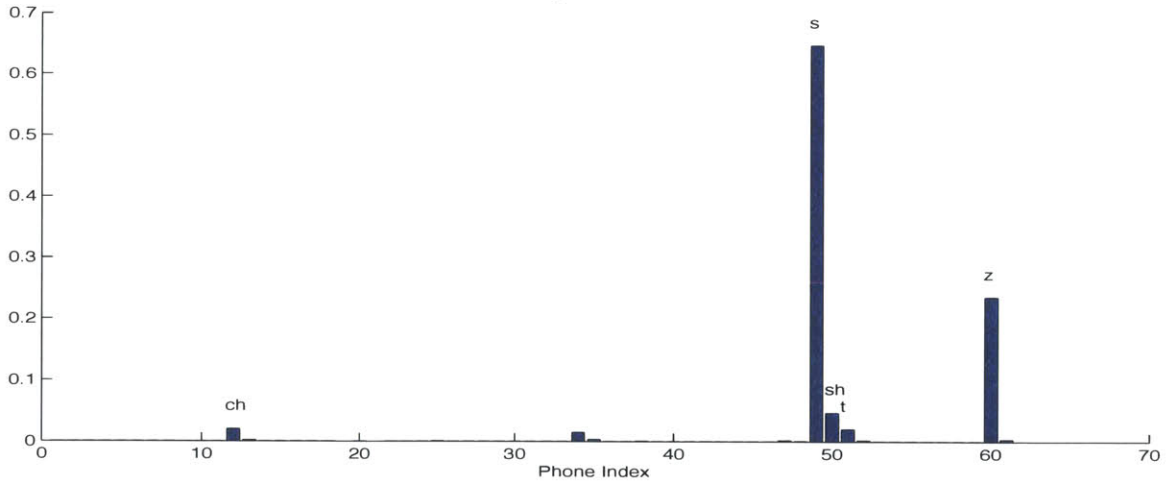


Component 9

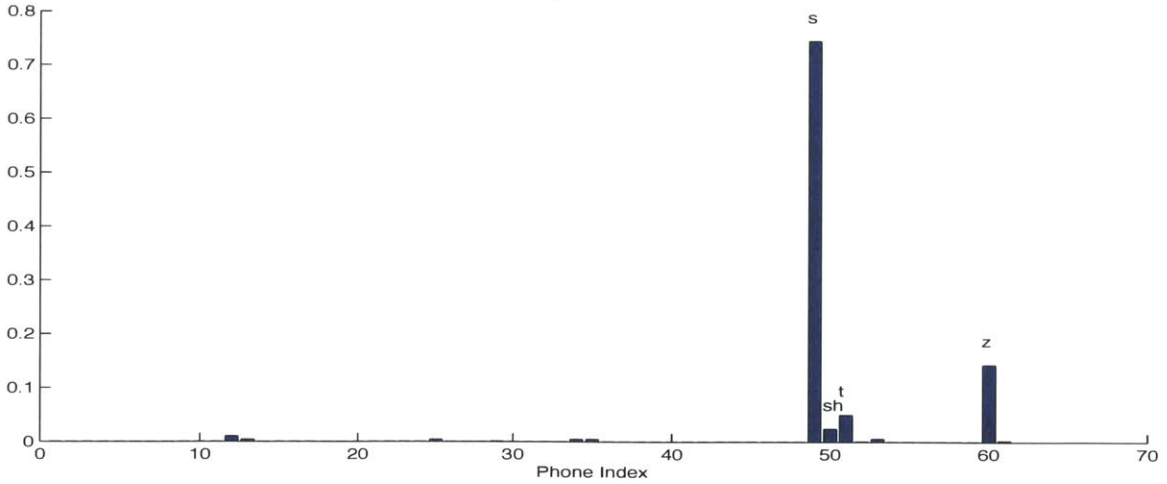
Figure A-10: Fricatives 1



Component 10

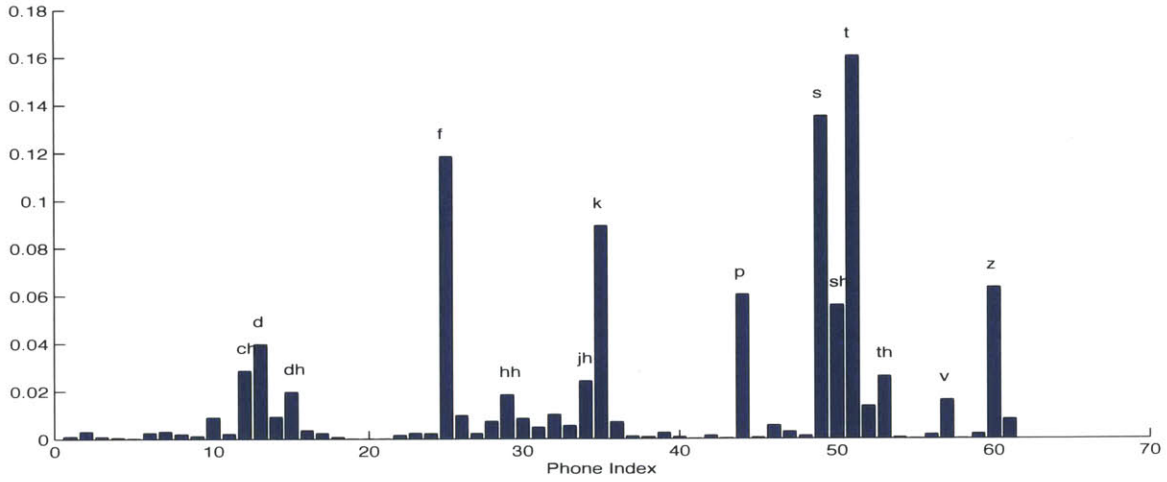


Component 17

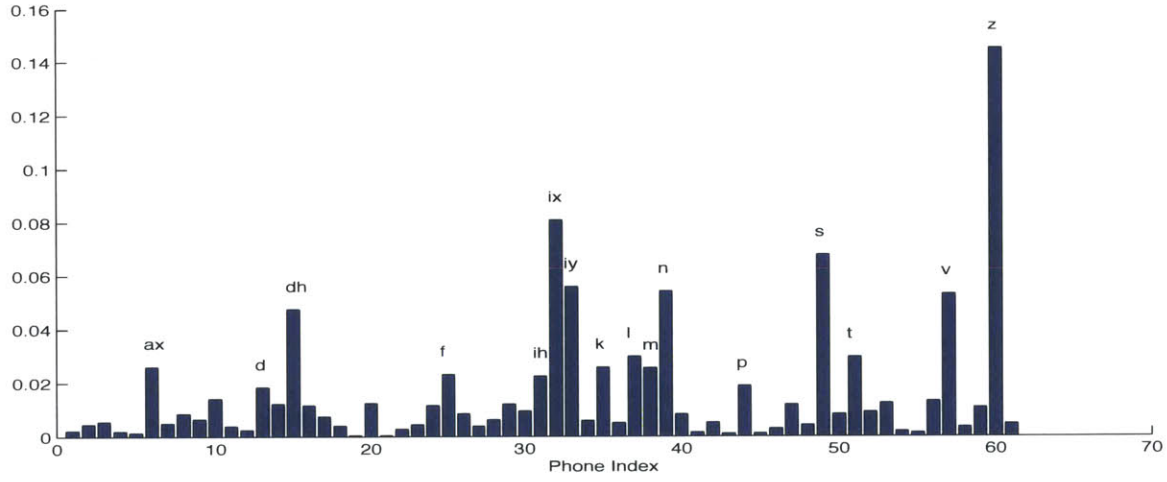


Component 19

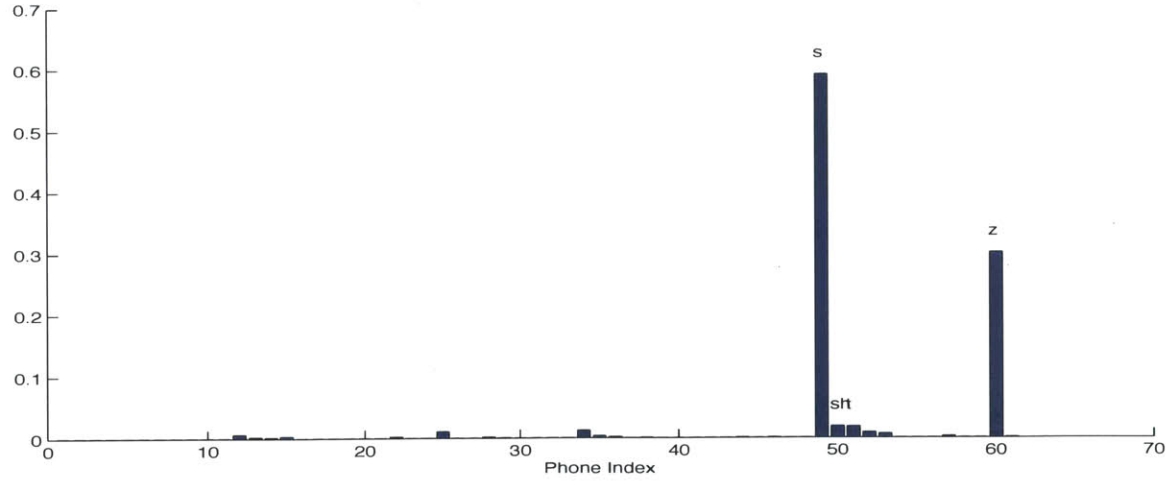
Figure A-11: Fricatives 2



Component 27



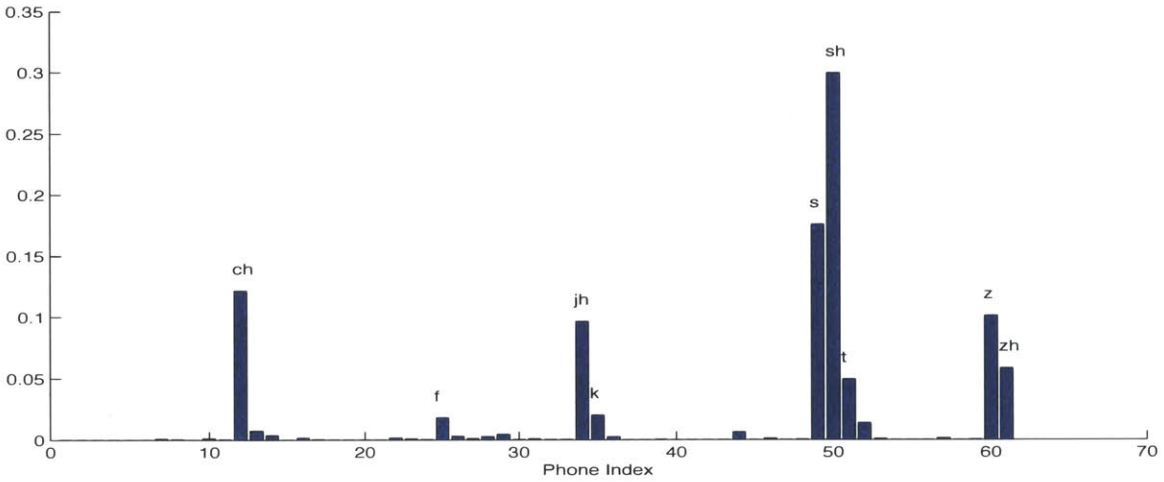
Component 37



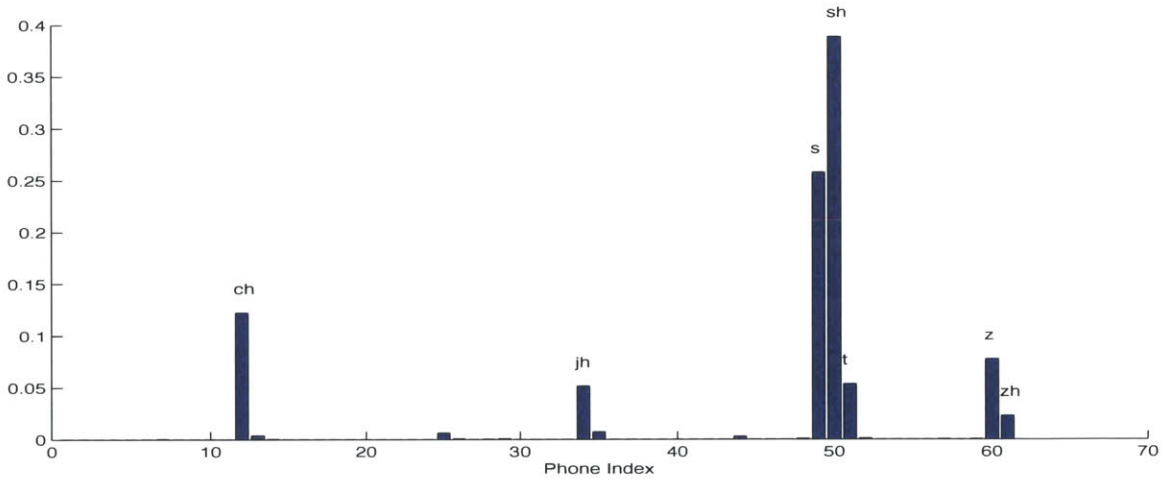
Component 47

Figure A-12: Fricatives 3

A.5 Affricates



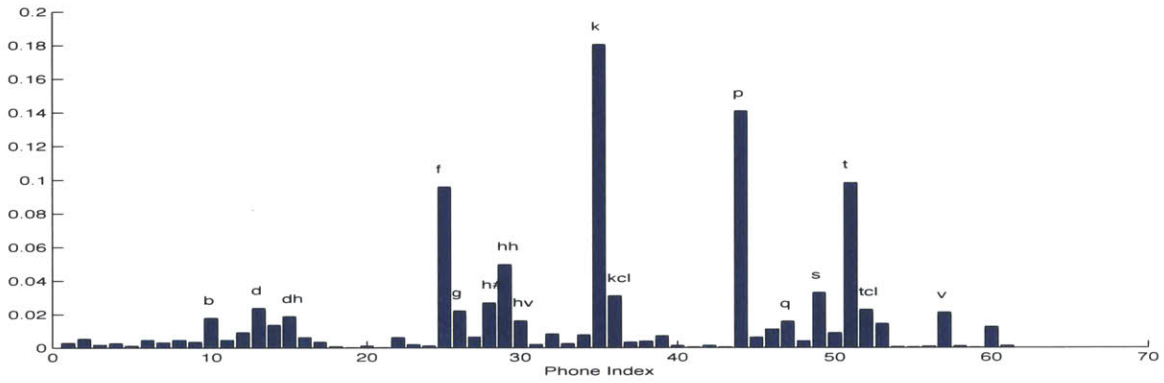
Component 20



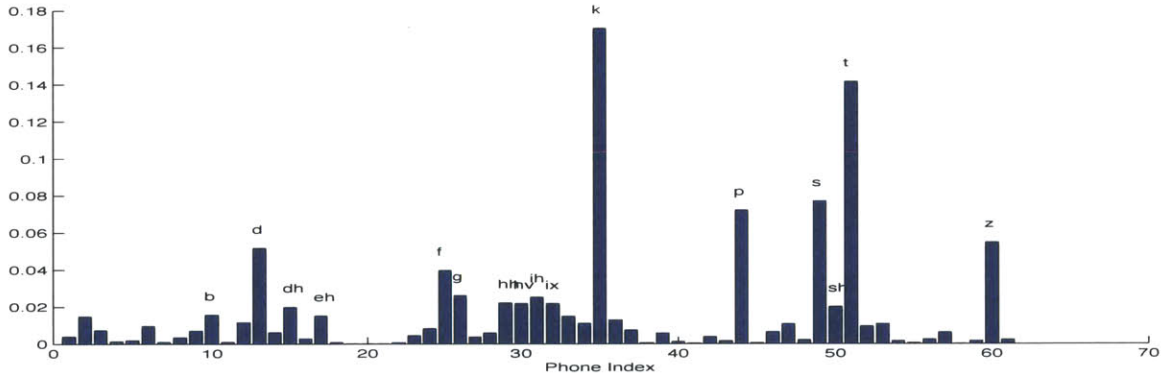
Component 43

Figure A-13: Affricates

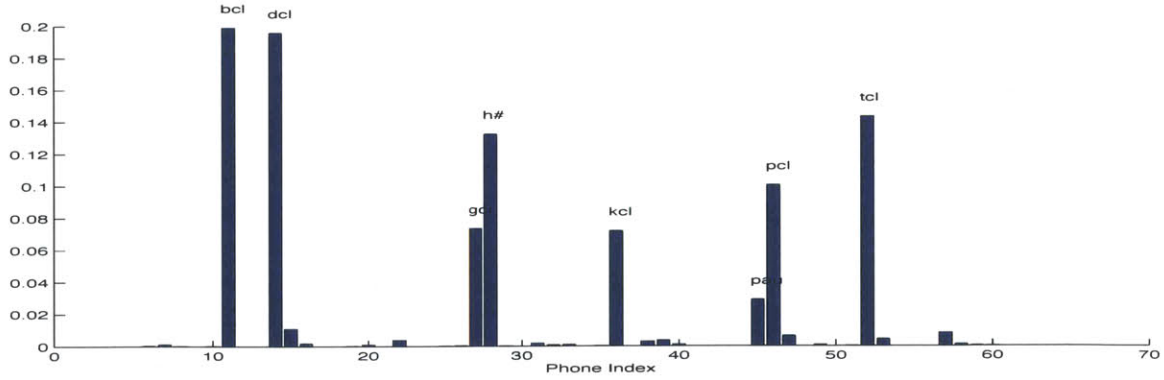
A.6 Stops and Stop Closures



Component 6



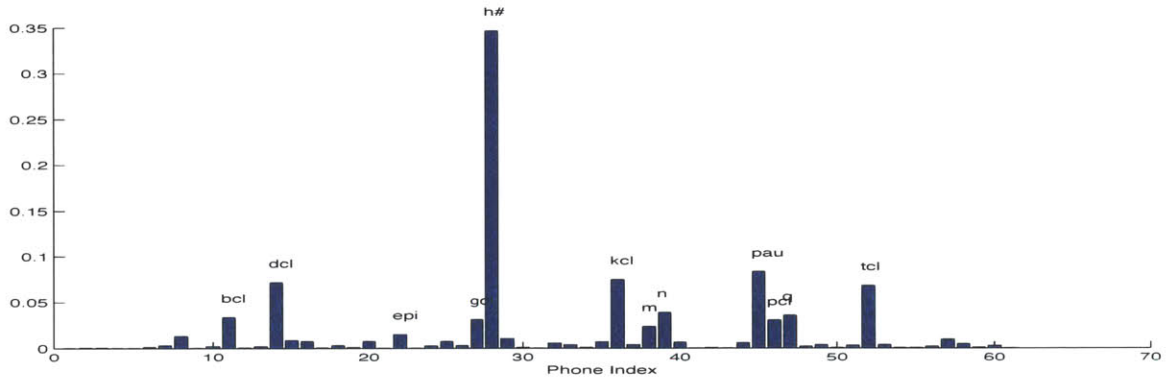
Component 11



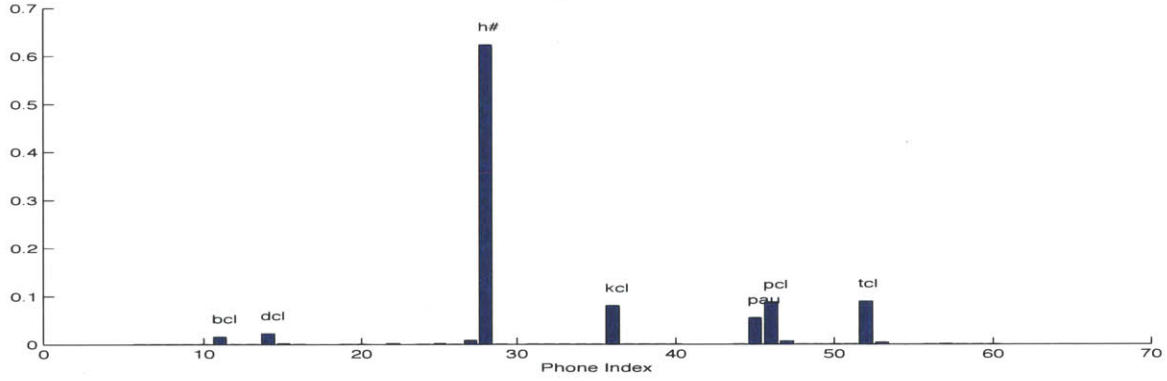
Component 35

Figure A-14: Stops and Stop Closures

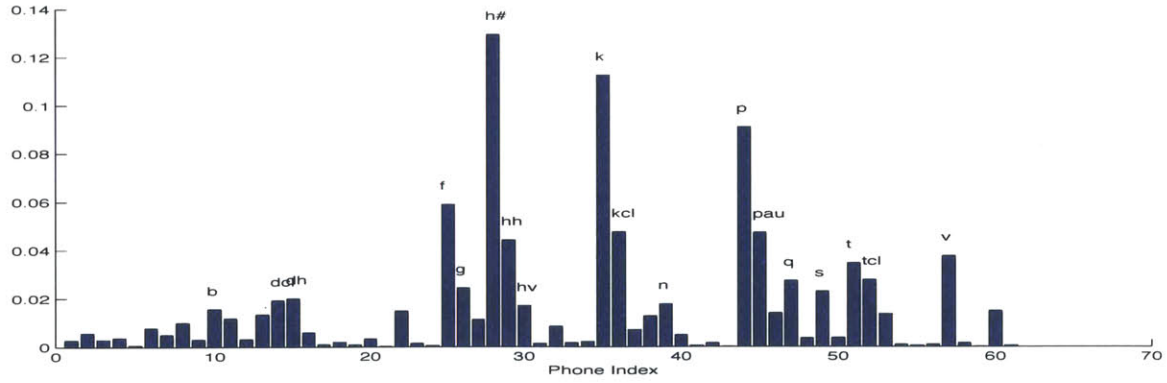
A.7 Silence



Component 13

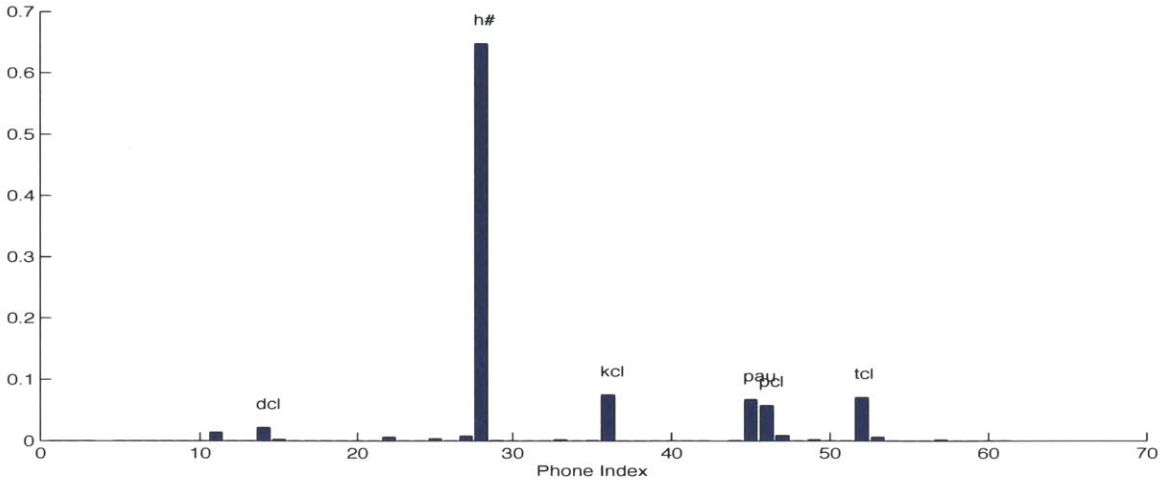


Component 16

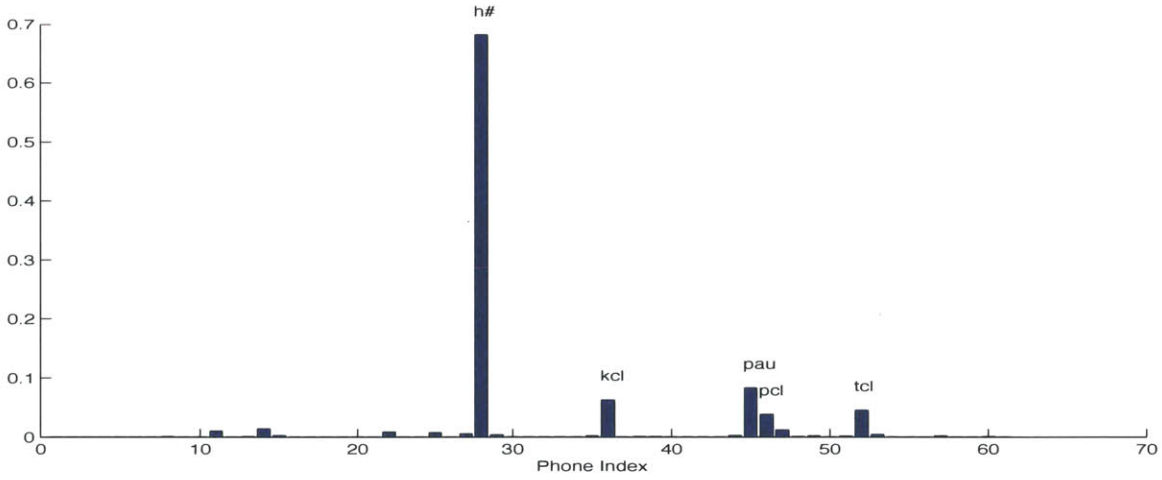


Component 21

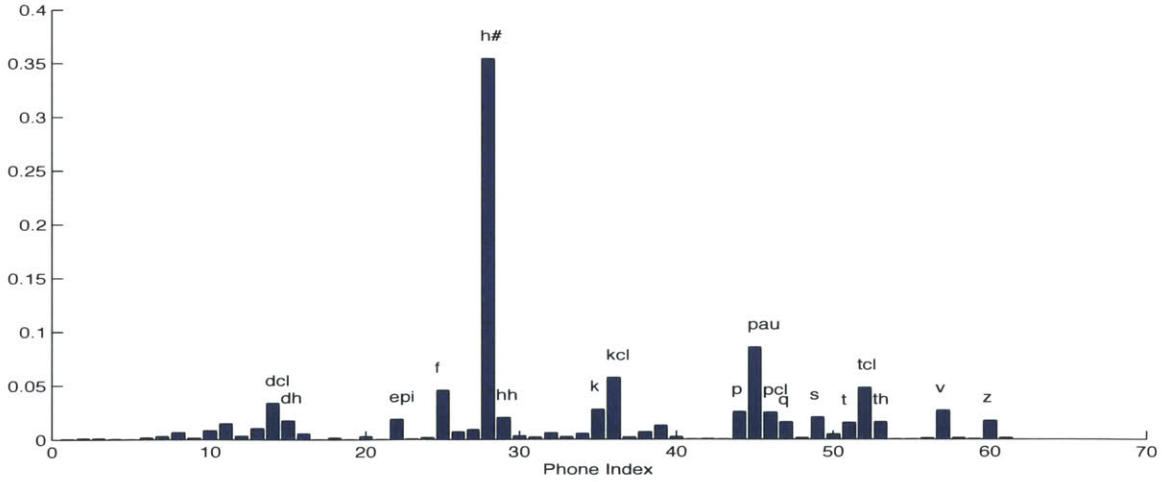
Figure A-15: Silence 1



Component 22

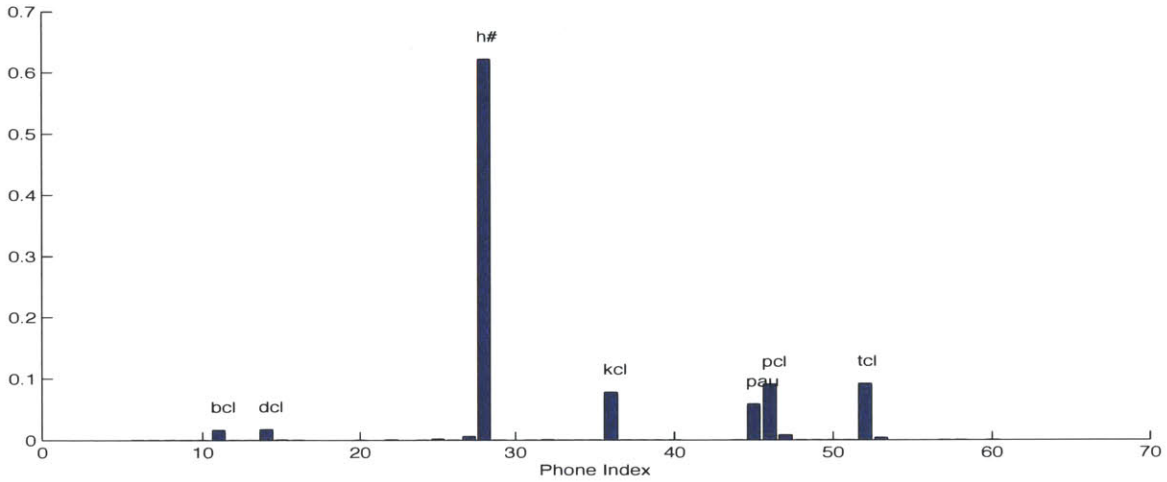


Component 36

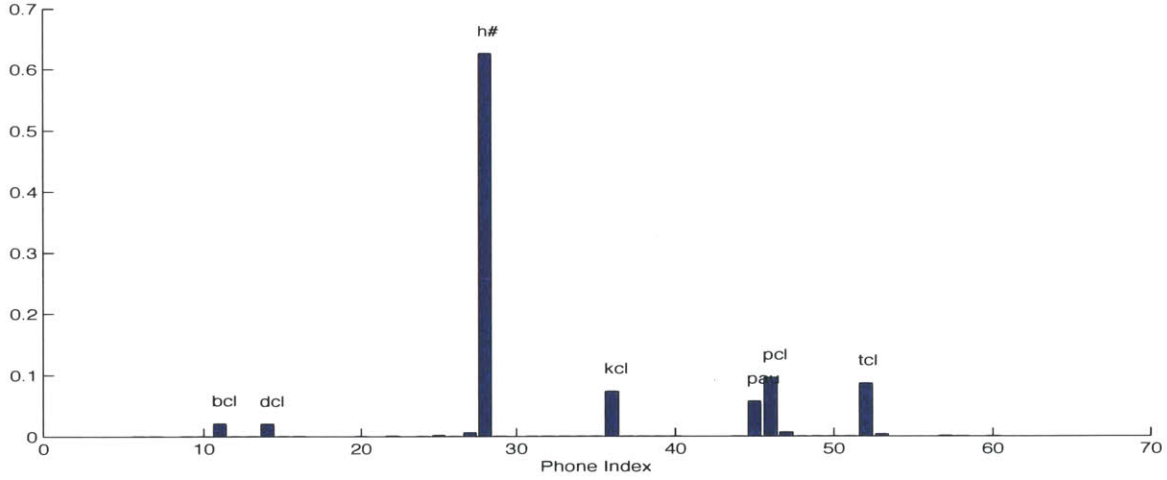


Component 38

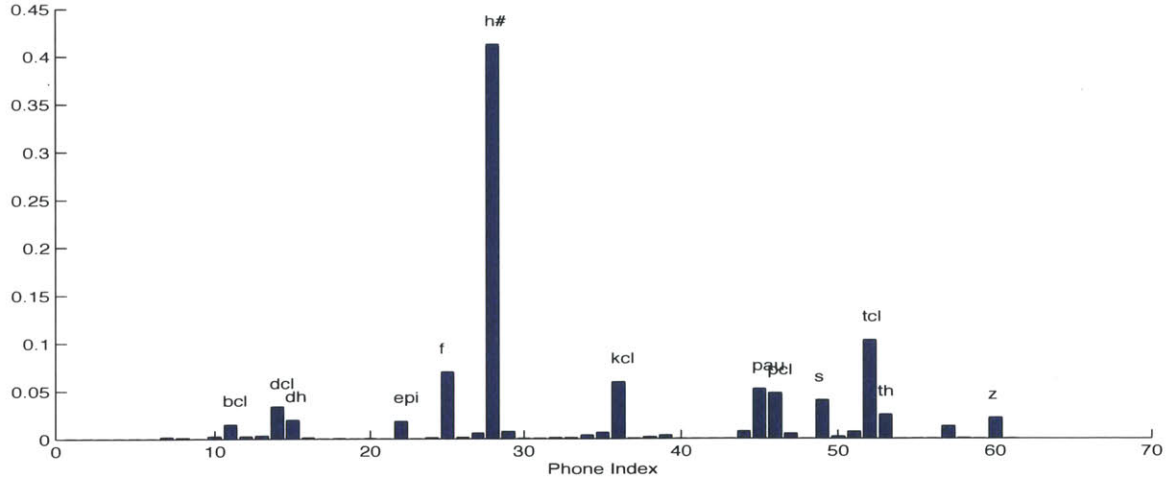
Figure A-16: Silence 2



Component 41



Component 42



Component 44

Figure A-17: Silence 3

Bibliography

[1] D. Aldous. Exchangeability and related topics. In *Ecole d'Ete de Probabilites de Saint-Flour*, pages 1–198, 1985.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *ACM Commununications*, 51:117–122, 2008.

[3] G. Aradilla, H. Bourlard, and M. Magimai-Doss. Posterior features applied to speech recognition tasks with user-defined vocabulary. In *Proc. ICASSP*, 2009.

[4] A. Asaei, B. Picart, and H. Bourlard. Analysis of phone posterior feature space exploiting class specific sparsity and MLP-based similarity measure. In *Proc. ICASSP*, 2010.

[5] I. Badr, I. McGraw, and J. Glass. Pronunciation learning from continuous speech. In *Proc. Interspeech*, pages 549–552, 2011.

[6] J. Baker, Li Deng, J. Glass, S. Khudanpur, C.-H. Lee, N. Morgan, and D. O'Shaughnessy. Developments and directions in speech recognition and understanding. *IEEE Signal Processing Magazine*, 26(3), 2009.

[7] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[8] Y. Bengio and X. Glorot. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, pages 249–256, 2010.

[9] NVidia CUDA C Best Practices Guide. `http://developer.nvidia.com/cuda-downloads/`.

[10] D. Blackwell and J. B. Macqueen. Ferguson distributions via pólya urn schemes. *The Annals of Statistics*, 1:353–355, 1973.

[11] D. M. Blei and M. I. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1:121–144, 2005.

[12] H. Bourlard and N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[13] H. Bourlard and N. Morgan. Continuous speech recognition by connectionist statistical methods. *IEEE Trans. on Neural Networks*, 4:893–909, 1993.

[14] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau. GPU accelerated acoustic likelihood computations. In *Proc. Interspeech*, pages 964–967, 2008.

[15] M. A. Carlin, S. Thomas, A. Jansen, and H. Hermansky. Rapid evaluation of speech representations for spoken term discovery. In *Proc. Interspeech*, 2011.

[16] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. on Database Systems*, 27(2):188–228, 2002.

[17] K. Chan, W. Fu,. and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE Trans. on Knowledge and Data Engineering*, 15(3):686–705, 2003.

[18] H.-A Chang and J. Glass. Hierarchical large-margin gaussian mixture models for phonetic classification. In *Proc. ASRU*, pages 272–277, 2007.

[19] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide. Pipelined back-propagation for context-dependent deep neural networks. In *Proc. Interspeech*, 2012.

[20] J. Chong, E. Gonina, Y. Yi, and K. Keutzer. A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit. In *Proc. Interspeech*, pages 1183–1186, 2009.

[21] W. Chung-Hsien and C. Yeou-Jiunn. Multi-keyword spotting of telephone speech using a fuzzy search algorithm and keyword-driven two-level CBSM. *Speech Communication*, 33:197–212, 2001.

[22] M. Cohen, N. Morgan, D. Rumelhart, V. Abrash, and H. Franco. Context-dependent connectionist probability estimation in a hybrid HMM-neural net speech recognition system. *Computer Speech and Language*, 8:211–222, 1994.

[23] NVidia CUDA CUBLAS. `https://developer.nvidia.com/cublas`.

[24] NVidia CUFFT. `https://developer.nvidia.com/cufft/`.

[25] G. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. In *IEEE Trans. on Audio, Speech and Language Processing*, 2012.

[26] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. In *Proc. ICASSP*, pages 4688–4691, 2011.

[27] L. Deng and H. Strik. Structure-based and template-based automatic speech recognition: Comparing parametric and non-parametric approaches. In *Proc. Interspeech*, 2007.

[28] P. R. Dixon, T. Oonishi, and S. Furui. Fast acoustic computations using graphics processors. In *Proc. ICASSP*, pages 4321–4324, 2009.

[29] R. E. Donovan and P. C. Woodland. Automatic speech synthesiser parameter estimation using hmms. In *Proc. ICASSP*, pages 640–643, 1995.

[30] M. Doss, T. Stephenson, H. Bourlard, and S. Bengio. Phoneme-grapheme based speech recognition system. In *Proc. ASRU*, pages 94–98, 2003.

[31] M. Dredze, A. Jansen, G. Coppersmith, and K. Church. NLP on spoken documents without ASR. In *Proc. EMNLP*, 2010.

[32] D. P. W. Ellis, R. Singh, and S. Sivadas. Tandem acoustic modeling in large-vocabulary recognition. In *Proc. ICASSP*, pages 517–520, 2001.

[33] U. Erra. Toward real time fractal image compression using graphics hardware. In *Proc. Advances in Visual Computing*, 2005.

[34] NIST Spoken Term Detection 2006 Evaluation. `http://www.nist.gov/speech/tset/std/`.

[35] T. S. Ferguson. A bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230, 1973.

[36] W. N. Francis and H. Kucera. *Frequency analysis of English usage: lexicon and grammar*. Houghton-Mifflin, 1982.

[37] A. Garcia and H. Gish. Keyword spotting of arbitrary words using minimal speech resources. In *Proc. ICASSP*, 2006.

[38] H. Gish, M. Siu, A. Chan, and W. Belfield. Unsupervised training of an hmm-based speech recognizer for topic classification. In *Proc. Interspeech*, pages 1935–1938, 2009.

[39] J. Glass. Towards unsupervised speech processing. In *Proc. ISSPA*, 2012.

[40] J. Glass, T. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay. Recent progress in the mit spoken lecture processing project. In *Proc. Interspeech*, pages 2553–2556, 2007.

[41] J. Glass, T. Hazen, L. Hetherington, and C. Wang. Analysis and processing of lecture audio data: preliminary investigations. In *Proc. of HLT-NAACL*, pages 9–12, 2004.

[42] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood component analysis. *Advances in Neural Information Processing Systems*, 17:513–520, 2005.

[43] Google. `http://www.google.com/mobile/voice-search/`.

[44] N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High performance discrete fourier transforms on graphics processors. In *Proc. the IEEE conference on Supercomputing*, pages 1–12, 2008.

[45] P. J. Green, S. Richardson, and I. France. Modelling heterogeneity with and without the dirichlet process. *Scandinavian Journal of Statistics*, 28:355–377, 2000.

[46] T. Hazen, W. Shen, and C. White. Query-by-example spoken term detection using phonetic posteriorgram templates. In *Proc. ASRU*, 2009.

[47] J. Hennebert, C. Ris, H. Bourlard, S. Renals, and N. Morgan. Estimation of global posteriors and forward-backward training of hybrid HMM/ANN systems. In *Proc. Eurospeech*, pages 1951–1954, 1997.

[48] H. Hermansky, D. P.W. Ellis, and S. Sharma. Tandem connectionist feature extraction for conventional HMM systems. In *Proc. ICASSP*, 2000.

[49] G. Hinton. A practical guide to training restricted boltzmann machines. Technical report, Department of Computer Science, University of Toronto, 2010.

[50] G. Hinton, L. Deng, D. Yu, G. Dahl, A.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, , and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29, 2012.

[51] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief networks. *Neural Computation*, 18(7):1527–1554, 2006.

[52] J. Hoberock and N. Bell. Thrust: A Parallel Template Library. `http://www.meganewtons.com/`.

[53] D. Horn, J. Sugerman, M. Houston, and P. Hanrahan. Interactive k-d tree GPU raytracing. In *Proc. the symposium on Interactive 3D graphics and games*, pages 167–174, 2007.

[54] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development.* Prentice Hall PTR, 2001.

[55] H. Ishwaran and L. F. James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96:161–173, 2001.

[56] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans. on Speech and Audio Processing*, 23:52–72, 1975.

[57] C. Jankowski, A. Kalyanswamy, S. Basson, and J. Spitz. NTIMIT: a phonetically balanced, continuous speech, telephone bandwidth speech database. In *Proc. ICASSP*, 1990.

[58] A. Jansen and K. Church. Towards unsupervised training of speaker independent acoustic models. In *Proc. Interspeech*, 2011.

[59] A. Jansen, K. Church, and H. Hermansky. Towards spoken term discovery at scale with zero resources. In *Proc. Interspeech*, 2010.

[60] A. Jansen and B. Durme. Efficient spoken term discovery using randomized algorithms. In *Proc. ASRU*, 2011.

[61] A. Jansen and B. Durme. Indexing raw acoustic features for scalable zero resource search. In *Proc. Interspeech*, 2012.

[62] A. Jansen and P. Niyogi. An experimental evaluation of keyword-filler Hidden Markov Models. Technical report, University of Chicago, 2009.

[63] E. Keogh. Exact indexing of dynamic time warping. In *Proc. VLDB*, pages 406–417, 2002.

[64] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. the International Conference on Knowledge Discovery and Data mining*, pages 285–289, 2000.

[65] M. Killer, S. Stker, and T. Schultz. Grapheme based speech recognition. In *Proc. of the Eurospeech*, pages 3141–3144, 2003.

[66] S. Kim, S. Park, and W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. the International Conference Data Engineering*, pages 607–614, 2001.

[67] T. Kinnunen and H. Li. An overview of text-independent speaker recognition: From features to supervectors. *Speech Communications*, 52(1):12–40, 2010.

[68] K. Kintzley, A. Jansen, and H. Hermansky. Event selection from phone posteriorgrams using matched filters. In *Proc. Interspeech*, pages 1905–1908, 2011.

[69] K. Kirchhoff, J. Bilmes, S. Das, N. Duta, M. Egan, G. Ji, F. He, J. Henderson, D. Liu, M. Noamany, P. Schone, R. Schwartz, and D. Vergyri. Novel approaches to arabic speech recognition: Report from the 2002 johns-hopkins summer workshop. In *Proc. ASRU*, 2003.

[70] K. Kurihara. Collapsed variational dirichlet process mixture models. In *Proc. Joint Conference on Artificial Intelligence*, 2007.

[71] LDC. `http://www.ldc.upenn.edu/`.

[72] Y. Lecun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. *Lecture Notes in Computer Science*, 1524:5–50, 1998.

[73] C. Lee and J. Glass. A nonparametric bayesian approach to acoustic model discovery. In *Proc. ACL*, pages 40–49, 2012.

[74] C.-H. Lee, F. K. Soong, and K. K. Paliwal. *Automatic Speech and Speaker Recognition: Advanced Topics.* Springer, 1996.

[75] H. Lee, Y. Largman, P. Pham, and A. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Advances in Neural Information Processing Systems*, pages 1096–1104, 2009.

[76] K. F. Lee. Context dependent phonetic HMMs for continuous speech recognition. *IEEE Trans. Speech and Audio Processing*, 1990.

[77] Y. Lin, T. Jiang, and K. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Com. Syst. Sci.*, 65(3):570–586, 2002.

[78] I. Malioutov and R. Barzilay. Minimum cut model for spoken lecture segmentation. In *Proc. the Annual Meeting of the Association for Computational Linguistics (COLING-ACL)*, pages 25–32, 2006.

[79] I. Malioutov, A. Park, R. Barzilay, and J. Glass. Making sense of sound: Unsupervised topic segmentation over acoustic input. In *Proc. ACL*, pages 504–511, 2007.

[80] M. McCool, K. Wadleigh, B. Henderson, and H. Lin. Performance evaluation of gpus using the rapidmind development platform. In *Proc. the IEEE conference on Supercomputing*, 2006.

[81] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocky. Strategies for training large scale neural network language models. In *Proc. ASRU*, pages 196–201, 2011.

[82] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. Recurrent neural network based language model. In *Proc. Interspeech*, pages 1045–1048, 2010.

[83] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE Trans. on Audio, Speech and Language Processing*, 20(1):14–22, 2012.

[84] A. Mohamed, G. E. Dahl, and G. E. Hinton. Deep belief networks for phone recognition. In *Proc. NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

[85] B. C. J. Moore. *An Introduction to the Psychology of Hearing.* Academic, 1997.

[86] N. Morgan. Deep and wide: Multiple layers in automatic speech recognition. *IEEE Trans. on Audio, Speech and Language Processing*, 20(1):7–13, 2012.

[87] N. Morgan and H. Bourlard. Continuous speech recognition using multilayer perceptrons with hidden Markov models. In *Proc. ICASSP*, pages 413–416, 1990.

[88] A. Muscariello, G. Gravier, and F. Bimbot. Audio keyword extraction by unsupervised word discovery. In *Proc. Interspeech*, pages 656–659, 2009.

[89] R. Neal. Bayesian mixture modeling. In *Proc. the Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, pages 197–211, 1992.

[90] M. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6), 2004.

[91] Nuance. `http://www.nuance.com/recognizer/languages.asp`.

[92] Nuance. `http://www.nuance.com/for-individuals/by-industry/education-solutions/transcribing-interview/index.htm`.

[93] A. Oppenheim. *Signals and systems.* Prentice Hall, 1997.

[94] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[95] A. Park. *Unsupervised pattern discovery in speech: applications to word acquistion and speaker segmentation.* PhD thesis, MIT Department of Electrical Enginering and Computer Science, 2006.

[96] A. Park and J. Glass. Towards unsupervised pattern discovery in speech. In *Proc. ASRU*, pages 53–58, 2005.

[97] A. Park and J. Glass. Unsupervised word acquisition from speech using pattern discovery. In *Proc. ICASSP*, 2006.

[98] A. Park and J. Glass. Unsupervised pattern discovery in speech. *IEEE Trans. on Audio, Speech and Language Processing*, 16(1), 2008.

[99] S. Phillips and A. Rogers. Parallel speech recognition. *International Journal of Parallel Programming*, 27(4):257–288, 1999.

[100] NIST Spoken Term Detection Evaluation Plan. `http://www.nist.gov/speech/tests/std/docs/std06-evalplan-v10.pdf/`.

[101] L. Rabiner and B.-H. Juang. *Fundamentals of speech recognition.* Prentice-Hall, Inc., 1993.

[102] J. Ramos. Using TF-IDF to determine word relevance in document queries. In *Proc. of ICML*, 2003.

[103] C. E. Rasmussen. The infinite gaussian mixture model. In *Neural Information Processing Systems*, pages 554–560. MIT Press, 2000.

[104] T. Rath and R. Manmatha. Lower-bounding of dynamic time wapring distances for multivariate time series. Technical Report MM-40, University of Massachusetts Amherst, 2002.

[105] S. Renals and N. Morgan. Connectionist probability estimation in HMM speech recognition. *IEEE Trans. on Speech and Audio Processing*, 2:161–174, 1992.

[106] D. A. Reynolds. An overview of automatic speaker recognition technology. In *Proc. ICASSP*, pages 4072–4075, 2002.

[107] D. A. Reynolds and R. C. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE Trans. on Speech and Audio Processing*, 3(1):72–83, 1995.

[108] A. Robinson, G. Cook, D. Ellis, E. Fosler-Lussier, S. Renals, and D. Williams. Connectionist speech recognition of broadcast news. *Speech Communications*, pages 27–45, 2002.

[109] R. Rose and D. Paul. A Hidden Markov Model based keyword recognition system. In *Proc. ICASSP*, volume 2, pages 129–132, 1990.

[110] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, pages 696–699, 1988.

[111] T. Sainath, B. Kingsbury, and B. Ramabhadran. Auto-encoder bottleneck features using deep belief networks. In *Proc. ICASSP*, pages 4153–4156, 2012.

[112] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on Speech and Audio Processing*, 26:43–49, 1978.

[113] R. Salakhutdinov. *Learning Deep Generative Models.* PhD thesis, Dept. of Computer Science, University of Toronto, 2009.

[114] R. Salakhutdinov and G. Hinton. Deep Boltzmann Machines. In *Proc. AI and Statistics*, volume 5, pages 448–455, 2009.

[115] R. Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proc. AI and Statistics*, 2010.

[116] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul. Accelerating dynamic time warping subsequence search with gpus and fpgas. In *Proc. IEEE International Conference on Data Mining*, pages 1001–1006, 2010.

[117] S. Scanzio, S. Cumani, R. Roberto, F. Mana, and P. Laface. Parallel implementation of artificial neural network training for speech recognition. *Pattern Recognition Letters*, 31(11):1302–1309, 2010.

[118] J. Sethuraman. A constructive definition of dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.

[119] M. Siu, H. Gish, S. Lowe, and A. Chan. Unsupervised audio pattern discovery using HMM-based self-organized units. In *Proc. Interspeech*, pages 1001–1004, 2011.

[120] M. Siu, H., A. Chan, and W. Belfield. Improved topic classification and keyword discovery using an hmm-based speech recognizer trained without supervision. In *Proc. Interspeech*, pages 2838–2841, 2010.

[121] G. Sivaram and H. Hermansky. Sparse multilayer perceptron for phoneme recognition. *IEEE Trans. on Audio, Speech and Language Processing*, 20(1):23–29, 2012.

[122] K. Stevens. *Acoustic Phonetics*. MIT Press, 1999.

[123] ATI Stream. `http://www.amd.com/stream/`.

[124] Y. Takebayashi, H. Tsuboi, and H. Kanazawa. Keyword-spotting in noisy continuous speech using word pattern vector subabstraction and noise immunity learning. In *Proc. of ICASSP*, volume 2, pages 85–88, 1992.

[125] Intel Hyperthreading Technology. `http://http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`.

[126] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[127] Y. W. Teh, K. Kurihara, and M. Welling. Collapsed variational inference for HDP. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[128] K. Tokuda, T. Kobayashi, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *Proc. ICASSP*, pages 1315–1318, 2000.

[129] NVidia CUDA Toolkit 4.0. `http://developer.nvidia.com/cuda-toolkit-40/`.

[130] E. Trentin and M. Gori. A survey of hybrid ann/hmm models for automatic speech recognition. *Neurocomputing*, 37:91–126, 2001.

[131] F. Valente, M. Doss, C. Plahl, S. Ravuri, and W. Wang. A comparative large scale study of mlp features for mandarin ASR. In *Proc. Interspeech*, pages 2630–2633, 2010.

[132] F. Valente, M. Doss, C. Plahl, S. Ravuri, and W. Wang. Transcribing mandarin broadcast speech using multilayer perceptron acoustic features. *IEEE Trans. Audio, Speech and Language Processing*, 19, 2011.

[133] J. Vanek, J. Trmal, J. V. Psutka, and J. Psutka. Optimization of the gaussian mixture model evaluation on GPU. In *Proc. Interspeech*, pages 1737–1740, 2011.

[134] B. Varadarajan, S. Khudanpur, and E. Dupoux. Unsupervised learning of acoustic sub-word units. In *Proc. of ACL*, pages 165–168, 2008.

[135] D. Vergyri, W. Wang, A. Stolcke, J. Zheng, M. Graciarena, D. Rybach, C. Gollan, R. Schlter, K. Kirchhoff, A. Faria, and N. Morgan. Development of the SRI/Nightingale arabic ASR system. In *Proc. Interspeech*, pages 1437–1440, 2008.

[136] O. Vinyals and S. Ravuri. Comparing multilayer perceptron to deep belief network Tandem features for robust ASR. In *Proc. ICASSP*, pages 4596 –4599, 2011.

[137] VistaWide. `http://www.vistawide.com/languages/language_statistics.htm`.

[138] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *Proc. SIGKDD*, pages 216–225, 2003.

[139] M. D. Wachter, M. Matton, K. Demuynck, P. Wambacq, R. Cools, and D. V. Compernolle. Template-based continuous speech recognition. *IEEE Trans. on Audio, Speech and Language Processing*, 15(4), 2007.

[140] L. Wilcox and M. Bush. Training and search algorithms for an interactive wordspoting system. In *Proc. of ICASSP*, volume 2, pages 97–100, 1992.

[141] Y. Yan, M. Fanty, and R. Cole. Speech recognition using neural networks with forward-backward probability generated targets. In *Proc. ICASSP*, pages 3241–3244, 1997.

[142] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary L_p forms. In *Proc. VLDB*, pages 385–394, 2000.

[143] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. the International Conference Data Engineering*, pages 201–208, 1997.

[144] Youtube. `http://www.youtube.com`.

[145] Y. Zhang, K. Adl, and J. Glass. Fast spoken query detection using lower-bound dynamic time warping on graphical processing units. In *Proc. ICASSP*, pages 5173–5176, 2012.

[146] Y. Zhang and J. Glass. Unsupervised spoken keyword spotting via segmental DTW on Gaussian posteriorgrams. In *Proc. ASRU*, pages 398–403, 2009.

[147] Y. Zhang and J. Glass. Towards multi-speaker unsupervised speech pattern discovery. In *Proc. ICASSP*, pages 4366–4369, 2010.

[148] Y. Zhang and J. Glass. An inner-product lower-bound estimate for dynamic time warping. In *Proc. ICASSP*, pages 5660–5663, 2011.

[149] Y. Zhang and J. Glass. A piecewise aggregate approximation lower-bound estimate for posteriorgram-based dynamic time warping. In *Proc. Interspeech*, pages 1909–1912, 2011.

[150] Y. Zhang, R. Salakhutdinov, H.-A Chang, and J. Glass. Resource configurable spoken query detection using deep Boltzmann machines. In *Proc. ICASSP*, pages 5161–5164, 2012.

[151] Q. Zhu, A. Stolcke, B. Chen, and N. Morgan. Using MLP features in SRIs conversational speech recognition system. In *Proc. Interspeech*, pages 2141–2144, 2005.