An Integration of Hidden Markov Model and Neural Network for Phoneme Recognition

by
Patrick Shu Pui KO

Submitted to the Department of Computer Science in Partial Fulfilment of the Requirements for the Degree of MASTER OF PHILOSOPHY in Computer Science

The Chinese University of Hong Kong
Shatin, Hong Kong
March, 1993
Abstract

Speech recognition has been a popular research topic in the past 20 years. Different approaches have been attempted by other research and they exhibit some advantages over another. Hidden Markov Model (HMM) is one common approach that has been used in many researches for the past ten years. HMM is a stochastic process which provides an efficient means of modelling the sequential structure such as speech. However, in the past five years HMM approach has been evaluated and its discrimination problem has been another popular topic. In our research, we based on our interpretation of this problem and developed a new model for phoneme recognition. Our model transforms the problem of recognizing dynamic sequential patterns using HMM into a static pattern recognition problem using an integrated HMM-Neural Network approach. We carried out our experiments using the TIMIT multiple speakers speech database. We compared our approach with the HMM approach using 600 speech samples in six phoneme classes. For a training/testing data ratio of 300/300, the integrated approach obtained an increase of 1.3% in recognition rate over the HMM. When the training/testing data ratio became 450/150, the integrated approach obtained an increase of 4.7% in recognition rate over the HMM. Based on these results, we conclude that a neural network is justified to partially solve the HMM discrimination problem.
Acknowledgements

It has been an enjoyable experience to be involved in the speech recognition project supervised by Dr. Lai Wan Chan. She is always patient to explain and discuss the difficulties and problems that we have encountered in this project. Without her valuable advice and insight, this thesis would not have been completed.
# Table of Contents

1. Introduction 1
   1.1 Introduction to Speech Recognition 1
   1.2 Classifications and Constraints of Speech Recognition Systems 1
      1.2.1 Isolated Subword Unit Recognition 1
      1.2.2 Isolated Word Recognition 2
      1.2.3 Continuous Speech Recognition 2
   1.3 Objective of the Thesis 3
      1.3.1 What is the Problem 3
      1.3.2 How the Problem is Approached 3
      1.3.3 The Organization of this Thesis 3

2. Literature Review 5
   2.1 Approaches to the Problem of Speech Recognition 5
      2.1.1 Template-Based Approaches 6
      2.1.2 Knowledge-Based Approaches 9
      2.1.3 Stochastic Approaches 10
      2.1.4 Connectionist Approaches 14

3. Discrimination Issues of HMM 16
   3.1 Maximum Likelihood Estimation (MLE) 16
   3.2 Maximum Mutual Information (MMI) 17

4. Neural Networks 19
   4.1 History 19
   4.2 Basic Concepts 20
   4.3 Learning 21
      4.3.1 Supervised Training 21
      4.3.2 Reinforcement Training 22
      4.3.3 Self-Organization 22
   4.4 Error Back-propagation 22

5. Proposal of a Discriminative Neural Network Layer 25
   5.1 Rationale 25
   5.2 HMM Parameters 27
   5.3 Neural Network Layer 28
   5.4 Decision Rules 29

6. Data Preparation 31
   6.1 TIMIT 31
   6.2 Feature Extraction 34
   6.3 Training 43
# 7. Experiments and Results

## 7.1 Experiments

### 7.2 Experiment I

### 7.3 Experiment II

### 7.4 Experiment III

### 7.5 Experiment IV

### 7.6 Experiment V

### 7.7 Computational Issues

### 7.8 Limitations

## 8. Conclusion

## 9. Future Directions

## Appendix

### A. Linear Predictive Coding

### B. Implementation of a Vector Quantizer

### C. Implementation of HMM

#### C.1 Calculations Underflow

#### C.2 Zero-lising Effect

#### C.3 Training With Multiple Observation Sequences

## References
List of Figures

Figure 1: A General Speech Recognition System 5
Figure 2: A 3-state Left-to-right HMM 11
Figure 3: A Neural Network 20
Figure 4: A Neuron 21
Figure 5: Error Backpropagation Forward Phase Algorithms 23
Figure 6: Architecture of the Integrated Model 27
Figure 7: Waveform of "aa" 32
Figure 8: Waveform of "ae" 32
Figure 9: Waveform of "ah" 32
Figure 10: Waveform of "er" 33
Figure 11: Waveform of "ih" 33
Figure 12: Waveform of "iy" 33
Figure 13: Cluster Distribution of 32 Prototype Vectors 36
Figure 14: Average Distortions of 32 Prototype Vectors 36
Figure 15: Cluster Distribution of 64 Prototype Vectors 37
Figure 16: Average Distortions of 64 Prototype Vectors 37
Figure 17: Cluster Distribution of 128 Prototype Vectors 38
Figure 18: Average Distortions of 128 Prototype Vectors 38
Figure 19: Cluster Distribution of 256 Prototype Vectors 39
Figure 20: Average Distortions of 256 Prototype Vectors 39
Figure 21: Cluster Distribution of 512 Prototype Vectors 40
Figure 22: Average Distortions of 512 Prototype Vectors 40
Figure 23 Total Average Distortion of Different Codebook Sizes 41
Figure 24 HMM Recognition Rate for Different Codebook Sizes 42
Figure 25: HMM Observation Distribution of "aa" in 3 States 46
Figure 26: HMM Observation Distribution of "ae" in 3 States 47
Figure 27: HMM Observation Distribution of "ah" in 3 States 48
Figure 28: HMM Observation Distribution of "er" in 3 States 49
Figure 29: HMM Observation Distribution of "ih" in 3 States 50
Figure 30: HMM Observation Distribution of "iy" in 3 States 51
List of Tables

Table 1: HMM Transition Matrix of "aa" 44
Table 2: HMM Transition Matrix of "ae" 44
Table 3: HMM Transition Matrix of "ah" 44
Table 4: HMM Transition Matrix of "er" 45
Table 5: HMM Transition Matrix of "ih" 45
Table 6: HMM Transition Matrix of "iy" 45
Table 7: Result of Experiment I 53
Table 8: Confusion Matrix of Experiment I with $\phi_{\text{recog}}=0$ 54
Table 9: Confusion Matrix of Experiment I with $\phi_{\text{recog}}=6$ 54
Table 10: Result of Experiment II with $\phi_{\text{train}}=3$ 55
Table 11: Confusion Matrix of Experiment II with $\phi_{\text{recog}}=0$ 56
Table 12: Confusion Matrix of Experiment II with $\phi_{\text{recog}}=6$ 56
Table 13: Result of Experiment III 57
Table 14: Confusion Matrix of Experiment III with $\phi_{\text{recog}}=6$ 58
Table 15: Result of Experiment IV 59
Table 16: Confusion Matrix of Experiment IV with $\phi_{\text{recog}}=6$ 59
Table 17: Result of Experiment V 60
Table 18: Confusion Matrix of Experiment V with $\phi_{\text{recog}}=0$ 61
Table 19: Confusion Matrix of Experiment V with $\phi_{\text{recog}}=6$ 62
1. Introduction

1.1 Introduction to Speech Recognition

Speech recognition is a difficult topic in the sense that it is closely related to the language which requires human intelligence. This topic has been studied since the 1950's, and a significant increase in the level of activities was observed following the support of the Advanced Research Project Agency in 1971 [1]. In the past twenty years, most of the research in speech recognition can be classified by the approaches they attack this problem, or in a more general sense by the "level" of recognition. Basically, a speech recognition system usually falls into one of the following categories:

- Isolated subword recognition
- Isolated word recognition
- Continuous speech recognition

1.2 Classifications and Constraints of Speech Recognition Systems

1.2.1. Isolated Subword Recognition

A word usually consists of more than one subword. A vowel, a diphthong, or a consonant are all considered as subwords. For example, the word "meet" (mi:t) consists of three subwords, consonants "m", "t" and a vowel "i:". These subwords are usually referred to as phonemes. In isolated subword recognition experiments, the boundaries of the speech samples are usually pre-identified. Context sensitivity is one of the problems in phoneme recognition experiments. By "context sensitive" it means that subwords are easily affected by their positions in a word which we call the "coarticulation effect". For example, the vowel "a:" in the word "bar" (ba:) differs from the vowel "a:" in "cart" (ka:t) since the latter is affected by the consonant "t". In general, phoneme recognition experiment results vary from 35% to 70% of recognition rate depending on different
sets of subwords used. For example, in one of the phoneme recognition experiment in Carnegie Mellon University, the recognition rate achieved was 49.78% using a single codebook [2].

1.2.2 Isolated Word Recognition

Various attempts have been made to recognize isolated words. In this kind of experiment, word boundaries are usually assumed. The size of the vocabulary varies from ten digits, the set of alphabets to several hundreds of words. The recognition rate in these experiments can be as high as 98% given certain constraints such as simple vocabulary set or trained speakers [3].

1.2.3 Continuous Speech Recognition

The research in continuous speech recognition aims at designing a system which recognizes a sentence rather than distinctly spoken words. The word boundaries in this kind of experiment are usually unknown. The set of recognized sentences is usually governed by a simple grammar. The best example of this kind of experiment is the SPHINX system in CMU [4].

Besides different levels of recognition, a speaker-independent recognition system also introduces a level of difficulties. A report from Levinson showed that the recognition accuracy decreased from 88.3% for speaker-dependent systems to 65.1% for speaker-independent systems [4].

One point worths pondering is that comparing different recognition systems by their recognition rates is not very meaningful. Factors such as differences in the vocabulary sets, number of speakers, quality of speech samples etc. affect a subword or isolated word recognition system significantly. For continuous speech recognition, the grammar as well as the perplexity (i.e. average possible number of words following each word) vary. One should not attempt to compare the results unless some sort of normalization can be carried out.
1.3 Objective of the Thesis

1.3.1 The Problem

This thesis focuses on the problem of phoneme recognition. To be more specific, this thesis attempts to integrate the Hidden Markov Model (HMM) with neural networks. We try to give a comparison of this approach with the traditional Hidden Markov Model approach by some experiment results. As discussed above, phoneme recognition is a difficult problem in the sense that it is greatly sensitive to the context. This is reflected in our experiments where each phoneme sample set is extracted from parts of different words. We aim at improving the recognition performance of HMM by adding a neural-network layer to compensate HMM's inability in discrimination.

1.3.2 How the Problem is Approached

HMM is a common speech modelling method in the 80's. The advantages of HMM are small storage, efficient calculations and its unified approach in modelling phonemes, words, and even sentences. One major weakness which has been a hit topic since 1989 is the discrimination ability. In our experiment, we identified this weakness and designed a neural network layer in order to perform a more accurate recognition task.

1.3.3 The Organization of this Thesis

This thesis begins with a literature review in Chapter 2. In the review, several common approaches in speech recognition will be introduced. In Chapter 3 the problem of discrimination of HMM will be discussed in detail. Some background information of neural networks follows in Chapter 4 and we particularly describe the error back-propagation learning algorithm which will be used in our model. The rationale behind is narrated and followed by the description of our proposed model in Chapter 5 in detail. Chapter 6 provides various experiment configuration information as well as introducing the phoneme database, TIMIT. Chapter 7 discloses our experiment results. Finally,
some discussion and conclusion will be followed. Last but not least, we give some thoughts of possible future research.
2. Literature Review

2.1 Approaches to the Problem of Speech Recognition

In general, all speech recognition systems can be conceived as in Figure 1. Real-time, analog speech signals are usually sampled and digitized to yield discrete, digital signals to be processed by digital devices, especially computers. In the second step, the digital signals are processed to extract various interpretation. Such interpretation includes spectral information (e.g. Fourier transforms) as well as other information based on wave-form encoding. Thus, at this stage an abstraction of raw speech signal to a sequence of tokens has been developed.

![Figure 1: A General Speech Recognition System](image)

The sequence of tokens can be further interpreted as generated from a speech model. A speech model is an abstract concept that a particular system assumes what the structures of various pieces of speech should be. Such information is stored in the reference model base which is available to
the model matching process. It produces a set of hypothesis for higher-level speech processing in
the speech hierarchy which the scope of the system assumes. Higher-level speech processing is
usually guided by a set of rules, for example, simple grammar. As in human speech recognition,
ambiguity may be resulted from multiple hypothesis. Thus, higher-level speech processing usually
considers the information such as semantics and context in order to generate the final recognition
result.

Various approaches have been attempted in the last twenty years. They include the traditional
template-based approach [1], the knowledge-based approach [5], the stochastic approach [6], and
the connectionist approach [7]. Most of the speech recognition systems based on these approaches
are analogous to the general speech recognition system described above. They differ mostly in the
modelling process. Thus, by reviewing these approaches as well as identifying where their major
differences are with respect to the general recognition system, we can compare their advantages
and limitations, and to derive better models.

2.1.1 Template-Based Approaches

The underlying idea of template-based approach is to store a set of prototypical speech patterns
known as "templates". An unknown spoken utterance is then matched against each of these
templates and the best-matcher becomes the recognized word. In general, "templates" are the
extracted speech features. Filter banks are among the most popular features used [1]. A frame of
speech signal is usually passed through a bank of bandpass filters, which covers the speech band
from 100 Hz to a cutoff frequency, for example, 3000 Hz. The number of filters varies from 5 to
32 and the filter spacing can be either linear or logarithmic [3]. The resulting coefficients from
each band form a feature vector, or a pattern.

During training, speech samples are abstracted to form feature vectors. Statistical methods are
then applied to the feature vectors to build a template base. These feature vectors are called the
reference patterns. During recognition, the template base is consulted and the best matching template becomes the recognized pattern of an unknown pattern (test pattern).

In practice, the length of a test pattern $T(t)$ is usually different from the reference template $R(t)$. Thus, a function $w(t)$ is needed to map $T(t)$ to $R(w(t))$ so that each $R(t)$ is associated with zero or more than one $T(t)$. This function $w(t)$ is usually called a time-warping function. Mathematically finding $w(t)$ is similar to the problem of finding geodesic where we minimize the distance $D(T,R)$:

$$D(T, R) = \min_{\{w(t)\}} \int_0^T d(T(t), R(w(t))) dt$$  \hspace{1cm} (1)

Here $d$ is the distance function and Euclidean distance is often used. The Euclidean distance is defined as:

$$d(T, R) = \|T - R\| = \sum_{i=0}^P (T_i - R_i)^2$$  \hspace{1cm} (2)

LPC log likelihood distance proposed by Itakura [5] is also common:

$$d(T, R) = \log \left[ \frac{a_R^T V_T a_R}{a_T^T V_T a_T} \right]$$  \hspace{1cm} (3)

where $a_R$ and $a_T$ are the LPC coefficient vectors of the reference and test frames respectively. $V_T$ is the matrix of autocorrelation coefficients of the test frame. This measure gives different weights to each dimension of a vector and is generally considered as a better measure than the Euclidean distance for speech [3].
For digital signal processing, we have a solution to the problem shown in equation (1). Suppose there are \( N_T \) tokens in a test template \( T \):

\[
T = \{ T(1), T(2), ..., T(N_T) \}
\]  

(4)

and \( N_R \) tokens in a reference template \( R \):

\[
R = \{ R(1), R(2), ..., R(N_R) \}
\]  

(5)

and we constrain \( w(1) = 1 \) and \( w(N_T) = N_R \).

There exists several methods to find \( w \) but the most elegant one is the dynamic time warping (DTW) [3] which finds out the minimum distance \( D^* \).

\[
D^* = \min_{w(n)} \left[ \sum_{n=1}^{N_T} d(T(n), R(w(n))) \right]
\]  

(6)

The DTW method usually makes use of dynamic programming to solve the recursive equation:

\[
D_A(n, m) = d(T(n), R(m)) + \min_{q \leq m} \left[ D_A(n-1, q) \right]
\]  

(7)

where \( D_A(n, m) \) is the accumulated distance from \( n=1 \) and \( m=1 \).

There also exist variants of DTW which relax the fixed-end-points constraint [3]. They apply to the cases where the end-points of a speech cannot be detected accurately.
To make use of the DTW for connected-word recognition, we can perform an exhaustive search on a test pattern $T(n)$ so that the closest corresponding string $R_1(n_1)R_2(n_2)\ldots R_L(n_L)$ consisting $L$ words can be found. However, this approach is computationally intractable since we need $V^L$ attempts given $V$ the size of vocabulary. Alternatively, an approach called "level-building" is often used. Level-building basically divides the calculation of $D_A$ into $L$ levels, where in each level the calculation is similar to the normal constraint DTW with several accumulated parameters kept as the initial conditions of the next level [3].

2.1.2 Knowledge-Based Approaches

The knowledge-based approach incorporates the experts' knowledge of the speech as part of the constraints in the search space. This approach assumes that the more we understand the internal structure of the speech, the better we recognize the speech based on our knowledge-base. Although among the major arguments of this approach there exist doubts on the ability to discover and represent the "knowledge", an experiment conducted by Cole in 1978 gave the implication that there are places for the knowledge-base approach to develop [5].

In that experiment, a trained spectrogram reader was asked to provide a phonetic transcription based on a spectrogram of unknown utterances. A group of phoneticians were asked to listen to the spoken version of the same utterances (which might be syntactically and semantically anomalous) and gave their phonetic transcription as well for references. Surprisingly enough, the results from both groups were close and accurate up to over 90% [5]. With these results, some researchers expected that one day the experts' knowledge of spectrogram reading can be replaced by machines through knowledge engineering.

Common knowledge-base includes the knowledge on auditory modeling such as the concept of critical-band which weights some filter banks more than the rest. This simulates the response of
human's auditory system to various frequencies. Another knowledge is the set of phonotactic constraints. For example, in English vocabulary a "q" never goes after a "t", etc.

In one experiment conducted by Cole [8] the set of letters and digits were tested. Major features for classifying were analyzed and extracted. These features included the formant frequencies, formant slopes, the ratio of high frequency energy to low frequency energy, etc. Clusters were created for the same features for different speakers in order to perform speaker-independent recognition. A tree structure was created where each node represented a decision made on one of the features. The system achieved an accuracy of 89% on the letters and digits. And when tested with a confusing set of letters, the result outperformed the frame-by-frame template-based approach which used spectral parameters only.

Nevertheless, the complex structure of speech is yet to be discovered. Before a complete and sound speech knowledge-base to come into practice, other approaches which provide ignorance modelling prevail. Ignorance modelling [1] refers to the mechanisms to model the unknown structure. For example, in Hidden Markov Models speech is represented by a set of probabilistic density functions which are derived from speech samples.

2.1.3 Stochastic Approaches

The Hidden Markov Model (HMM) is the most popular stochastic model employed in speech recognition research [9]. The best part of this model is the provision for capturing the ignorance (i.e. the unknown internal structure) of the speech. First its transition matrix captures the temporal information and second, its output distributions matrix captures the spectral variations of the speech. In addition, efficient algorithms exist which make the HMM more expandable for large systems [6].

Definition
A Hidden Markov Model (HMM) is an \( N \)-state model completely described by the tuple \( \lambda = (A, B, \pi) \). \( A = \{a_{ij}\} \) is defined as the transition probability which is the probability density function of the state transition from \( i \) to \( j \) at any discrete time \( t \) where \( i, j \leq N \), and a transition always results in generating a discrete observation \( O_t \). The probability of generating which observation is governed by \( B = \{b_{jk}\} \) which is defined as the probability density function of generating \( v_k \) at state \( j \), where \( 1 \leq k \leq M \) and \( v_k \) is a discrete observation. At \( t=0 \), the state of a HMM is determined by \( \pi = \{\pi_i\} \), the initial probability, which is defined as the probability density function in state \( i \) at \( t=0 \). Thus, for a given length of time \( T \), an HMM will traverse from an initial state to its final state and will generate \( T \) discrete observations while the actual path the HMM undertakes from state to state is unknown (hence "Hidden").

When applied to speech recognition, a special case of HMM which we call left-to-right model [6] is usually used. The states in such a model are ordered, and the probability of transition from the current state to a "previous" state is zero by definition. In addition, the initial probability is usually preset such that the probability of starting from the first state is 1 and others zero. Figure 2 shows an example of a 3-state left-to-right HMM:

![A 3-state Left-to-right HMM](image-url)
Training

In the HMM methodology, a common practice to adapt an observation sequence \( O = O_1, O_2, ... \) to an HMM is by a procedure called the Baum-Welch Reestimation [6]. This procedure can be used to adjust the \( A \) and \( B \) parameters of a HMM. In order to achieve efficient calculations, a faster method which calculates \( \alpha_t(i) \) and \( \beta_t(i) \) was developed [6]. \( \alpha_t(i) \) is defined as \( P(O_1, O_2, ... \text{state } = i \text{ at time } t \mid \lambda) \). \( \beta_t(i) \) is defined as \( P(O_{t+1}, O_{t+2} ... O_T \mid \text{state } = i \text{ at time } t, \lambda) \).

Forward Procedure:

\[
\alpha_{t+1}(j) = \sum_{i=1}^{n} \alpha_t(i) a_{ij} b_j(O_{t+1}) \quad 1 \leq t \leq T - 1
\]  

(8)

Backward Procedure:

\[
\beta_t(i) = \sum_{j=1}^{n} a_{ji} b_j(O_{t+1}) \beta_{t+1}(j) \quad T - 1 \geq t \geq 1
\]  

(9)

When we calculate the \( \alpha_t(i) \) and \( \beta_t(i) \), we need to build a trellis structure [6] to calculate them. That is, for \( \alpha_t(i) \), we start with \( t=1 \), calculate all \( \alpha_1(i) \), and store them for reference before calculating \( \alpha_2(i) \), and so on. Similar to \( \beta_t(i) \), we start with \( t=T \), calculate all \( \beta_T(i) \), and store them for reference before calculating \( \beta_{T-1}(i) \). During initialization, we put \( \alpha_1(i) = \pi_i b_i(O_1) \) and put \( \beta_T(i) = 1 \). Baum-Welch reestimation formula is applied for each iteration until the parameters converge.

Baum-Welch Reestimation Formula:

\[
\gamma_{ij} = \frac{1}{P} \sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)
\]  

(10)
\[ \gamma_i = \sum_{j=1}^{n} \gamma_{ij} \]  

(11)

\[-a_{ij} = \frac{\gamma_{ij}}{\gamma_i} \]  

(12)

\[ b_{ij}(k) = \frac{\sum \alpha_t(j)\beta_t(j)}{\sum \alpha_t(j)\beta_t(j)} \]  

(13)

Note that in practical speech recognition problem, there are often more than one training observation sequence. The above formula can also adapt to this problem by a few modifications. Thus, the whole HMM training procedure for classifying \( V \) speech classes consists of training \( V \) HMM models with corresponding sets of observation sequence \( O^V \) where \( v \leq V \).

**Recognition**

Given an unknown class speech sequence, we may use HMM to classify this sequence by scoring it against \( V \) HMM's using the forward procedure (8). This procedure can be used to efficiently calculate the probability of a sequence generated from a given HMM [6], that is:

\[ P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \]  

(14)

Thus, if we were to classify a sequence from \( V \) HMM models, we need to score that sequence against each one of the \( V \) models. The one which gives the highest probability score [6] is considered the class which the sequence belongs to.
2.1.4 Connectionist Approaches

Despite the fact that the neural network paradigm has the potential to offer massive parallelism, recent research using neural networks showed that this approach can be used to obtain comparatively higher recognition rate on the phoneme level without any grammatical constraints [10]. This characteristics is important since the accuracy of low-level speech recognition directly affects the performance of higher level recognition such as word level or continuous speech level.

Research in speech recognition using neural networks generally falls into one of the following categories:

- Use neural networks such as multi-layer perceptrons to analyze static speech patterns
- Use recurrent or time-delayed neural networks to perform dynamic speech analysis
- Integrate neural networks with other models

In the first category, the speech samples are often pre-segmented before training or recognition. In speaker-dependent, small vocabulary set experiments, it shows that the accuracy of this approach can be comparable to HMM or template-based approaches [7].

However, this approach cannot be practically applied to more difficult problems such as a larger set of vocabulary or continuous speech recognition since it requires pre-segmentation. Thus, some researchers designed some kinds of recurrent [11] or time-delayed neural network [10] to cater for the temporal or dynamic nature of speech.

In an experiment performed by Waibel et al [12], $N$ delayed connections in addition to each original connection were connected to each input unit such that there were $N+1$ weights. This network is known as a Time Delayed Neural Network (TDNN). The architecture of this neural
network was so designed to relate the speech events in time without precise pre-segmentation of the input speech sample. In practice, the TDNN consisted of two hidden layers. The delay factors in these layers could be different. In Waibel's experiment, the first hidden layer was associated with a delay factor of $N=2$ where the second hidden layer was associated with $N=4$. Back-propagation learning procedure was used. The experiment tried to distinguish between three phonemes, namely, "ba", "da", and "ga". Shift-invariance tests were also included which placed the phonemes 30 ms misaligned in the input units. The results showed that the TDNN could obtain a result better than an HMM as well as being shift-invariance to a certain extent [12].

Despite all these facts about TDNN, the computational requirements of a TDNN are usually several orders above other approaches [12]. This may be one reason which hindered more further experiments on TDNN by other researchers that could have been observed in recent years.

Various attempts in combining neural networks with conventional approaches especially HMM are also observed in recent years. One of the most successful direction was headed by Bourlard et al [13] [14]. He trained a recurrent multilayer perceptron (MLP) such that it acted as an emission-probability generator of a HMM. The MLP they used took the speech context (neighbour frames) into consideration. That is, besides the current frame in use during training, a finite number of frames for left-context and right-context were also used in the input layer of the MLP. The phoneme classification result proved that this approach improved the recognition rate based on HMM with traditional training method from 44.8% to the best of 62.7% [13].
3. Discrimination Issues of HMM

3.1 Maximum Likelihood Estimation (MLE)

As we have seen, the HMM exhibits a powerful ability to deal with sequential structure although it possesses some limitations which cannot model speech perfectly. However, the standard HMM training is often described as a "within-class" method [11]. Consider the expansion:

\[ P(\lambda_i|O^v) = \frac{P(O^v|\lambda_i)P(\lambda_i)}{P(O^v)} \]  

(15)

where \( P(\lambda_i|O^v) \) is the measurement of the closeness of a given observation sequence \( O^v \) to a model \( \lambda_i \) during training. Since there exist \( V \) classes, \( P(O^v) \) is in fact contributed by all the \( V \) classes:

\[ P(O^v) = \sum_{i=1}^{V} P(O^v|\lambda_i)P(\lambda_i) \]  

(16)

In general, for a training sequence \( O^v \), we want to maximize

\[ P(\lambda_i|O^v) = \frac{P(O^v|\lambda_i)P(\lambda_i)}{P(O^v|\lambda_i)P(\lambda_i) + \sum_{k \neq i} P(O^v|\lambda_k)P(\lambda_k)} \]  

(17)

However, in standard HMM training, given an observation sequence \( O^v \) belonging to class \( v \), we generally simplify equation (17) by maximizing only on

\[ P^* = \max_{\lambda_i} P(O^v|\lambda_i) \]  

(18)
where \( \lambda_i = (A_i, B_i, \pi_i) \) instead of also minimizing the whole denominator in (17). This simplified optimization criterion is generally referred to as Maximum Likelihood Estimation (MLE). The standard training procedures Baum-Welch Reestimation (8)-(13) as well as the Viterbi algorithm [14] fall into this category.

This method, although being quite satisfactory in some experiments, lacks discriminative ability in distinguishing between classes. This problem becomes more obvious as the number of classes becomes large.

### 3.2 Maximum Mutual Information (MMI)

If we consider maximizing (17) other than MLE, we can define \( I^*_v \) such that it maximizes the logarithm of (17):

\[
I^*_v = \max_{\lambda_i} \log \left[ \frac{P(O^v|\lambda_i)P(\lambda_i)}{P(O^v|\lambda_i)P(\lambda_i) + \sum_{k\neq i} P(O^v|\lambda_k)P(\lambda_k)} \right]
\]

which yields

\[
I^*_v = \max_{\lambda_i} \left[ \log P(O^v|\lambda_i) - \log \sum_{w=1}^v P(O^v|\lambda_w) \right]
\]

assuming

\[
P(\lambda_i) = P(\lambda_j) \forall i, j
\]
We call $I^*_v$ the Maximum Mutual Information (MMI) for the observation sequence $O^v$. That means, the training takes into considering that a training sequence $O^v$ belongs to a class $v$ and also exclusive from other classes. Thus, the overall optimization procedure becomes:

$$I^* = \max_v \sum_{v=1}^{V} \left[ \log P(O^v | \lambda_v) - \log \sum_{w=1}^{W} P(O^v | \lambda_w) \right]$$  \hspace{1cm} (21)

However, the analytical method for this criterion fails since convergence is not guaranteed [6].
4. Neural Networks

Albeit the fact that HMM is a useful tool in recognizing dynamic sequential structures, it is obviously not discriminative. On the other hand, neural networks are often described as an approach to learn the internal structure from the training data and to discriminate among them. In this chapter we describe the basic concepts of neural networks. Since neural networks have exploded into a broad area of research, we limit our scope of neural networks particularly to the multi-layer perceptron (MLP).

4.1 History

The history of neural networks can be traced back to 1943 when Warren McCulloch and Walter Pitts wrote the paper "A logical calculus of the ideas immanent in nervous activity" which showed that the neural networks could compute any arithmetic function [15]. Before 1958, various papers were published about neural networks but none of them actually implemented a neural network to achieve a specific function. In 1958, Frank Rosenblatt et al developed the first neurocomputer which was called the "Mark I Perceptron" to perform tasks such as pattern recognition. However, following the publish of the book "Perceptrons" by Minsky and Papert in 1969 which mathematically proved that a neural network at that time was not capable to learn the exclusive-or function, over a decade of silent years came. Following this "dark age," more research in neural networks reopened after the Defense Advanced Research Projects Agency (DARPA) began funding neural networks research in 1983 [15]. From 1983 onwards, John Hopefield played a key-role in promoting the field of neural networks through his papers and lecturers. In 1986, one of the famous classical books on neural networks edited by David Rumelhart and James McClelland, "Parallel Distributed Processing", was published. Solutions to some vital problems in the 1960s were provided in this book with new algorithms and research results [15].
4.2 Basic Concepts

A neural network is basically a model which can be conceived as a connected graph. Each node in the graph is called a neuron. A connection between two neurons is usually referred to as a weight. A neural network is often organized as layers. A layer is a group of neurons which serves the same purpose. For example, an input layer is the group of neurons where signal inputs to the neural network. An output layer is the group of neurons where responses are observed. A hidden layer is an intermediate layer in between [16]. Figure 3 shows a general neural network model:

![Figure 3: A Neural Network](image-url)
A neuron can be conceived as a processing element which has its own storage and transfer function. A storage is another source of input and output which is local to the neuron. A transfer function, sometimes referred to as an activation function, is a mapping between the inputs to the neuron and outputs. A common example of transfer function is summation of the products of inputs and weights. Figure 4 shows the organization within a neuron [15]:

4.3 Learning

Neural networks learn by adapting their weights during a dedicated phase for training. Weights connected to each neuron are adapted such that the global network goal is achieved, for example, minimizing mean squared error. There are in general three types of training: supervised training, reinforcement training, and self-organization [15].

4.3.1 Supervised Training
Suppose $x$ stands for the input vector and $y$ stands for the output vector. Multiple training $(x,y)$ pairs are usually presented to the neural network as examples. For each example pair, a score is given through a cost function, or built in the transfer function of the output neurons. The neural network then adjusts its weights using this score such that for every given $x$, the corresponding $y$ can be generated to obtain higher score (lower error).

### 4.3.2 Reinforcement Training

Reinforcement training differs from supervised training in that it is guided by one score for a set of $(x,y)$ pairs instead of one pair. One of the characteristics is that sometimes we may not have a definite score for a $(x,y)$ pair and the score depends on a group of $(x,y)$ performance.

### 4.3.3 Self-Organization

Self-Organizing neural networks adapt their weights through a cost function alone. No $(x,y)$ pair is presented during training phase. One example is the Kohonen learning rule. The processing elements compete with each other with inputs. The winner sets its output to 1, losers set it to 0. The Kohonen learning rule then adjust the weights according to outputs and previous weights. One application of Kohonen learning rule is to find the famous statistical problem K-means [15].

### 4.4 Error Back-propagation

We used supervised training in our experiments. Specifically, we used a multi-layer perceptron trained with the error back-propagation learning algorithm. This set of learning rules is also known as the general delta rules [16] and it consists of two training phases, namely forward phase and backward phase. The whole network is trained by iteratively running through these two training phase for a number of cycles and in each cycle an input pattern $i_p$ and a target $t_p$ is presented to the network. There can be a total of $N$ different patterns and this process continues until a error value $E$ drops to a minimum.
In the forward phase, a value $\delta_p^j$ is calculated for every neuron $j$ and the $p^{th}$ training pattern. The exact calculations vary and depend on whether $j$ is an output unit or hidden unit. If $j$ is an output unit which outputs $o_p^j$ but we expect a target value $t_p^j$, we set:

$$\delta_p^j = (t_p^j - o_p^j)f'_j(\text{net}_p^j)$$  \hfill (22)

If $j$ is a hidden unit and outputs $\text{net}_p^j$, we set

$$\delta_p^j = f'_j(\text{net}_p^j)\sum_k \delta_p^k w_{kj}$$  \hfill (23)

where $w_{kj}$ refer to the weight which connects the hidden unit $k$ and the unit $j$ in the next forward layer. Figure 5 shows a multi-layer perceptron labelled with the required calculations in the forward phase:

![Figure 5: Error Backpropagation Forward Phase Algorithms](image)
In the backward phase, each weight $w_{ji}$ in the network will be added with the value $\Delta_p w_{ji}$ calculated by the following algorithm:

$$\Delta_p w_{ji} = \eta \delta_p o_{pj}$$

(24)

where $\eta$ is known as the learning rate and is usually set empirically.

At the end of each cycle, $E_p$ is calculated as a mean-squared value:

$$E_p = \sum_j (t_p - o_p)^2$$

(25)

Thus, minimization of the $E$ is possible through the minimization of the summation of $E_p$ for all $p \leq N$ as we define:

$$E = \sum_p E_p$$

(26)

This algorithm can be proved to exhibit a gradient-descent characteristic [16]. In practice, we often train the neural network until $E$ drops to a very small value close to zero.
5. Proposal of a Discriminative Neural Network Layer

5.1 Rationale

In Chapter 3 we have mentioned the discrimination issue of traditional HMM approach. In this chapter, we shall give our interpretation of this problem and propose an integrated HMM-Neural Network model. Our interpretation of the discrimination problem can be described in the following simplified view which pinpoints one weakness of HMM.

Suppose a 2-state HMM $\lambda_A$ generates the following class of observations:

- Class $A_1$: $v_1 \ldots v_4, v_3 \ldots v_3$
- Class $A_2$: $v_2 \ldots v_2, v_4 \ldots v_4$

and another 2-state HMM $\lambda_B$ generates the following class of observations:

- Class $B_1$: $v_1 \ldots v_4, v_4 \ldots v_4$
- Class $B_2$: $v_2 \ldots v_2, v_3 \ldots v_3$

Obviously, in both HMM's, the observation probability distributions for $v_1$ and $v_2$ in the first state will be higher, and the observation probability distributions for $v_3$ and $v_4$ in the second state will also be higher. On the other hand, the probability of $v_3$ and $v_4$ in the first state, and $v_1$ and $v_2$ in the second state will be lower. Thus, if we evaluate the probability of a class $B$ sequence $O = v_1 \ldots v_4, v_4 \ldots v_4$, using the forward equation (8), the probabilities of $P(O|\lambda_A)$ and $P(O|\lambda_B)$ will be very close and confusing. This situation will be worse as the number of classes increases.

This problem is mainly due to the fact that in recognizing a sequence $O=O_1O_2\ldots O_T$ against a set of HMM's, we only consider the final scores of the calculations of $P(O|\lambda_g)$, without using the
trajectory information (14). By trajectory information we mean the association of each $O_i$ to a state in the HMM, as well as the exact pattern of the observation symbol $\{O_i\}$ from $t=0$ to $t=T$.

Refer to the above example, if it is possible to have a model to store the patterns $(v_{1, state 1})$, $(v_{1, state 1}) (v_{3, state 2})$ for class $A_j$, and solve it as a pattern recognition problem, we may avoid such confusion described above.

Since neural network is a common tool in handling pattern recognition problems [15], and it offers the ability to learn from the examples instead of specifically coding the knowledge and specifying the knowledge structure, we initially investigated a HMM-neural network integrated model. We encountered one major problem which was the length of a pattern. Neural networks are usually presented a number of static or fix-length patterns during each training cycle. However, the length of the observation sequences in a speech experiment varies. In order to solve this mismatch, we now compress each dynamic pattern into a static pattern by enforcing some constraints which will be described in the next section. As a whole, we can train the neural network with static HMM segmentation patterns after applying Viterbi algorithm [6]. By using these patterns, we hope that some trajectory information can be learned by the neural network instead of relying on the first-order HMM where the probability of entering a state depends only on the previous one.

As we see in Figure 6, the model architecture consists of low-level feature extraction sub-systems such as LPC and vector quantizer. In this chapter we focus on the HMM and neural network layers. LPC will be described in Appendix A and vector quantization will be described in Appendix B.
5.2 HMM Parameters

This section introduces the procedure of extracting the HMM observation distribution parameters for the neural network layer as another source of knowledge.

For a set of \( V \) classes of phoneme, we employ traditional HMM training procedures (8)-(14) to come up with \( V \) \( N \)-state HMMs for their corresponding classes. These HMMs are labeled as \( \lambda_v \) where \( v \in V \). Then, for each training sequence \( O^v \) of class \( v \), we segment it into \( N \) subsequences:

\[
O^v = \{ [O_1^v = v_{k_1}, O_2^v = v_{k_2}, \ldots, O_{k_1}^v = v_{k_1}], [O_{k_1+1}^v = v_{k_{k_1}}, O_{k_1+2}^v = v_{k_{k_1}}, \ldots, O_{k_2}^v = v_{k_2}], \ldots, [O_{k_1}^v = v_{k_1}] \}
\]  

(27)
such that each subsequence is generated from its corresponding state in the $N$-state HMM. This segmentation procedure is carried out using the Viterbi algorithm. After segmentation, for each training sequence we associate each observation with its $B$ parameter in $\lambda_v = (A_v, B_v, \pi_v)$. $B_v$ is a $N \times M$ matrix of real numbers in the interval $(0,1)$ where $M$ is the size of the discrete observation set. Here we perform some tricks to convert $B$ to $L'$ in order to be suitable for neural network training. For each observation in the subsequence, we mark its corresponding position in $B_v$ as a "1". For example, the above observation will be marked as:

$$B_v = \{b_{1k_1} = 1, b_{1k_2} = 1, ..., b_{1k_q} = 1\}, \{b_{2k_{p+1}} = 1, b_{2k_{p+2}} = 1, ..., b_{2k_q} = 1\}, ..., \{b_{Nk_{p+N-1}} = 1\}$$ (28)

In addition, if a position is marked more than once it is retained as "1". On the other hand, if a position is never marked it is set to "0". Thus, for each training sequence $O$ we can transform it into a bit-matrix $B'$. This constraint is considered as a compression procedure to obtain a static pattern. The following example explains better.

Suppose we have a four-observation-symbol, bi-state-HMM system. The four observation symbols are $v_1, v_2, v_3, v_4$. Thus, $B'$ is a $2 \times 4$ matrix. The example observation $O = v_1 v_2 v_3 v_4$ can be converted to $B' = [1, 0, 0, 0] [0, 0, 0, 1]$ given the HMM $\lambda_B$.

### 5.3 Neural Network Layer

The bit matrix $B'$ becomes the input pattern to the multi-layer perceptron. Thus, the dimension of the input layer of the multi-layer perceptron (MLP) is $NM$. We used one hidden layer of 20 units and a output layer of $V$ units in our experiment. During training, a target vector $T_v = (0,0,...1...0)$ is constructed for each training bit matrix $B_v$ where the $v^{th}$ position in $T_v$ is set to 1 and others set to 0. A gradient descent algorithm (Error Back-propagation) is used to train the MLP.
5.4 Decision Rules

Although there should be a lot of information contained in the segmentation pattern $B_\nu$, we cannot barely rely on the MLP recognition layer and completely ignore the probability scoring of the HMM's. Instead, we allow only the observation sequences which are "close" to each other to be recognized by the MLP for final decision. That means, if an observation sequence scores "highly" and can be distinguished from the rest of the classes, we can skip the MLP classification. We define the order of closeness $W$ between two classes $\nu_1$ and $\nu_2$ for an observation $O^{\nu}$ as the difference of the log probabilities $P(\lambda_1|O^{\nu})$ and $P(\lambda_2|O^{\nu})$. That is,

$$W_{i,j}(O^{\nu}) = |\log P(\lambda_i|O^{\nu}) - \log P(\lambda_j|O^{\nu})| \quad (29)$$

Thus, the training procedure is:

1. Every $O^{\nu}$ is converted to $B_{\nu}'$ from $\lambda_{\nu}$, and the corresponding target vector $T_{\nu}$ is created as training pattern.

2. Select a threshold value $\phi_{\text{train}}$ where $W_{i,j}(O^{\nu}) \leq \phi_{\text{train}}$ such that every $O^{\nu}$ bound by $\phi_{\text{train}}$ is converted to $B_{\nu}'$ from all $\lambda_i$. The corresponding target vectors $T_{\nu}$ is created as training pattern.

Note that $\phi_{\text{train}}$ is calculated from $W_{i,j}(O^{\nu})$ based on empirical data.

The recognition rules are:

1. Select a threshold value $\phi_{\text{recog}}$ such that for an unknown observation sequence $O^{\nu}$, if there exists $u$ for all $i \neq u$, and $W_{i,u}(O^{\nu}) > \phi_{\text{recog}}$, $O^{\nu}$ is recognized as belonging to class $u$.

2. If there exist $i$ such that $W_{i,u}(O^{\nu}) \leq \phi_{\text{recog}}$ for some $i \neq u$, convert $O^{\nu}$ to $B_{i}'$ for all $\lambda_i$. All $B_{i}'$ and $B_{u}'$ are fed into the MLP for recognition. A number of output vectors $U_i$ and $U_u$ are then
obtained. $O^v$ will be recognized as from class $j$ if the value in the $j^{th}$ position of $U_j$ is the maximum among all $U_i$'s where $i \neq j$. 
6. Data Preparation

6.1 TIMIT

Our experiments extracted training and test data from the TIMIT speech database [17]. TIMIT contains a total of 6300 sentences of speech samples, with 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. These speech samples were recorded in a sound-proof room, and were in digital, 16-bit PCM format with a sampling frequency of 16kHz of the waveform of the speech source. We selected 20 males' speech samples from dialect region one. Since every speaker spoke only around five sentences, we needed to extract the phoneme using the labeling index (built by "hand-segmentation") provided by TIMIT. Six phoneme groups were finally extracted. They were "aa", "ae", "eh", "er", "ih", and "iy". Figure 7-12 show typical waveform of these phoneme from the TIMIT. Each group consists of 50 training samples and 50 test samples. Thus, there were altogether 300 training samples and 300 test samples. The issue of multiple-speaker was not specially handled and we assumed that the data fell into the same cluster.
Figure 7: Waveform of "aa"

Figure 8: Waveform of "ae"

Figure 9: Waveform of "ah"
Figure 10: Waveform of "er"

Figure 11: Waveform of "ih"

Figure 12: Waveform of "iy"
In the above figures, we can observe that there are some pseudo-stationary periods in each phoneme. For example, in Figure 7-12 the vowels show a pseudo-stationary periods in the middle of their waveform. A pseudo-stationary period naturally forms a distinct state in a HMM. From Figure 7-12 these periods seldom occur in the beginning nor in the end of the speech. In the boundaries the phoneme is easily affected by the context. In 6.3 we shall present the HMM’s of our experiments in numerical forms. We shall see that in the middle state of a 3-state left-to-right HMM, the second state is usually subject to fewer variations in terms of observation distributions.

6.2 Feature Extraction

In order to compare our recognition results with the HMM phoneme recognition experiment performed by Lee [2] using TIMIT, we adopted the same 12-order normalized LPC cepstrum encoding on the samples where each frame spanned 10ms. The value of 10ms was chosen for the theoretical reasons that the properties of speech remain roughly invariant in the short-time analysis of speech [18]. Neighboring frames overlapped for 5ms. Thus, we collected a total of 11226 frames for all the samples. The cepstrum vectors were then vector quantized according to the algorithm described in Appendix B to create a size-128 codebook. All the cepstrum vectors (11226 vectors) were then quantized to a index which referred to one of the entry in the codebook. The size of our codebook was determined from one of our preliminary experiments which found out the best codebook size for the TIMIT data. As we mentioned in Appendix C.2, the larger the size of the codebook makes the zero-lising effect of HMM worse for limited training data. On the other hand, if the codebook size is too small, the codebook cannot cater with the variety of information contained in the speech data. Thus, we have carried out a preliminary experiment to attempt various codebook with sizes 32, 64, 128, 256 and 512. There are several criteria we need to consider. First, is the number of vectors represented by each prototype vector "average" enough, or is it dominated by a few prototype vectors? Second, is the average distortion small enough? Third, is the codebook size too large for the HMM? That is, is the recognition rate negatively affected by the codebook size? And finally, is it feasible to carry out our integrated model
experiment using the codebook of this size? Figure 13-24 show the frequencies of the vectors represented by prototype vectors, and the average distortion of each codebook size:
Figure 13: Cluster Distribution of 32 Prototype Vectors

Figure 14: Average Distortions of 32 Prototype Vectors
Figure 15: Cluster Distribution of 64 Prototype Vectors

Figure 16: Average Distortions of 64 Prototype Vectors
Figure 17: Cluster Distribution of 128 Prototype Vectors

Figure 18: Average Distortions of 128 Prototype Vectors
Figure 19: Cluster Distribution of 256 Prototype Vectors

Figure 20: Average Distortions of 256 Prototype Vectors
Figure 21: Cluster Distribution of 512 Prototype Vectors

Figure 22: Average Distortions of 512 Prototype Vectors
At the first glance, it seems that the charts for all 5 codebook sizes are similar in their shape as we consider Figure 13, Figure 15, Figure 17, Figure 19, and Figure 21. However, the maximum number of vectors represented by a size=32 codebook (700) is far higher than that of a size=128 codebook (210). And if we consider the codebook size 512, the maximum number of vectors represented by each prototype drops to 68. This proves that as we double the size of a codebook, a prototype is more equally shared by vectors. Moreover, if we consider the maximum average distortion, it is obvious from Figure 14, Figure 16, Figure 18, Figure 20, Figure 22 that the smallest codebook size (i.e. 32) yields the largest distortion. Figure 23 plots the total average distortion against each codebook size:

![Figure 23: Total Average Distortion of Different Codebook Sizes](image)

In Figure 23, we see a trend towards smaller distortion when we increase the codebook size. In fact, the slope of the line segment between size=32 and size=128 is slightly steeper than the slope of the line segment between size=128 and size=512. Thus, up to this point, it is still justified to use the largest codebook size.
We then carried out HMM recognition experiments using various codebook sizes. We trained six HMM's with 50 training sequences from each class, and then we scored each training sequence against all six HMM. We obtained the following recognition rate:

![HMM Recognition Rate for Different Codebook Sizes](image)

Figure 24: HMM Recognition Rate for Different Codebook Sizes

From Figure 24, we find the codebook sizes of 32 and 64 unacceptable since they yield the recognition rates of 70.3% and 79.7% respectively. The recognition rates for the sizes 128, 256 and 512 are very close although that of 512 is 97.3%, and that of 128 is 90.3%.

Since our experiments compared the HMM with our integrated model which consisted of neural networks, we did not want a codebook size which was inadequate for the HMM, nor was it too large for our neural networks which might take too much computational resources. Therefore, we finally chose the codebook size 128.
6.3 Training

50 observation sequences (vector quantized) from each phoneme were taken to train a HMM using the Baum-Welch formula (12) (13). The HMM were all 3-state, left-to-right models. After the training terminated, we obtained six HMM's. Each training sequence was then passed back to all HMM's for probability scoring and Viterbi segmentation. This procedure has already been described in 5.3. A neural network was then trained with the Viterbi segmentation output with error back-propagation algorithm. The termination criteria for the training of back-propagation neural network was fixed at a mean squared error of 0.01 per output unit. Table 1-6 show the transition probability matrices of the corresponding HMM's of the six phoneme. Figure 25-30 show the HMM observation distributions of the six phoneme. From Table 1-6 it can be observed that we have constrained the probability to the minimum of 0.000001 to avoid the zero-lising effect (See Appendix C.2). The initial probability at state 1 ($\pi_1$) was set to be 1. State 3 is in general a state which absorbs all the variations at the tail segment of a speech sample, thus it is always closest to 1. From the observation distribution charts in Figure 25-30, we can also observe that in state 2 there are usually fewer variations which is due to the psuedo stationary periods we have mentioned in 6.1. State 3 is generally subject to more variations. It is especially obvious in Figure 16 for the phoneme "ae".

This section ends the description of the data preparation in this chapter. We have described the procedures to obtain raw audio signals and the feature vectors, how the codebook size was chosen, and how the HMM's were prepared. Chapter 7 will introduce each experiment as well as how the training data for the neural network were obtained.
<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>$j =$ State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i =$ State 1</td>
<td>0.621361</td>
<td>0.165908</td>
<td>0.212732</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.893795</td>
<td>0.106204</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 1: HMM Transition Matrix of "aa"

<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>$j =$ State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i =$ State 1</td>
<td>0.859712</td>
<td>0.033783</td>
<td>0.106505</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.931987</td>
<td>0.068012</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 2: HMM Transition Matrix of "ae"

<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>$j =$ State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i =$ State 1</td>
<td>0.494062</td>
<td>0.245884</td>
<td>0.260055</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.855511</td>
<td>0.144488</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 3: HMM Transition Matrix of "ah"
<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>j = State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = State 1</td>
<td>0.473456</td>
<td>0.231282</td>
<td>0.295263</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.899701</td>
<td>0.100298</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 4: HMM Transition Matrix of "er"

<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>j = State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = State 1</td>
<td>0.463756</td>
<td>0.201245</td>
<td>0.334999</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.842297</td>
<td>0.157702</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 5: HMM Transition Matrix of "ih"

<table>
<thead>
<tr>
<th>$a_{ij}$</th>
<th>j = State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = State 1</td>
<td>0.384687</td>
<td>0.337673</td>
<td>0.277640</td>
</tr>
<tr>
<td>State 2</td>
<td>0.000001</td>
<td>0.847033</td>
<td>0.152966</td>
</tr>
<tr>
<td>State 3</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.999998</td>
</tr>
</tbody>
</table>

Table 6: HMM Transition Matrix of "iy"
Figure 25: HMM Observation Distribution of "aa" in 3 States
Figure 26: HMM Observation Distribution of "ae" in 3 States
Figure 27: HMM Observation Distribution of "ah" in 3 States
Figure 28: HMM Observation Distribution of "er" in 3 States
Figure 29: HMM Observation Distribution of "ih" in 3 States
Figure 30: HMM Observation Distribution of "iy" in 3 States
7. Experiments and Results

7.1 Experiments

Five recognition experiments were carried out. The procedures of performing these experiments were exactly the same as described in 5.3. All experiments used the same set of HMM's which was obtained from training 50 speech samples of respective classes using Baum-Welch algorithm. The neural networks for Experiment I to Experiment IV resulted from training the HMM segmentation parameters of 300 speech samples selected by $\phi_{train}$. For Experiment V, we changed the ratio of training-testing speech samples, and 450 speech samples were selected for training. The parameters used in all five experiments can be summarized as the number of hidden units in the neural network (h), the mean squared error per output unit in the neural network during training (err), whether the experiment used same set of data for training and testing (train), or training and testing data differed completely (test), and the training data selection criteria $\phi_{train}$. In short, the five experiments can be described below:

- Experiment I $\phi_{train}=3$, h=20, err=0.01, train
- Experiment II $\phi_{train}=3$, h=20, err=0.01, test
- Experiment III $\phi_{train}=20$, h=20, err=0.01, test
- Experiment IV $\phi_{train}=40$, h=20, err=0.01, test
- Experiment V $\phi_{train}=40$, h=20, err=0.01, test

7.2 Experiment I

In Experiment I, we passed 300 training speech samples through all HMM's and selected the resulting segmentation parameters by $\phi_{train}=3$ for neural network training. As a result, a total of 329 HMM B patterns (segmentation parameters) were collected. They were presented to a neural network with 384 input units (128 $\times$ 3), one hidden layer with 20 hidden units, and 6 output units.
Error back-propagation algorithm was used and the training terminated when the mean square error per output unit became 0.01. Table 7 shows the result of Experiment I:

<table>
<thead>
<tr>
<th>$\phi_{\text{recog}}$</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90</td>
<td>90</td>
<td>80</td>
<td>96</td>
<td>94</td>
<td>92</td>
<td>90.3</td>
</tr>
<tr>
<td>0.5</td>
<td>90</td>
<td>90</td>
<td>80</td>
<td>96</td>
<td>94</td>
<td>92</td>
<td>90.3</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
<td>90</td>
<td>80</td>
<td>96</td>
<td>92</td>
<td>92</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>94</td>
<td>88</td>
<td>86</td>
<td>96</td>
<td>94</td>
<td>92</td>
<td>91.7</td>
</tr>
<tr>
<td>6</td>
<td>92</td>
<td>88</td>
<td>86</td>
<td>96</td>
<td>94</td>
<td>92</td>
<td>91.7</td>
</tr>
</tbody>
</table>

Table 7: Result of Experiment I

Note that for an order of 0 ($\phi_{\text{recog}}=0$), the neural network layer is not used according to the rules described in 5.3. Thus, the first row of the results give us a reference to compare with the traditional HMM approach. It is obvious that even when we carried out testing using the training data, the average recognition rate of the pure-HMM approach was of 90.3%. This implies 9.7% of the training data were confused with other HMM's. The reasons have been discussed in Chapter 3. This situation can be described more in-depth using confusion matrix in Table 8 and Table 9.
<table>
<thead>
<tr>
<th></th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aa</strong></td>
<td>90</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>ae</strong></td>
<td>2</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td><strong>ah</strong></td>
<td>12</td>
<td>2</td>
<td>80</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>er</strong></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>96</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><strong>ih</strong></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td>4</td>
</tr>
<tr>
<td><strong>iy</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>92</td>
</tr>
</tbody>
</table>

Table 8: Confusion Matrix of Experiment I with $\phi_{\text{recog}}=0$

<table>
<thead>
<tr>
<th></th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>aa</strong></td>
<td>92</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>ae</strong></td>
<td>4</td>
<td>88</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td><strong>ah</strong></td>
<td>8</td>
<td>2</td>
<td>86</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>er</strong></td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>96</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><strong>ih</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td>6</td>
</tr>
<tr>
<td><strong>iy</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>94</td>
</tr>
</tbody>
</table>

Table 9: Confusion Matrix of Experiment I with $\phi_{\text{recog}}=6$

In Table 8, when $\phi_{\text{recog}}=0$, we can observe some serious confusion such as some "aa" is often recognized as "ae" and "ah", and "ih" is often confused with "iy". In Table 9, after we applied the neural network layer which was trained with the training samples selected by $\phi_{\text{train}}=3$, the situation slightly improved for the closeness measure of $\phi_{\text{recog}}=6$. Some "aa" began to be
recognized by the neural network as "aa" while the traditional HMM rejected them. And the greatest improvement in recognizing "ah" was also observed.

Experiment I gives us an index to the limitations of the traditional HMM approach. It also confirms us that the HMM parameters are trainable and the phoneme classes are separable. However, we do not know its generalization ability from this experiment. Thus, we need to consider Experiment II.

7.3 Experiment II

Experiment II used a completely different set of data for testing. We used 50 testing speech samples from each phoneme. These samples were passed through the HMM's for segmentation parameters. They were selected by various $\phi_{\text{recog}}$ values and presented to the neural network. The neural network was the same as the one used in Experiment I.

The following table shows the results of Experiments II.

<table>
<thead>
<tr>
<th>$\phi_{\text{recog}}$</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
<td>42</td>
<td>44</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53</td>
</tr>
<tr>
<td>0.5</td>
<td>68</td>
<td>42</td>
<td>46</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53.3</td>
</tr>
<tr>
<td>1</td>
<td>68</td>
<td>42</td>
<td>46</td>
<td>58</td>
<td>36</td>
<td>66</td>
<td>52.6</td>
</tr>
<tr>
<td>3</td>
<td>68</td>
<td>42</td>
<td>50</td>
<td>54</td>
<td>34</td>
<td>66</td>
<td>52.3</td>
</tr>
<tr>
<td>6</td>
<td>68</td>
<td>42</td>
<td>56</td>
<td>58</td>
<td>34</td>
<td>64</td>
<td>53.7</td>
</tr>
</tbody>
</table>

Table 10: Result of Experiment II with $\phi_{\text{train}}=3$
In this experiment, when \( \phi_{\text{recog}} \) increased from 0 to 6, we observed a fluctuation in the recognition rate. Some phonemes were recognized correctly by the HMM but wrongly by the neural network, and vice versa. This phenomenon can be observed in the following confusion matrices.

<table>
<thead>
<tr>
<th>Confusion Matrix of Experiment II (recognition rate %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_{\text{recog}} = 0 )</td>
</tr>
<tr>
<td>aa</td>
</tr>
<tr>
<td>ae</td>
</tr>
<tr>
<td>ah</td>
</tr>
<tr>
<td>er</td>
</tr>
<tr>
<td>ih</td>
</tr>
<tr>
<td>iy</td>
</tr>
</tbody>
</table>

Table 11: Confusion Matrix of Experiment II with \( \phi_{\text{recog}} = 0 \)

<table>
<thead>
<tr>
<th>Confusion Matrix of Experiment II (recognition rate %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_{\text{recog}} = 6 )</td>
</tr>
<tr>
<td>aa</td>
</tr>
<tr>
<td>ae</td>
</tr>
<tr>
<td>ah</td>
</tr>
<tr>
<td>er</td>
</tr>
<tr>
<td>ih</td>
</tr>
<tr>
<td>iy</td>
</tr>
</tbody>
</table>

Table 12: Confusion Matrix of Experiment II with \( \phi_{\text{recog}} = 6 \)

Comparing Table 11 and Table 12, we can observe that while there was an improvement in the recognition rate of "ah", some "er", "ih" and "iy" were confused. For "er", it seems that it was
easily confused with "aa", "ae" and "ah". We then considered the confusing patterns selected by $\phi_{\text{train}}=3$. We found that among the 329 training patterns for the neural network, there were 8 confusing patterns from "aa", 2 from "ae", 11 from "ah", 2 from "er", 4 from "ih" and 2 from "iy". We suspected that the confusing patterns among the training samples were not adequate, especially for "er", "ih" and "iy". Thus, in Experiment III, we increased $\phi_{\text{train}}$ to 20.

7.4 Experiment III

In Experiment III, a neural network with the same topology and training criteria was used. The only difference with Experiment II was the set of training patterns. In this experiment, the training patterns were selected with $\phi_{\text{train}}=20$, and we obtained 513 training patterns. Among the training patterns, 33 confusing patterns were from "aa", 18 from "ae", 57 from "ah", 26 from "er", 47 from "ih" and 32 from "iy". Table 13 shows the result of Experiment III:

<table>
<thead>
<tr>
<th>$\phi_{\text{recog}}$</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
<td>42</td>
<td>44</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53</td>
</tr>
<tr>
<td>0.5</td>
<td>68</td>
<td>42</td>
<td>48</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53.7</td>
</tr>
<tr>
<td>1</td>
<td>70</td>
<td>42</td>
<td>48</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>72</td>
<td>40</td>
<td>50</td>
<td>62</td>
<td>32</td>
<td>68</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>40</td>
<td>56</td>
<td>62</td>
<td>28</td>
<td>66</td>
<td>53.7</td>
</tr>
</tbody>
</table>

Table 13: Result of Experiment III

For $\phi_{\text{train}}=20$, the overall recognition rate improved. The situation for mis-recognizing "er" improved especially. However, the situation for "ih" was worse. Table 14 shows a confusion matrix for $\phi_{\text{recog}}=6$:  

57
We observed that since "ih" is often confused with "iy", the decision surface constructed by the neural network based on the extra confusion patterns may not be general enough to catch all "ih" instances. This may account for the decrease in recognition rate of "ih". On the other hand, it seems that the more confusion patterns we presented to the neural network, the better its performance. Thus, we carried out Experiment IV with $\phi_{\text{train}}=40$.

### 7.5 Experiment IV

In Experiment IV we used the same neural network as Experiment III except that we trained it with training samples selected by $\phi_{\text{train}}=40$. As a result, the confusion patterns increased. A total of 771 training patterns were gathered. Among them, 68 confusing patterns were from "aa", 47 from "ae", 116 from "ah", 66 from "er", 109 from "ih" and 65 from "iy". Table 15 shows the result:

<table>
<thead>
<tr>
<th>Confusion Matrix of Experiment III (recognition rate %)</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_{\text{recog}}=6$</td>
<td>70</td>
<td>8</td>
<td>20</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>aa</td>
<td>22</td>
<td>40</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>ae</td>
<td>16</td>
<td>20</td>
<td>56</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ah</td>
<td>12</td>
<td>18</td>
<td>4</td>
<td>62</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>er</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>28</td>
<td>42</td>
<td>66</td>
</tr>
<tr>
<td>ih</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>28</td>
<td>66</td>
<td>66</td>
</tr>
</tbody>
</table>

Table 14: Confusion Matrix of Experiment III with $\phi_{\text{recog}}=6$
Compare with Table 13, the overall recognition rate improved. The problem of mis-recognition was less serious. The confusion matrix for $\phi_{\text{recog}}=6$ is shown in Table 16:

<table>
<thead>
<tr>
<th>$\phi_{\text{recog}}$</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
<th>Averag.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
<td>42</td>
<td>44</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53</td>
</tr>
<tr>
<td>0.5</td>
<td>68</td>
<td>42</td>
<td>48</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>53.7</td>
</tr>
<tr>
<td>1</td>
<td>70</td>
<td>42</td>
<td>48</td>
<td>62</td>
<td>36</td>
<td>66</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>72</td>
<td>42</td>
<td>48</td>
<td>62</td>
<td>34</td>
<td>64</td>
<td>53.7</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>42</td>
<td>54</td>
<td>66</td>
<td>32</td>
<td>62</td>
<td>54.3</td>
</tr>
</tbody>
</table>

Table 15: Result of Experiment IV

From Table 16, while the overall recognition rate increased, "ih" and "iy" still confused with each other. We suspected that these two classes were close even in view of HMM segmentation parameters. This can be deduced from Figure 19 and Figure 20 where the HMM observation
distributions were plotted in each state for these two classes. Their observation distributions were especially similar in state 1 and state 2. Thus, the neural network might not be able to give a good generalization for these classes.

We believe that more data may help the neural network to build a better decision surface. To investigate on this with our limited set of training data and resources, we decided to perform Experiment V with different training/testing data ratio.

### 7.6 Experiment V

Experiment V reused the set of speech samples in Experiment I-IV. However, in this experiment the number of training speech samples was 450 (75 x 6), and the number of testing speech samples was 150 (25 x 6). The HMM’s were identical to those of Experiment I-IV. The topology of neural network and training criteria remained unchanged. We selected the training patterns for the neural network with $\phi_{train}=40$. We obtained the following result:

<table>
<thead>
<tr>
<th>$\phi_{recog}$</th>
<th>aa</th>
<th>ae</th>
<th>ah</th>
<th>er</th>
<th>ih</th>
<th>iy</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>76</td>
<td>40</td>
<td>52</td>
<td>60</td>
<td>28</td>
<td>68</td>
<td>54</td>
</tr>
<tr>
<td>0.5</td>
<td>76</td>
<td>40</td>
<td>56</td>
<td>60</td>
<td>28</td>
<td>68</td>
<td>54.7</td>
</tr>
<tr>
<td>1</td>
<td>76</td>
<td>40</td>
<td>56</td>
<td>64</td>
<td>28</td>
<td>68</td>
<td>55.3</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>40</td>
<td>60</td>
<td>64</td>
<td>28</td>
<td>68</td>
<td>56.7</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td>44</td>
<td>60</td>
<td>72</td>
<td>32</td>
<td>68</td>
<td>58.7</td>
</tr>
</tbody>
</table>

Table 17: Result of Experiment V
Comparing Table 15 and Table 17 where the only difference in both experiments was the training/testing data ratio, we found that when we provided more training data, there was obvious improvement in the recognition rate. In Table 15 when the training/testing data ratio was 300/300, there was a maximum increase of 1.3% in recognition rate (54.3%) over traditional HMM (53%). Table 17 shows that when we provided more training data such that the training/testing data ratio became 450/150, we obtained an even more stable recognition performance as well as a maximum increase of 4.7% in recognition rate (58.7%) over traditional HMM (54%). The problem of the confusion between "ih" and "iy" was not observed in this experiment. The confusion matrices are shown in Table 18 and Table 19:

<table>
<thead>
<tr>
<th>Confusion Matrix of Experiment V (recognition rate %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_{recog}=0$</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>aa</td>
</tr>
<tr>
<td>ac</td>
</tr>
<tr>
<td>ah</td>
</tr>
<tr>
<td>er</td>
</tr>
<tr>
<td>ih</td>
</tr>
<tr>
<td>iy</td>
</tr>
</tbody>
</table>

Table 18: Confusion Matrix of Experiment V with $\phi_{recog}=0$
In Table 19, it is obvious that the confusion between "ae" and "er" was solved better by the neural network. Consider and compare row 2 and row 4 in both matrices. There was a 4% increase in recognizing "ae" due to solving the confusion with "er", and a 12% increase in recognizing "er" due to solving the confusion with "aa", "ae" and "ah".

Thus, instead of relying only on the HMM's, we can use the neural network as an arbiter to improve the recognition rate in case there are more than one high-score given by the HMM's. The problem of discrimination is now partially solved.

### 7.7 Computational Issues

During training, for \( N \) training sequences and \( V \) HMM's, there can be at most \( NV \) training sequences for the neural network given a large enough \( \phi_{train} \) (which is often unnecessary). The neural network training accounts for a large portion of the additional computation and storage.

For any observation sequence, in worse case there can be \( V \) presentations to the neural network given a large enough \( \phi_{recog} \) as the order of closeness. For a single processor neural network,
emulator, it means in worse case it costs $V$ neural network forward calculation time. For parallel implementation, it may imply $V$ times the storage and one forward calculation time.

### 7.8 Limitations

One major limitation of this integrated model comes from the compression of segmentation data within the same state in the HMM. Two phoneme can be confused by this model if they exhibit different observation sub-sequences within one state but with similar distributions. For example, suppose there are 2 classes which can be represented by 3-state HMM's:

**Class A:** $v_1, v_2, v_3, v_4, v_5$

**Class B:** $v_1, v_2, v_3, v_4, v_5$

In this case, class A and class B will be confused since the order of observation is not preserved within the same state in our model.

Another limitation is that this approach is more suitable for a small and fixed-size class of speech such as phoneme. If the class size is large and grows, the neural network needs to be retrained and this process may take in worse case $NV$ times of presentation to the neural network during one training cycle as described in the above section. This becomes a problem if the class size is large and grows deliberately such as isolated words.
8. Conclusion

Speech recognition is a difficult problem in many aspects. Firstly, we have lower-level problems such as end-point detection of speech and noise removal. Secondly, we have higher level problems such as solving the variations arise from different speakers, and lastly when the problem sublimes to the level of human language, it is an intelligence problem where numerous ambiguity needs to be resolved based on context, semantics and pragmatics.

In the history of speech recognition, different approaches have been attempted. In recent years there seems a trend towards using stochastic models and neural networks. Despite which method those research were in favor of, all the approaches assumed a hierarchy of speech. That is, for a continuous speech, all the approaches first model subword units, then words, then some sort of simplified grammar. Thus, inevitably different levels of problems are encountered by all the approaches.

One major advantage an approach has over another is the way it acquires and models the knowledge. Knowledge-based approach assumes some domain experts to provide accurate and concise knowledge. This approach will be beneficial when one day all the internal structures of speech are discovered. Neural networkers are working on this goal. On the other hand, stochastic approach such as HMM offers a clever solution to absorb multiple variable-length sequences to a finite number of states which is more resource-efficient than other approaches. Nevertheless, HMM suffers from the problem of lacking discrimination.

As a result, we came up with the idea of integrating HMM and neural networks. We have performed various experiments on the TIMIT speech database. Our experiments showed that a reasonable codebook size for the LPC cepstrum parameters of TIMIT should be at least 128 in order to get a more accurate recognition and smaller distortion. Using this codebook, we have also
performed experiments with the pure HMM approach. Our experiments showed that for a chosen set of six phonemes, a 300-sample training set was not completely recognized by the HMM's and the recognition was only 90.3% using a 3-state HMM. This is an evidence of the discrimination problem of the HMM.

In our experiments we found that the observation probability distribution matrices in the HMM can be a source of knowledge in its own right. Thus, we trained a MLP with error back-propagation algorithm on this set of data to extract the internal knowledge. The experiments showed that as we increase $\phi_{\text{train}}$ (a threshold value which defines how close two observation sequences should be in order to put them both in neural network training) to provide more training data for the MLP, the better the recognition rate over pure HMM approach. In our experiments with a training/recognition data ratio of 300/300, we achieved a 1.3% increase in recognition rate over pure HMM approach.

We believed that while more training data may not be an advantage for HMM, it definitely would be an advantage for neural networks. We therefore performed another experiments using a different training/recognition data ratio of 450/150. The experiment showed that there was only 1% increase in recognition rate for the pure HMM approach, while an increase of 4.4% in recognition rate for the integrated approach was observed.

We conclude that when a neural network is used as an arbiter for two sequences classified by HMM alone and both obtain high and close probability scores, the neural network often discriminates them better than the HMM. Moreover, the integrated approach we propose will perform more successfully when more training data are given.

Furthermore, our analysis on the model shows that our approach is more suitable to be applied on a small and fixed-sized speech class such as phoneme. The resource and time complexities may
grow considerably as the number of classes increase. The time complexity can be as large as \( NV \) for \( V \) classes \( N \) training sequences for each class.
9. Future Directions

HMM is an extensible approach. The same procedures for phoneme recognition can also be extended to word recognition and to connected word recognition. To recognize a word "bar" (ba:), in HMM we have two options. The first option is to separately train two HMM's which recognize "b" and "a:". We combine these two phoneme HMM's to form one word HMM by creating a transition from the final state of "b" to "a:". The second option may be more suitable if the number of words to be recognize is large. We simply put all the phoneme HMM's in parallel, joining the all starting states together and all final states together to form one complex HMM. Then, for an utterance we use Viterbi algorithm to segment the speech against this complex HMM to analyse the path that has been taken inside. Thus, we can decide which phoneme HMM's have been traversed such that a word is formed. One challenge that we face is the way to use our approach so that it can be as extensible as HMM, and the phoneme recognizer can be reused for word and connected-word recognition without requiring excessive resources.

Modelling the intra-state distribution pattern is another challenge. In our present approach, we simply represent an observation symbol by turning the corresponding position in the bit vector to one. This has been mentioned in Chapter 5 as the constraint for a fixed-length pattern of a variable-length utterence. However, this constraint may handicap classification especially when the intra-state patterns are crucial. For example, the following two classes of speech can be recognized as one under our approach for a 2-state HMM:

\[ C_1: \ v_1 \ v_1 \ v_1 \ldots v_2 \ v_2 \ v_4 \ v_4 \ v_4 \]
\[ C_2: \ v_1 \ v_1 \ v_1 \ldots v_4 \ v_4 \ v_4 \ v_2 \ v_2 \]

For a 2-state HMM which absorbs \( v_2 \) and \( v_4 \) in its second state, it is difficult to discriminate class \( C_1 \) from \( C_2 \) due to the inability of our model to extract the intra-state knowledge. Thus, another
future direction is to design a model or representation such that these intra-state parameters can be
more distinguishable with each other.
Appendix A.  Linear Predictive Coding

There is a high correlation between the adjacent samples of speech waveforms. Thus, for speech samples \( \{ y_n \} \) (in waveform), we may assume that:

\[
y_n \approx \alpha_1 y_{n-1} + \alpha_2 y_{n-2} + \ldots + \alpha_p y_{n-p}
\]  \hspace{1cm} (22)

where \( \alpha_i \) a LPC coefficient and \( (\alpha_1, \alpha_2, \ldots, \alpha_p) \) is a \((p-1)\)-order LPC coefficient vector.

To find out \( \alpha_i \), we need to solve the equation:

\[
\begin{bmatrix}
    r_0 & r_1 & r_2 & \ldots & r_{p-1} \\
    r_1 & r_0 & r_1 & \ldots & r_{p-2} \\
    r_2 & r_1 & r_0 & \ldots & r_{p-3} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    r_{p-1} & r_{p-2} & \ldots & r_0
\end{bmatrix}
\begin{bmatrix}
    \alpha_1 \\
    \alpha_2 \\
    \alpha_3 \\
    \vdots \\
    \alpha_p
\end{bmatrix}
= 
\begin{bmatrix}
    r_1 \\
    r_2 \\
    r_3 \\
    \vdots \\
    r_p
\end{bmatrix}
\]  \hspace{1cm} (23)

where

\[
r_j = \frac{1}{N} \sum_{n=0}^{N-j-1} y_n w_n y_{n+j} w_{n+j}
\]  \hspace{1cm} (24)

and \( w \) is a time window function such that \( y_n \) is assumed zero outside the window.
Appendix B  Implementation of a Vector Quantizer

Since we cannot train a HMM using an infinite set of observations, we need to sacrifice some preciseness in order to find a finite set which can well represent the signal sequences without losing accuracy. There are several algorithms [19] and we finally adopt the solution proposed by Linde-Buzo-Gray [20]. The algorithm is quite interesting and we implemented it using C language under a DEC 5200. Basically it consists of two stages. The first stage is codebook generation and the second stage is choosing the closest code from the codebook. The second stage is trivial after the first stage is described (in functional description):

Function:  Codebook Generation  

Input: Training sequence \{x_j; j=0, ..., n-1\}  

(In our case \(x_j\)'s are the 12-order LPC cepstrum coefficients.)  

\(N\): size of the codebook (must be in power of 2)  
\(\varepsilon\): a fixed perturbation vector  
\(d\): a distortion function  
\(x\): a function to find the centroid  

Algorithms:

Step (0)  
set \(M = 1\), set \(A = \) input training sequence  
find \(A(M) = x(A) = \) centroid of \(A\)

Step (1)  
given \(A(M) = \{y_i; i = 1, ..., M\}\)  
split each \(y_i \in A(M)\) to \(y_i + \varepsilon\) and \(y_i - \varepsilon\), and put them to \(A(2M)\)  
set \(M = 2M\)
Step (2)
set $m=0$, set $D_{-1} = \infty$

Step (3)
partition the $x_j$'s: $P(A(M)) = \{S_i : i = 1, \ldots, M\}$
s.t. $x_j \in S_i$ iff $d(x_j, y_i) \leq d(x_j, y_j) \forall j$
set $D_m = n^{-1} \sum_{j=0}^{n-1} \min(d(x_j, y), y \in A(M))$

Step (4)
If $(D_m - D_{m-1}) / D_m \leq \varepsilon$ then goto Step (6)

Step (5)
set $A(M) = \{x(S_i) : i = 1, \ldots, M\}$
set $m = m + 1$, goto Step (3)

Step (6)
If $M = N$, output $A(M)$
else goto Step (1)

Output: A codebook containing $N$ representative codes of $x_j$'s.

This algorithm behaves like an amoeba. Starting from one point in a cloud of data, the algorithm gradually divides the cloud into two clusters and then repeats the centroid-finding and divide for individual cluster. Linde-Buzo-Gray also specified the distortion function and the centroid-finding function [20].
Finding the distortion $d(x_j, u)$:

$$\sum_{j \in S} (x_j - u)R(x_j)(x_j - u)'$$

(25)

Finding the centroid of $S$:

$$x(S) = \left[ \sum_{j \in S} R(x_j) \right]^{-1} \sum_{j \in S} R(x_j)x_j'$$

(26)

$R(x_j)$ is an auto-correlation matrix that is a byproduct of LPC analysis. The distortion function measures the distance of two vectors.
Appendix C. Implementation of HMM

There are several issues which become a problem when we need to realize a HMM on any machine [21]. These issues are:

- calculations underflow
- zero-lizing effect
- training with multiple observations

C.1 Calculations Underflow

From equation (8) and (9), we see that as $T$ approaches infinity, $\alpha_T(i)$ and $\beta_T(i)$ both approach zero in exponential fashion. In practice, the number of observations necessary to train a speech model will surely result in underflow on many computer. Thus, straight-forward implementation of the HMM using equation (8) to (13) is not feasible. Fortunately, there exists a method which provide scaling of the entire computation to avoid the underflow. In general, the method uses a carefully chosen scaling factor $c_t$ to scaling down the computation at each recursive level in calculating $\alpha_t(i)$ and $\beta_t(i)$:

$$c_t = \left[ \sum_{i=1}^{N} \alpha_t(i) \right]^{-1}$$  \hspace{1cm} (27)

Note that $\sum_{i=1}^{N} c_t \alpha_t(i) = 1$. Similarly, when we compute $\beta_t(i)$ we replace $\beta_t(i)$ with $c_t \beta_t(i)$ at each recursive level. Thus, the original calculation for $a_{ij}$ in (5) becomes:
where $C_t = \prod_{t=1}^{T} c_t$ and $D_t = \prod_{t=1}^{T} c_t$. Similar technique can be applied to equation (6). One of the best
thing is that in equation (18) no matter which value $t$ takes, $C_t D_{t+1}$ can always be factored out.

After solving this problem, another question arises. Note that for equation (11) and (12) we cannot
apply the scaling factor and yet obtain the probability $P$. Why? First, we have mentioned that we
cannot directly calculate $a_f(i)$. Second, when we apply scaling factor $c_t$ we obtain

$$C_t \sum_{i=1}^{N} \alpha_t(i) = 1$$ (29)

Note that this is not $P$ but $P$ multiplied by $C_t$. Can we simply calculate this product and divide it
by $C_t$? Obviously not since $C_t$ can neither be evaluated directly (which increases exponentially).
Thus, we cannot obtain $P$.

One way to overcome this dead-end is to find $\log P$ instead of $P$. From (29), we see that

$$C_t = \prod_{t=1}^{T} c_t = \frac{1}{P}$$ (30)

and we can calculate $\log P$ without multiplying the $c_t$'s together. Taking log of both side, we get,
\[ \log P = - \sum_{t=1}^{T} \log c_t \]  

(31)

and the numerical problem is now solved completely.

C.2 Zero-lising Effect

An ideal HMM training requires the training observation sequence to be as long as possible, and cover as many combinations of the output generated by that HMM as possible. In reality, such as applying HMM to speech recognition, it is impossible.

What is wrong with short or small set of training sequence? The answer was not obvious until we implemented an HMM using C with double precision floating-point arithmetic on a DECstation. We trained it with a few random sequence and then prompt it with another unseen sequence to ask for the probability. At a number of times this probability value degenerates to zero because in the final model some parameters in the HMM were zero. This is due to the fact that, in order to pave the "best" path for all the observation sequence (and it assumes that is all), some parameters are just naturally zero.

This is inevitable due to insufficient training data and generates many zeroes in the parameter space. We solved this problem by introducing the constraints \( a_{ij} \geq 0 \) and \( b_{jk} \geq 0 \). We did not build these constraints into the Baum-Welch reestimation formulas. Rather, we implemented it as a post-processor of the adapted model from HMM training. In our implementation, we adopted the following schemes:

For \( a_{ij} \) 's,

1. Suppose \( a_{iN-1} \) to \( a_{iN} < \varepsilon \), set them to \( \varepsilon \).
2. Set

\[ a_{iN} \]
\[ a_{ik} = (1 - 1 \epsilon) a_{ik} \left[ \sum_{j=1}^{N-1} a_{ij} \right] \forall a_{ik} \geq \epsilon \]  
(32)

For \( b_{jk} \)'s,

1. suppose \( b_{jN-1+1} \) to \( a_{iN} < \epsilon \), set them to \( \epsilon \).
2. set

\[ b_{jk} = (1 - 1 \epsilon) b_{jk} \left[ \sum_{i=1}^{N-1} b_{i} \right]^{-1} \forall b_{jk} \geq \epsilon \]  
(33)

Levinson has shown that using (32) and (33) yields the model which is as optimum as the original reestimated model plus the new constraints [21].

C.3 Training With Multiple Observation Sequences

The original reestimation formulas (12), (13), are designed for training a HMM with a single observation sequence only. While in speech recognition, we need to train the HMM with lots of different pieces of speech data. According to Levinson, the solution is trivial - we may just train each sequence separately and finally summing the result together [21].
References


[17] Lamel, L.F., Kassel, R.H., Seneff, S. Speech Database Development: Design and Analysis of
the Acoustic-Phonetic Corpus. Proceedings of the DARPA Speech Recognition Workshop,

IEEE 63(4), April, 1975, pp. 662-667.

294-303.


[21] Levinson, S. E., Rabiner L. R., and Sondhi M. M. An Introduction to the Application of the
Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. BSTJ.


[23] Rabiner, L.R., and Juang, B.H. Recognition of Isolated Digits Using Hidden Markov Models
pp.1211-1234.


and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition. BSTJ, vol. 62,
no. 4, April 1983, pp. 1075-1105.