# Comparison of feature vectors for speech recognition using the time delay neural network

by

Brian Lee Schmidt

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical Engineering and Computer Engineering
Major: Electrical Engineering

Signatures have been redacted for privacy

Iowa State University
Ames, Iowa

1993

## DEDICATION

To my loving wife Kara, for all of her support and understanding throughout this entire endeavor.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# CHAPTER 1. INTRODUCTION

## Speech Recognition

### Significance of Speech Recognition

There has been a desire for some time to have an automatic typing machine, which will use voice input to produce typewritten output. Such a machine will relieve people from having to develop keyboarding skills. Obviously, it will be of particular benefit to disabled people who lack the physical ability to use a keyboard. This automatic typing machine has not yet been realized but there has been progress toward this ideal. There are currently software packages that will do limited recognition and insert words into a word processor. Another useful application of speech recognition is in controlling light switches and other appliances that respond to a selected word or phrase. Speech recognition can be beneficial to people who have to perform some task, while they are using their hands for another task. A lot of tasks may be automated with speech recognition. Currently there are dictation machines that recognize a fairly large vocabulary (20-30,000 words), that are being marketed toward surgeons. This dictation machine recognizes words and generates a typewritten report. However, there are several problems and issues that must be taken into account when developing a speech recognition system.

### Problems Encountered in Speech Recognition

Several problems are encountered when trying to build a speech recognition system. Some of the common problems are speaker variations, ambiguity, variations in an individual's pronunciation, and noise or interference. Speaker variations refer to the differences in speech produced by different people. Ambiguity arises, e.g., due to similar sounding words such as "to", "too", and "two". An individual does not pronounce a word in exactly the same way every time. It is common for one to speak a particular word in different amounts of time or with different emphasis on the syllables.

There are several issues that must be addressed when building a speech recognition system. These include endpoint detection, data reduction, segmentation, time alignment, normalization, and the selection of a recognition technique. If the speech signal is composed of more than one unit of speech, the beginning and ending of each unit must be detected. This process is referred to as endpoint detection. Data reduction is usually required for a speech system because of the large amounts of data involved with speech signals. If, for example, a speech signal consists of 2 minutes of speech sampled at 12 kHz there would be about a million and a half data points. Often words are processed in a speech system and words alone tend to last anywhere from half a second to two seconds. When processing words, there are too much data in a full word to allow reduction of the data in a reliable manner; therefore it is common to segment the word. Segmentation is a process that divides a word into smaller components. Time alignment is an important issue in speech recognition. If a person spoke a particular word several times it is very unlikely that he/she would speak the word in exactly the same amount of time or spend the same time on each phoneme (basic unit of speech that completely represents all of the sounds in a given language) or syllable. Therefore, if the segmentation process simply divides the word based on the amount of time there may be severe misalignment between different realizations of the same word. Normalization is used to reduce the difference in speech produced by different speakers.

In addition to these problems and issues there are different levels of complexity depending upon whether a system handles isolated or continuous speech, small or large vocabularies, whether it is speaker dependent or independent, and whether speech understanding is required. Isolated speech has intentional pauses between each word, which makes endpoint detection easier. Continuous speech, however, has no intentional pauses and is generated at a normal speaking rate. Small vocabularies are usually based on 10 to 100 words, while large vocabularies include 1000 or more words. Speech understanding systems not only recognize the words, but also know sentence structure and grammar.

## Common Speech Recognizers

There are at least three types of speech recognizers currently being used [3]. These recognizers fall under the categories of dynamic time warping (DTW), hidden Markov models (HMM), and artificial neural networks (ANN). In the following each of these types of recognizers is described along with some pros and cons. The discussion of the dynamic time warping and the hidden Markov model recognizers is more detailed since they will not be discussed in the remainder of this paper.

The dynamic time warping recognition technique involves creating reference templates for each recognition unit. The templates are compared to a time warped speech signal (test word); the template that is closest to the warped speech signal is chosen to be the recognized unit. The test word (or test template) is compared to the reference templates in a manner that gives the best matching score. The matching score is determined by minimizing time differences between the references and the test word. The technique used to minimize these time differences is referred to as dynamic programming. In processing the test templates and the reference templates the features are recorded in the following manner:

$$\text{test template}: \quad t(1), t(2), t(3), \ldots, t(i), \ldots, t(I)$$

$$\text{reference template}: \quad r(1), r(2), r(3), \ldots, r(j), \ldots, r(J)$$

The cost is denoted as: $0 \le d(i_k, j_k) =$ cost of matching $t(i_k)$ with $r(j_k)$, where d is some distance measure and k is the number of transition in the path traversed. The global cost function is simply the sum of all of the individual costs: $D = \sum_{k=1}^{K} d(i_k, j_k)$, K is the total length of traversal. The match is selected for the template that provides the smallest overall cost. Figure 1.1 gives a typical algorithm for isolated word recognition. Figure 1.2 gives the search space for the DTW algorithm. The search space is the area that the DTW algorithm checks to determine if a word matches a template. If a path extends beyond the search space it is assumed to be unusable.

Advantages of the DTW include: alleviation of time alignment problem and optimum word sequence search. Problems with the DTW recognition technique are many states, poor generalization for large vocabularies, not being usable for continuous speech recognition, being computationally expensive, and storage problems for the many templates.

Hidden Markov models are stochastic models of speech production. HMM is a system that is capable of being in only a finite number of states; each state of the HMM generates a finite number of possible outputs. To generate an output sequence the system moves through different states emitting an output at each state until the entire word is produced. An example of a state diagram for a Markov process is given in Figure 1.3. The circles represent states and the arrows represent the transitions between states. The transitions between states and the outputs associated with each state are random. Allowing the transitions and outputs to be random gives the model the ability to overcome variations in pronunciation and timing.

```
DTW Algorithm for Isolated Word Recognition

Step 1 Initialization
    $D_{\min}(1,j) = d(1,j), \qquad j = 1,\ldots,\varepsilon$
    $D_{\min}(i,1) = d(i,1), \qquad i = 1,\ldots,\varepsilon$
        $D_{\min}$ is the minimum cost for traversing the path
        $d$ is a distance measure
Step 2 Recursion
    For $i = 2,\ldots,I$
        For $j = J,\ldots,2$
            $D_{\min}(i,j) = \min_{(i_{k-p},j_{k-p})} \left\{ D_{\min}\left[(i_{k-p},j_{k-p})\right] + \hat{d}\left[(i_k,j_k)\middle|(i_{k-p},j_{k-p})\right] \right\}$
            where $\hat{d}\left[(i_k,j_k)\middle|(i_{k-p},j_{k-p})\right] \overset{\text{def}}{=} \sum_{m=0}^{p-1} d\left[(i_{k-m},j_{k-m})\middle|(i_{k-m-1},j_{k-m-1})\right]$,
            $d\left[(i_{k-m},j_{k-m})\middle|(i_{k-m-1},j_{k-m-1})\right]$ is the distance to move to
            $(i_{k-m},j_{k-m})$ given the previous distance $(i_{k-m-1},j_{k-m-1})$,
            and p is a legal previous node
        Next $j$
    Next $i$
Step 3 Termination
    Best path has cost $\tilde{D} = \min \begin{cases} D_{\min}(I,j)/I, & j = J-\varepsilon,\ldots,J \\ D_{\min}(i,J)/J, & i = I-\varepsilon,\ldots,I \end{cases}$
```

Figure 1.1 DTW Algorithm

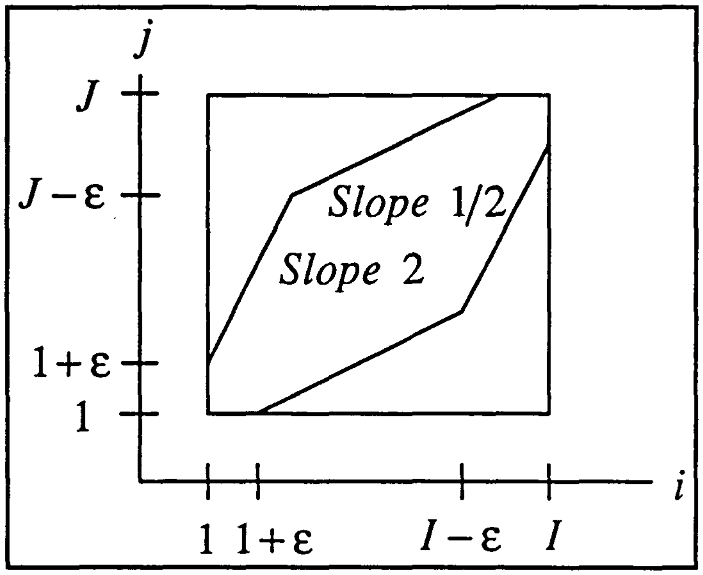


Figure 1.2 Search Space for the DTW Algorithm

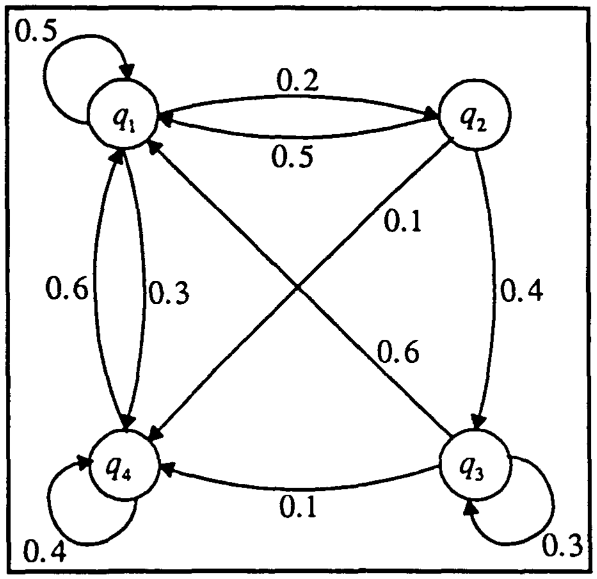


Figure 1.3 Markov State Diagram

Each word to be recognized is represented by a model similar to that given in Figure 1.3. The recognizer has only the outputs of the system to use in determining which model provided the output. It is assumed that each state-to-state transition occurs at discrete times and that the transition from state $q_i$ to $q_j$ has a probability that only depends on state $q_i$. The numbers seen in Figure 1.3 represent the transition probabilities between the states. The notation used to represent the transition probabilities for a system with N states is a NxN matrix denoted by A, where $a_{ij} = \mathrm{P}\{\text{transition from } q_i \text{ to } q_j\}$. The transition matrix for the state diagram in Figure 1.3 is:

$$A = \begin{bmatrix} 0.5 & 0.2 & 0.0 & 0.3 \\ 0.5 & 0.0 & 0.4 & 0.1 \\ 0.6 & 0.0 & 0.3 & 0.1 \\ 0.6 & 0.0 & 0.0 & 0.4 \end{bmatrix}$$

If there is a set of T possible outputs $\{z_i\}$, then the probabilities of the outputs corresponding to the different states are represented by a NxT matrix B. With each state $q_i$ there is an associated vector $\bar{b}_i$, of size T, where $b_{ij} = \mathrm{P}\{\text{output } = z_j | \text{state } = q_i\}$. The ith row vector of the output matrix B is $\bar{b}_i'$. The initial state probabilities are denoted by the vector $\bar{\mathrm{p}}(1)$, where $p_i(1) = \mathrm{P}\{\text{initial state is } q_i\}$. The entire system can be specified with the parameters N, p(1), A, and B. This model can be defined as $\mathrm{M} = \{\mathrm{N}, \bar{\mathrm{p}}(1), \mathrm{A}, \mathrm{B}\}$.

A library of models that represent the words in the vocabulary must be constructed to train the recognizer. This implies that for each word the number of states, the transition matrix A, the initial state probabilities p(1), and the output probabilities B must be determined. At recognition time the system observes the output $O(t) = (O_0, O_1, \ldots, O_{T-1})$. For each model $\mathrm{M}_i$ the probability $\mathrm{P}\{\mathrm{O}|\mathrm{M}_i\}$ is computed. The recognized word is determined to be the model word that yields the highest probability.

The recognition task is composed of estimating the probability $\mathrm{P}\{\mathrm{O}|\mathrm{M}_i\}$. A simple recursive algorithm can be used for this purpose and will be discussed here. There is a more efficient method called the Viterbi algorithm, which determines the optimal path [11]. The recursive algorithm presented here is called the forward method. To start with, assume that at some time t the model has reached a state $q_i$, having emitted the partial output sequence $O(t) = (O_0, O_1, \ldots, O_t)$, where each $O_i$ is some output symbol $z_k$. Assume that the probability of arriving at $q_i$ at time t with an observation vector $O(t)$ is known, and let this probability be

$\alpha_t(i) = \mathrm{P}\{O(t)$ and $q_i$ at t$\}$. Next we want to determine the probability at the next time interval: $\alpha_{t+1}(j)$. If the only transition considered are those from $q_i$, then the new probability would be $\alpha_t(i)a_{ij}b_{jk}$. Since more than one path leads to the state $q_i$, all of these paths must be considered. Therefore:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i)a_{ij}b_{jk}$$

The initial step for this recursion is $\alpha_1(j) = p_j(1)b_{jk}$ for all j. The probability that the word was produced by the model $M_i$ is:

$$\mathrm{P}\{\mathrm{O}|\mathrm{M}_i\} = \sum_{j=1}^{N} \alpha_{T-1}(j)$$

There are several key advantages of the HMM over the DTW. These advantages are: quick recognition, fewer states, simple algorithm (this gives a straight forward implementation), and cheap computation. The disadvantages of the HMM are: long training times, underflows when making calculations, and estimation of large number of parameters.

Artificial neural networks have typically been applied to pattern recognition type tasks. Neural networks are fairly new to speech recognition, but have been shown to be effective for the task. A neural network consists of several layers of interconnected nodes whose outputs are calculated by summing inputs multiplied by suitable weights and passing the sum through a nonlinear activation function. The neural network has been successfully applied to many complex problems. The advantages of neural networks are complex decision regions, quick recognition, possible VLSI implementation, and efficient training algorithms. The disadvantages of neural networks are long training times and no guarantee of convergence to a global minimum. More detailed discussion of ANN will be presented in Chapter 3.

## Objective/Scope

The purpose of this research is to compare several different feature vectors in terms of their accuracy of speech recognition when used in a time delay neural network. The time delay neural network (TDNN) was developed by Alex Waibel and will be discussed in Chapter 3. A feature vector is a small set of values that represent the speech signal in some manner; the representation of the signal in this manner is important since it reduces the amount of information that the neural network is required to learn. All the features considered in this work involve spectral components of the speech signal. These features are mel-scaled FFT (Fast Fourier Transform) coefficients, linear filter bank FFT coefficients, linear prediction coefficients, mel-scaled cepstral coefficients, linear filter bank cepstral coefficients, and wavelet transform coefficients. The mel-scale is a nonlinear scale which emphasizes low frequencies. The mel-scaled FFT generates coefficients by summing the spectrum in frequency ranges described by the mel-scale. The linear filter bank FFT simply divides the frequency range linearly and sums the spectrum values in those ranges. Linear prediction estimates the envelope of the spectrum by predicting future output based on past outputs using linear methods. The cepstrum is the inverse Fourier transform of the log of the FFT, which separates the excitation spectrum and shaping filter spectrum of the speech signal. Both the mel-scale filter and the linear filter bank are applied to the cepstrum to generate two different sets of coefficients. The mel-scaled cepstrum is currently the most commonly used feature vector for speech recognition. The last method of generating feature vectors is through the wavelet transform. Wavelet transforms adjust the scaling of the output depending on whether the signal is of high or low frequency. For low frequencies the wavelet transform provides more information about the frequency content while at high frequencies more information about when the frequency occurred is obtained. The fourth order Daubechies wavelet transform is used in this work. The goal is to determine one or two feature vectors that work well for speech recognition using the time delay neural network. The determination of the "goodness" will be based on comparing the effectiveness of the feature vectors in terms of speech recognition accuracy.

The speech signals used in this work were speaker dependent isolated words. The words chosen for this purpose were /b/, /d/, /e/, and /v/. These words were selected for their confusability in the /e/ family, which makes the recognition task more difficult. Having a more difficult recognition task was helpful in discriminating the effectiveness of the different feature vectors. The entire recognition system was made up of the following components: end point detection, segmentation of each word (blocking), generation of the feature vectors, compression of the feature vectors, normalization of the feature vectors, and recognition using the time delay neural network. Figure 1.4 shows the block diagram of the system used in this work.

The first step in building this recognition system was to collect speech samples. The speech samples were obtained using a sampling frequency of 12 kHz. Groups of around 12 words were collected at a time. Endpoint detection is required for the speech signals, since there are several words in each signal. For this work the endpoint detection was accomplished using a comparison of the variance of the silent sections and word sections.

The next step in the recognition system was segmentation of the word. Once the word has been extracted from the speech signal it was broken into 5 ms blocks (60 samples). These small blocks of speech samples were used to generate the individual feature vectors. Six different feature vectors were used as discussed above. Thirty sets (150 ms) of feature vectors were generated, which were then compressed to 15. The compression was accomplished by averaging adjacent blocks, which gave a broader base for the time delay neural network. Normalization was performed on the features to give coefficients that ranged between 0 and 1. The normalization provides a more dense input vector for the neural network, and removes variations due to loudness of the spoken words. Finally the time delay neural network uses 15 sets of feature vectors and classifies the input as one of the four possible outputs (/b/, /d/, /e/, /v/). The built-in delays in the TDNN allow the neural network to learn the speech signals without accurate time alignment. Not requiring precise time alignment allows for simple implementation of the endpoint detector.

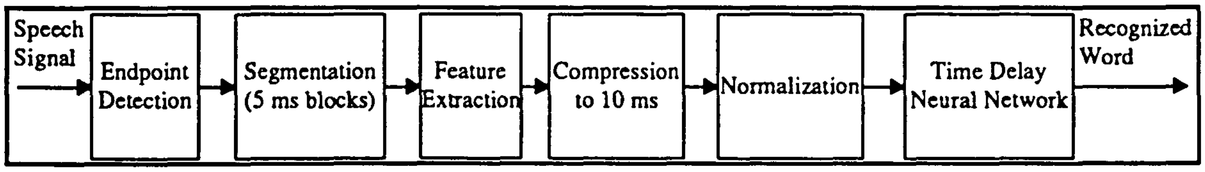


Figure 1.4 Speech Recognition System

## Organization of the Remaining Document

There are four chapters that follow this introductory chapter. Chapter 2 provides an introduction to speech production, speech processing, speech recognition, and a discussion of the techniques used in this work. Chapter 3 presents artificial neural networks, a discussion of neural networks that led to the development of the time delay neural network, and a description of the implementation of the time delay neural network. The results obtained in this work and the significance of those results are discussed in Chapter 4. The final chapter contains the concluding remarks and some suggestions for future research. Following these main chapters is an appendix that includes all of the programs used in the implementation of the time delay neural network.

# CHAPTER 2. SPEECH PROCESSING AND RECOGNITION

This chapter introduces the concepts of speech production, processing, and recognition. A discussion of the speech recognition system developed for this project is also included along with the different methods used to generate the feature vectors.

## Speech Production

The vocal system can be divided into three subsystems - the lungs and trachea, larynx, and vocal tract. The lungs and trachea act as the power source for the system, the larynx is the main sound generator, and the vocal tract modulates the sound from the larynx. There are two main functions associated with speech generation - excitation and modulation. Excitation is the generation of a pulse train and modulation imposes information onto the pulse train, see Figures 2.1, 2.2, 2.3.

A simplified diagram of the vocal tract is given in Figure 2.1 (e(n) is the excitation, θ (n) is the modulation, and s(n) is the output speech signal) or, from a system viewpoint, Figure 2.2. The two types of excitations used to generate phonemes are voiced and unvoiced. Voiced excitation is a periodic stream, while unvoiced excitation is turbulent. Acoustically the vocal tract is a tube of nonuniform cross section, which has many natural frequencies. Because of the nonuniformity the vocal tract resonances are not equally spaced; however, there is approximately 1 resonant frequency per kHz bandwidth. These resonance frequencies (formants) have different locations depending on the phoneme being spoken. Formants are the primary source of information for the generated speech. Formants are labeled by number in order of increasing frequency: $F_1$, $F_2$, etc. The formants for the phoneme /e/ are shown in Figure 2.4. $F_0$ is the pitch; pitch is the principle frequency generated. There is a large amount of information contained in the frequency of the signal. Figure 2.5 shows both a time and frequency plot of a speech signal (note that the information contained in the frequency signal is more readily usable then the time signal).

## Speech Recognition

### Complexity Issues

There are several levels of complexity involved in speech recognition systems. One distinguishing level of complexity is whether the system is using isolated or continuous words. Isolated words are words that have intentional pauses between each word (usually pauses are at least 250 ms), while continuous speech have no intentional pauses. Another possible

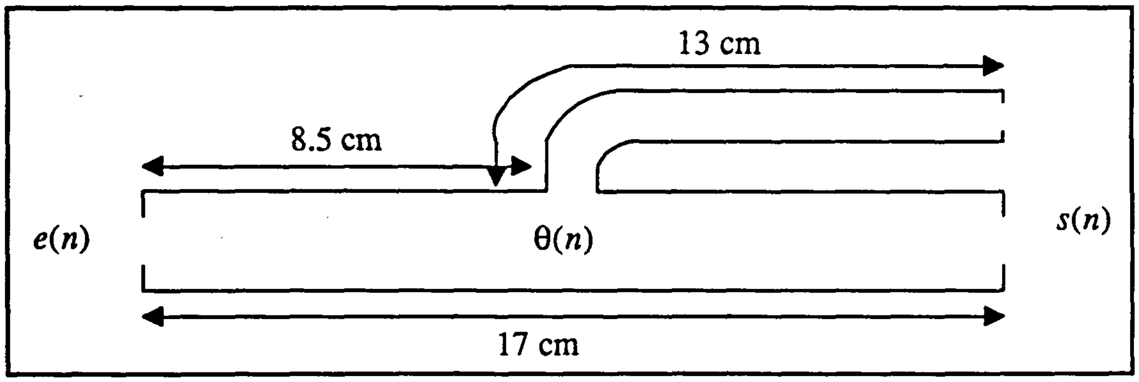


Figure 2.1 Simplified Vocal Tract Model

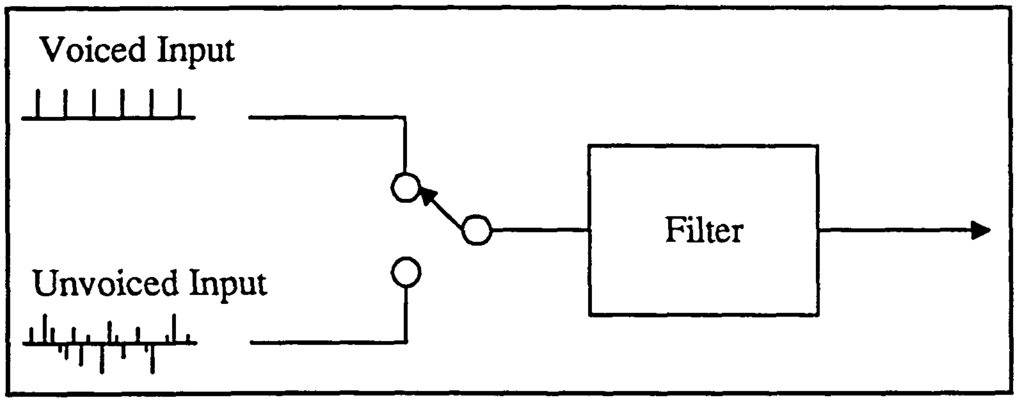


Figure 2.2 Vocal Tract System Model

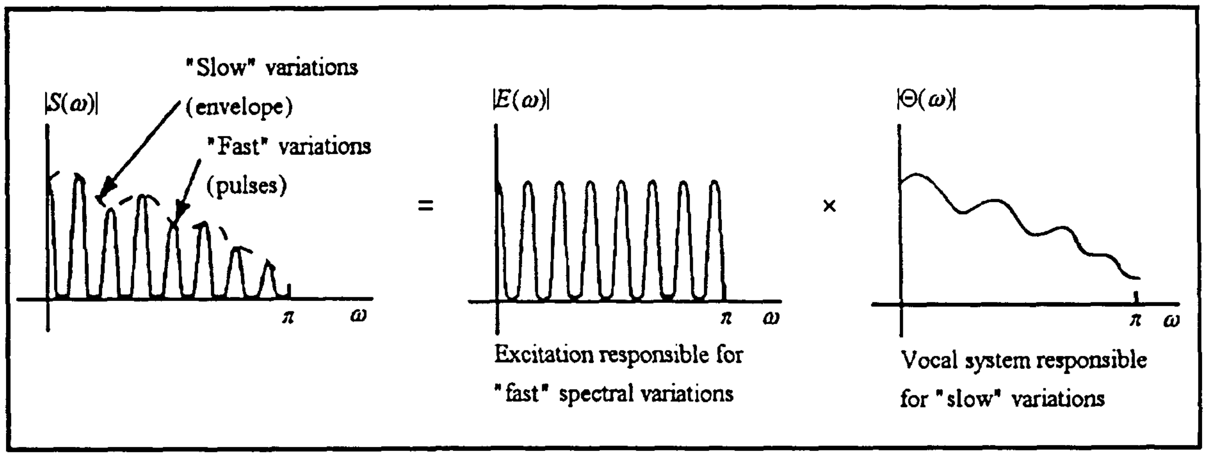


Figure 2.3 Vocal Response Including Excitation and Modulation

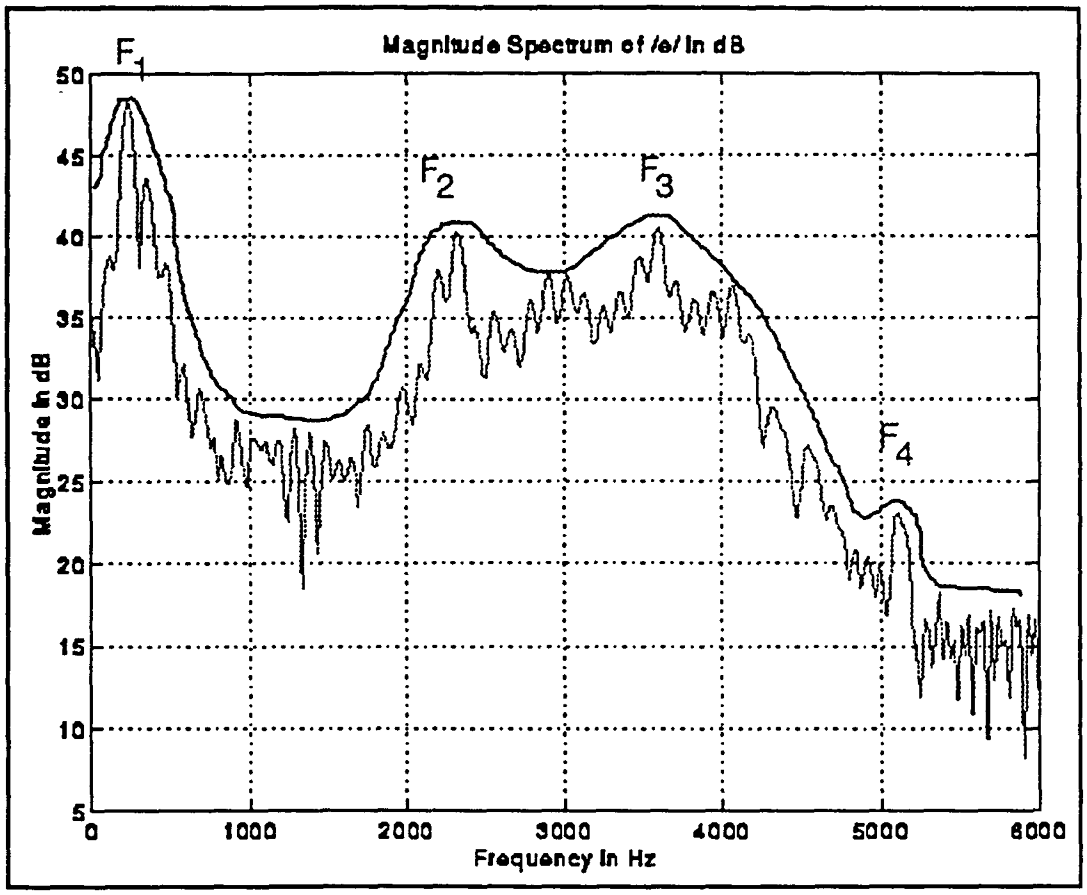


Figure 2.4 Formants for /e/

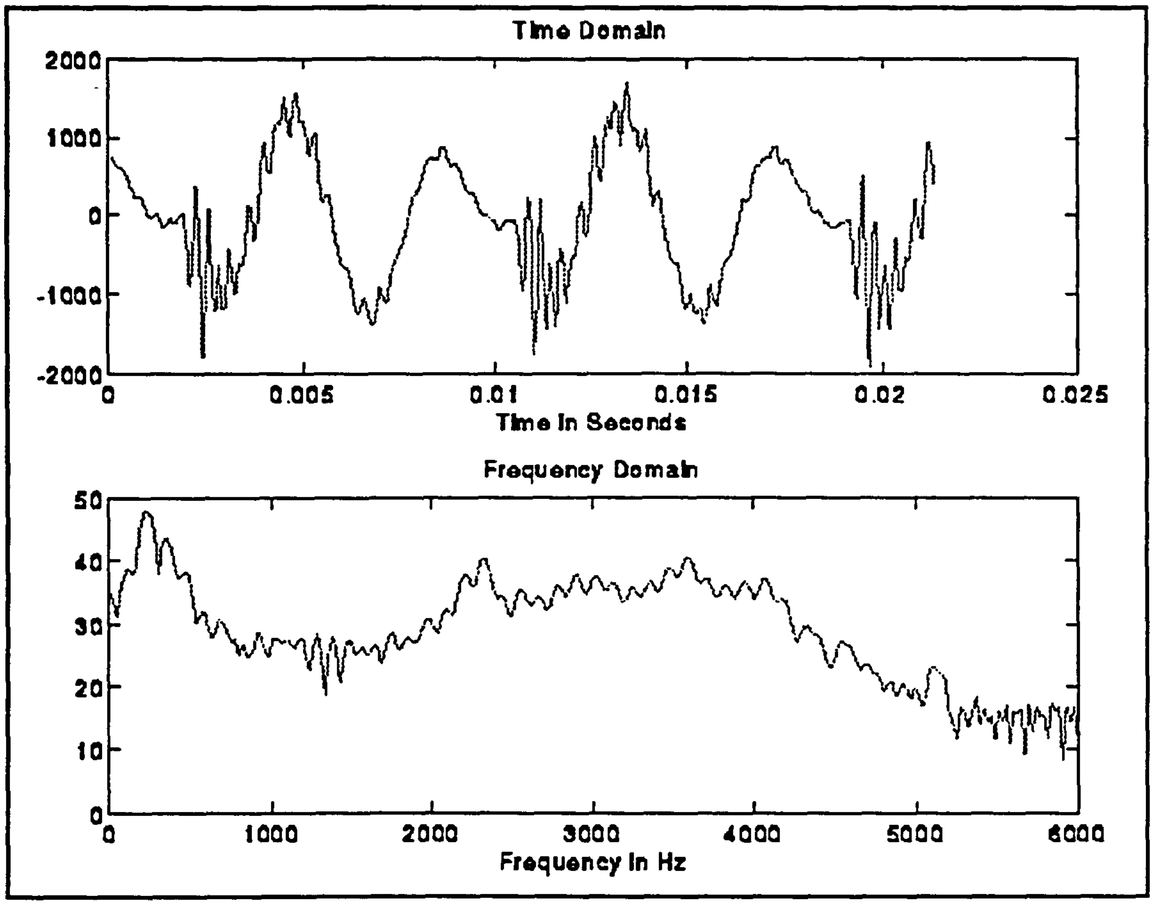


Figure 2.5 Time and Frequency Plot of /e/

system is one that only recognizes certain words (word spotting). Yet another possibility is a system that implements speech understanding. Speech understanding systems not only recognize the words but understand the context of the sentences. The order of complexity, from least to most complex, of these four types of recognizers are: isolated word, word spotting, continuous speech, and speech understanding. Speech systems are either built for individual speakers or multiple speakers. Having a system that works for multiple speakers opposed to a single speaker makes the system more complex.

**Problems Encountered**

The following are a few common problems that are encountered when developing a speech recognition system.

- Speaker Variations: Since no individual sounds exactly like another there is speaker dependent information in the speech signal.
- Ambiguity: There are many words like "two", "to" and "too" that sound the same. Also, words that sound similarly can cause problems such as "see", "bee", and "pea".
- Speech Variations: Individuals do not always pronounce words the same due to carelessness, phonetic variations, coarticulation, and temporal variations. Phonetic variations are slight changes in an individual's formant frequencies. Coarticulation refers to the phonetic features affected by the context of the speech. Temporal variations are time misalignments; words are not always pronounced in the same amount of time or with the same amount of time pronouncing the phonemes of a word.
- Noise & Interference: One must also consider the noise and interference that may be encountered when collecting speech samples.

These problems cause various difficulties when implementing a speech recognition system.

These speech related problems are coupled with signal processing issues. The signal processing issues that are commonly addressed are end point detection, time normalization, segmentation, and data compression. Generally, all of the signal processing issues must be taken into consideration while some of the speech problems can usually be ignored. Ignoring the speech problems usually makes a less robust but functional system. Many times systems are limited to a single speaker or the data is intentionally collected in a noise free environment to minimize the speech problems.

## Speech System

The speech recognition system developed for this research work is shown in Figure 2.6. The feature extraction block in Figure 2.6 can be replaced with one of the block diagrams illustrated in Figure 2.7, which generate the six feature vectors that are being compared in this work. Only feature vectors that are in the frequency domain are considered since there is a larger amount of accessible information in the frequency domain. In the next several sections, each component of the speech recognition system is discussed in detail.

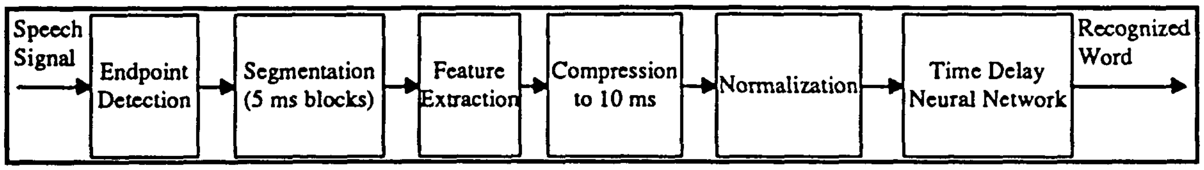


Figure 2.6 Speech Recognition System

The speech samples were collected using an Ariel DSP board. The sampling rate was set to 12 kHz in order to make comparisons with others' work. As mentioned in the introduction, the system uses isolated words and is speaker dependent. The words /b/, /d/, /e/, and /v/ were collected, typically 3 sets of the 4 words at a time. When processing the words the first 150 ms of the word was used since most of the distinguishing information is located in this region. There are 150 of each words for a total of 600 words. Not all 600 words were collected at the same time; the collection process spanned over three days to allow more variation in the speaker's voice. These 600 words were broken into training and recognition groups. Roughly 400 words were used for training while the other 200 were used for the recognition task. An entire speech sample of 10 seconds, containing 12 words, is shown in Figure 2.8.

## Signal Processing

### Endpoint Detection

The speech samples consisted of several isolated words and required a method of extracting the individual words from the silence and other words. The method used to extract the words was a simple standard deviation comparison. The steps for the standard deviation method are outlined in Figure 2.9.

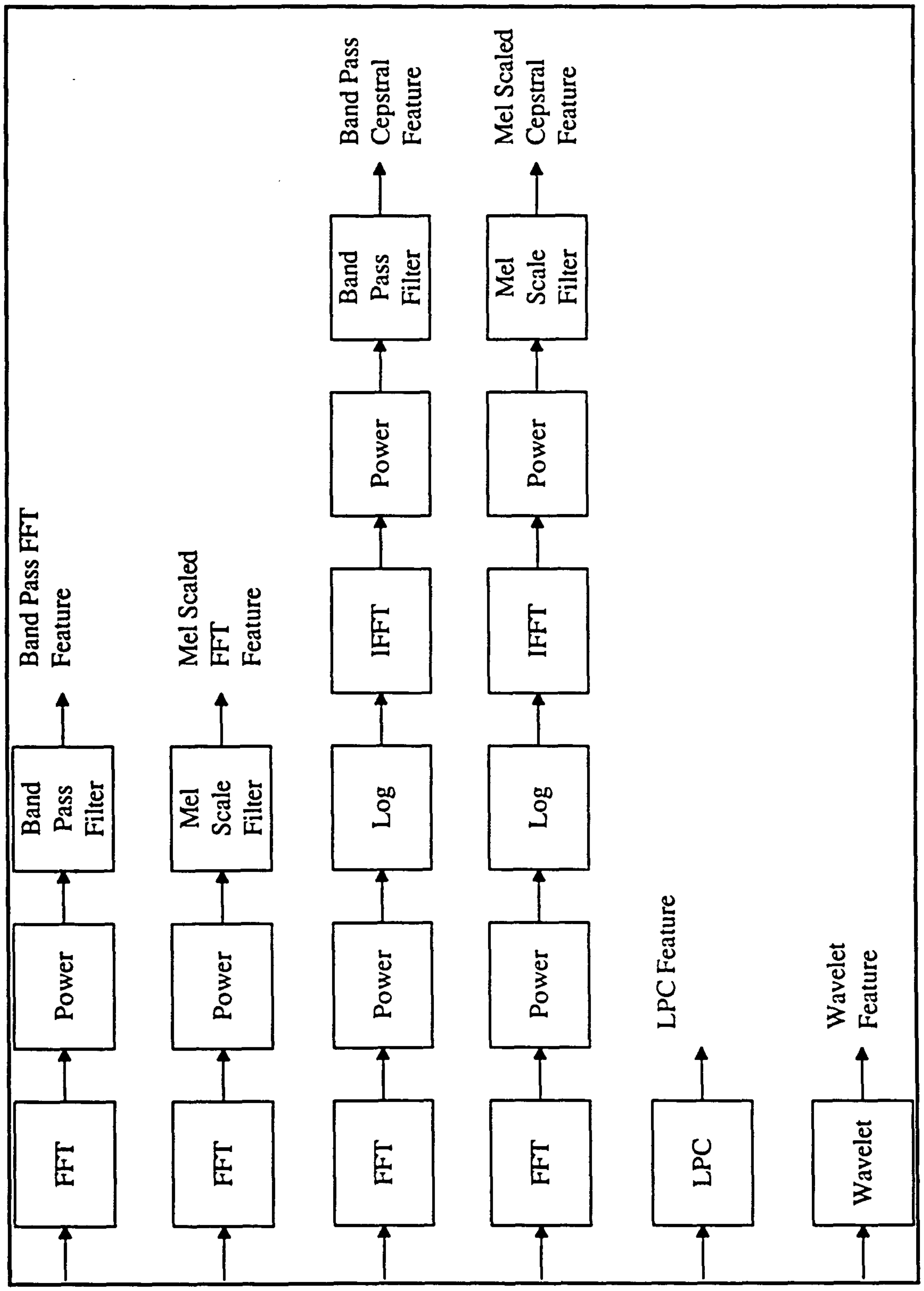


Figure 2.7 Block Diagrams for Feature Extraction

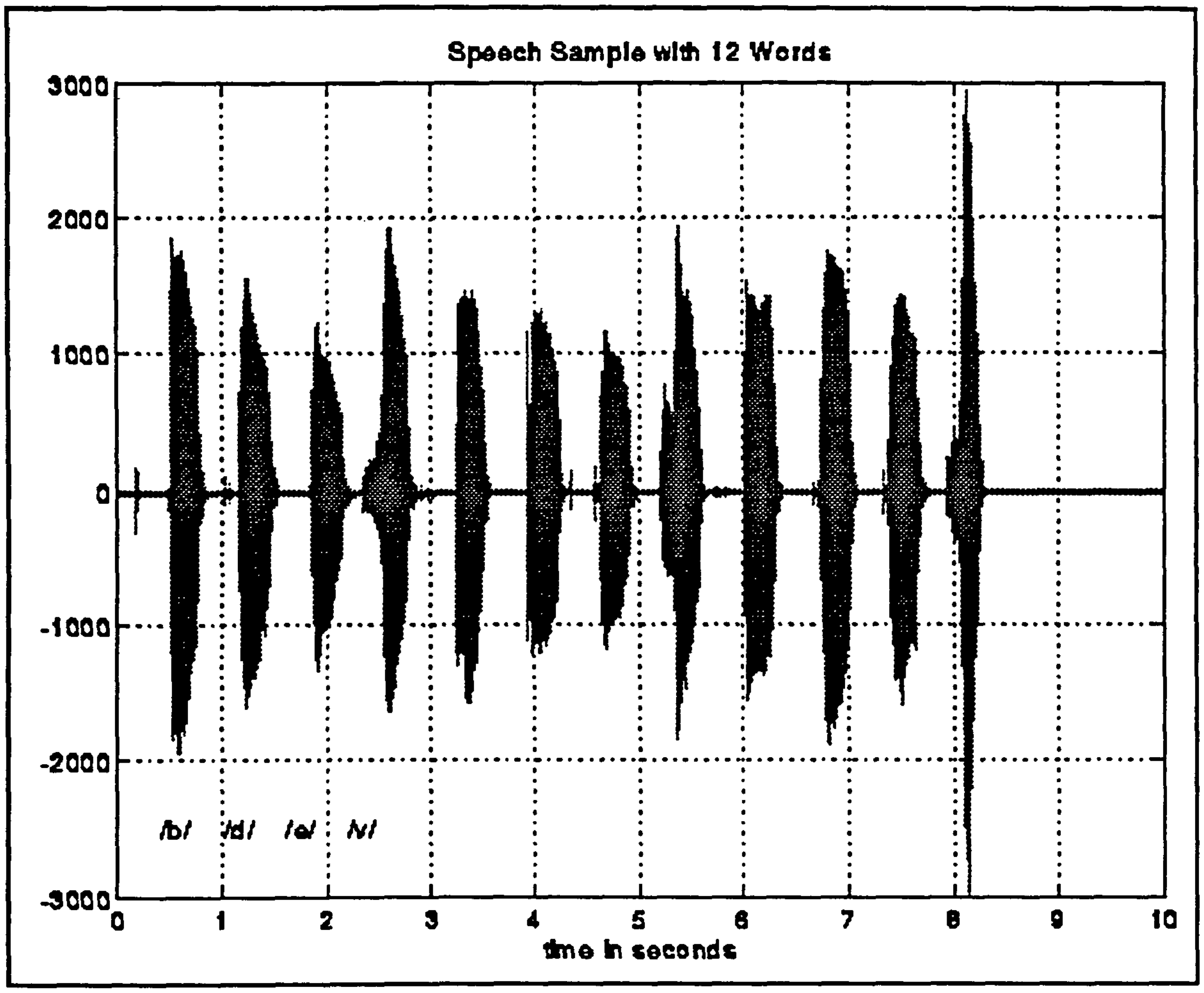


Figure 2.8 Speech Signal Containing 12 Words

Standard Deviation Endpoint Detection

Step 1. Determine the standard deviation of the silent parts of the speech signal. The samples between 600 and 1800 are used (there are 120000 samples in one data set) for this calculation. The standard deviation is calculated using:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x(i)^2 - \overline{x}^2\right)}, \text{ where } \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x(i).$$

Step 2. Determine the standard deviation of the entire speech sample for 5 ms blocks.

Step 3. Find the starting point of the word - If the standard deviation of each of three consecutive blocks is greater than three times the standard deviation of the silence the start of a word is detected.

Step 4. Find the end point of the word - If any two of the last four consecutive blocks standard deviation fall below three times the silence standard deviation the end of a word is detected. There is also a minimum length of 150 ms for the word.

Step 5. Go to Step 3. Repeat until end of speech sample is reached.

Figure 2.9 Endpoint Detection Scheme

A small speech sample with one word and the endpoints detected for the sample are shown in Figure 2.10. This very simple procedure gives reasonably good results when there is little noise in the environment. If the environment is noisy, however, the standard deviation of the silent section maybe large and it could be difficult to separate the noise from the words. When building speech recognition systems for noisy environments, more complex methods of endpoint detection are required [3].

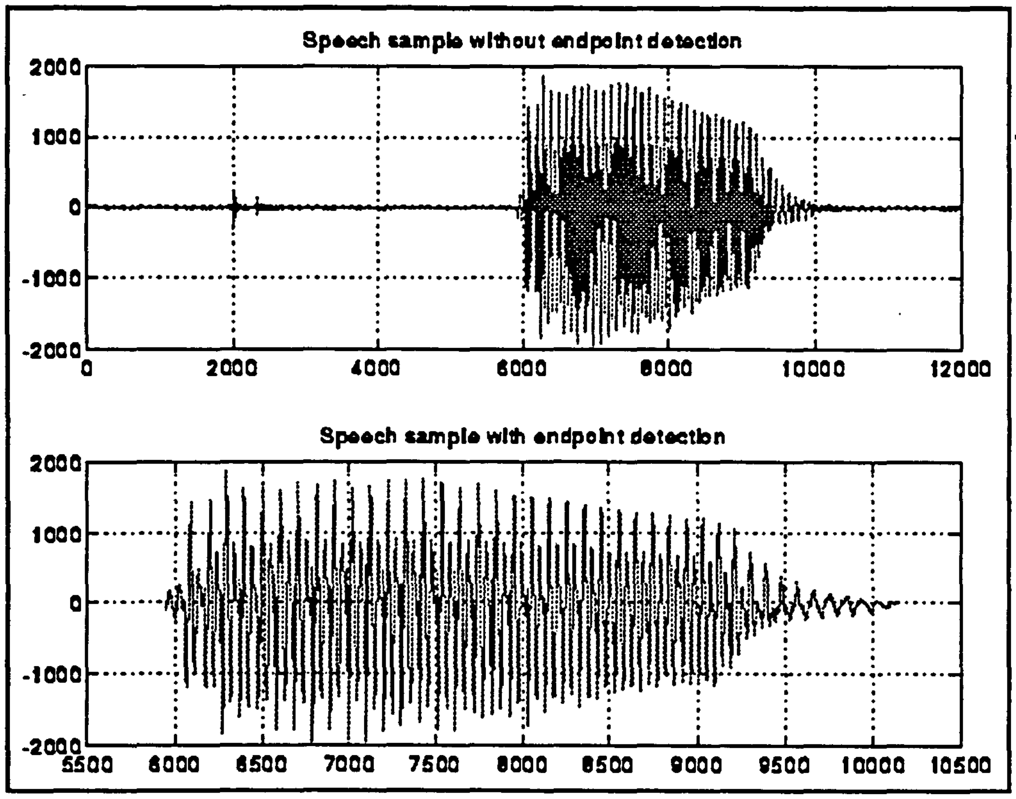


Figure 2.10 Processing of Speech Sample using Endpoint Detection Scheme

**Blocking (Segmentation)**

The segmentation procedure for the speech samples corresponding to each word was very simple. Segmentation was accomplished by simply dividing the word into 5ms blocks. Given that the sampling rate was 12 kHz, each 5 ms block contained 60 samples. The shift invariance property of the time delay neural network allowed the use of this simple segmentation procedure. If the time delay neural network were not shift invariant, a more complicated procedures for segmentation might have been required.

**Fourier Transform and Inverse Fourier Transform**

This section begins with an introduction to the discrete-in-time, continuous-in-frequency Fourier transform (DTFT) and then discusses the discrete Fourier transform (discrete both in time and frequency) and its implementation. The discrete Fourier transform (DFT) is used in this work. The Fourier transform uses a series of sinusoidal wave forms with varying amplitudes and frequencies to represent a given signal. The defining equations for the DTFT and its inverse are given in Figure 2.11.

| | |
|---|---|
| $X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$ | Fourier Transform |
| $x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega$ | Inverse Fourier Transform |

Figure 2.11 DTFT

The Fourier transform of a function x is denoted by $\mathcal{F}\{x\}$. The Fourier transform is used for analyzing a sequence to determine the frequency components that are required to generate the sequence. It is common to break the Fourier transform into magnitude and phase components, shown here.

$$X(e^{j\omega}) = |X(e^{j\omega})|e^{j\angle X(e^{j\omega})}$$

The Fourier transform is sometimes referred to as the spectrum, since it gives the amplitude and phase of the sinusoids required to generate a given sequence.

The discrete Fourier transform is often used in practical situations, since having discrete frequency components allows simple implementations. These implementations of the DFT are referred to as FFTs (Fast Fourier Transforms). Since the Fourier series representation is ideal for periodic signals, the approach taken for the DFT is to construct a periodic sequence for which the period is equal to the original finite length sequence. The defining equations for the DFT and its inverse are given in Figure 2.12. Both X(k) and x(n) in Figure 2.12 are periodic sequences.

The DFT was implemented with a decimation-in-time FFT algorithm [10]. The 5 ms blocks (60 samples) were zero padded to 256 samples and then the 256 point FFT was computed for the block. The two methods used to generate the 16 coefficients were the filter bank and mel-scale filter, discussed shortly.

## Cepstrum

The cepstrum transform was given its name because the cepstrum is the inverse of the spectrum. With the word cepstrum, other common terms have been changed to reflect the inverse transform; these terms include quefrency (inverse frequency) which is the independent parameter, and liftering (filtering) to "filter" the quefrencies [10]. The key feature of the cepstrum is that it allows for the separate representation of the glottal excitation and the vocal

$$X(k)=\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi nk}{N}}$$

Commonly the following abbreviation is used.

$$W_N=e^{\frac{j2\pi}{N}}=\cos\frac{2\pi}{N}+j\sin\frac{2\pi}{N}$$

Which gives: $X(k)=\sum_{n=0}^{N-1} x(n)W^{-nk}$

The inverse DFT is: $x(n)=\frac{1}{N}\sum_{k=0}^{N-1} X(k)W^{nk}$

Figure 2.12 Discrete Fourier Transform

tract modulation. The separation of the excitation and modulation can also be thought of as a deconvolution. The discussion begins with the real cepstrum, since it is a little easier to understand. Then the complex cepstrum (implemented in this work) is discussed. The defining equation for the real long term cepstrum is:

$$c_s(n)=\mathcal{F}^{-1}\{\log|\mathcal{F}\{s(n)\}|\}=\frac{1}{2\pi}\int_{-\pi}^{\pi}\log|S(\omega)|e^{j\omega n}d\omega$$

The block diagram of the real cepstrum computation is given in Figure 2.13.

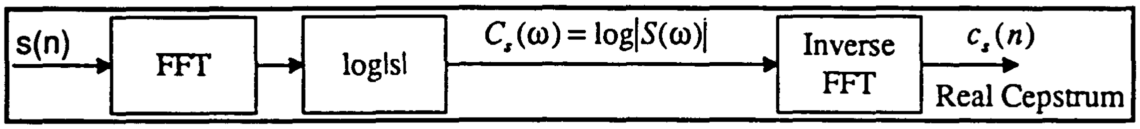


Figure 2.13 Computation of Real Cepstrum

If the signal s(n) is composed of two parts (excitation and modulation) the first two blocks given in Figure 2.13 give the following representation of the signal.

$$\begin{aligned}C_s(\omega)&=\log|S(\omega)|\\&=\log|E(\omega)\Theta(\omega)|\\&=\log|E(\omega)|+\log|\Theta(\omega)|\\&=C_e(\omega)+C_\theta(\omega)\end{aligned}$$

Now, the signal is represented as a sum of the excitation and modulation; then the inverse Fourier transform is taken giving the real cepstrum $c_s(n) = c_e(n) + c_\theta(n)$. This expression gives the desired characteristics of linear separation of the excitation and modulation. The transition from the long-term real cepstrum to the short-term real cepstrum is accomplished by using the short term DFT and inverse along with zero padding of the signal. The real cepstrum can be used for two key applications in speech processing: pitch estimation and format estimation [11].

Now the complex cepstrum will be discussed; the discussion will start with the long term concepts knowing that the transition to the short-term complex cepstrum will be accomplished with a short-term DFT. The complex cepstrum is a subclass of homomorphic signal processing [10]. The real cepstrum is not a subclass of homomorphic signal processing, however, since the original signal can not be reconstructed from the real cepstrum. The key difference between the complex cepstrum and the real cepstrum is the complex logarithm that is used by the complex cepstrum. The definition of the complex logarithm and its application to the Fourier transform of a signal is:

$$\log z = \log|z| + j\arg|z| \qquad \text{complex log}$$
$$\log S(\omega) = \log|S(\omega)| + j\arg\{S(\omega)\}$$

In order for $\log S(\omega)$ to be unique the $\arg\{S(\omega)\}$ must be selected to be an odd continuous function of $\omega$ [10]. Figure 2.14 illustrates the separation of the excitation and modulation and the block diagram of the complex cepstrum is shown in Figure 2.15. Diagram 2.15 is redrawn in Figure 2.16 to show the computation of the real and imaginary parts of the complex log, which leads to the decomposition of the signal into its even and odd parts. The cepstrum for the 5ms blocks was implemented using a 256 point FFT. The 16 coefficients were generated with the linear and mel-scale filter banks. Figure 2.17 shows a plot of the complex cepstrum for the phoneme /e/.

## Filter Bank

The filter bank is simply a set of band pass filters. There are 16 filters that cover the 6000 Hz frequency range of the data. These filters divide the frequency range linearly, so that every filter covers 375 Hz. Then, to generate the feature coefficients with the filter bank, the strength of the frequency components in each range are summed. The features are determined

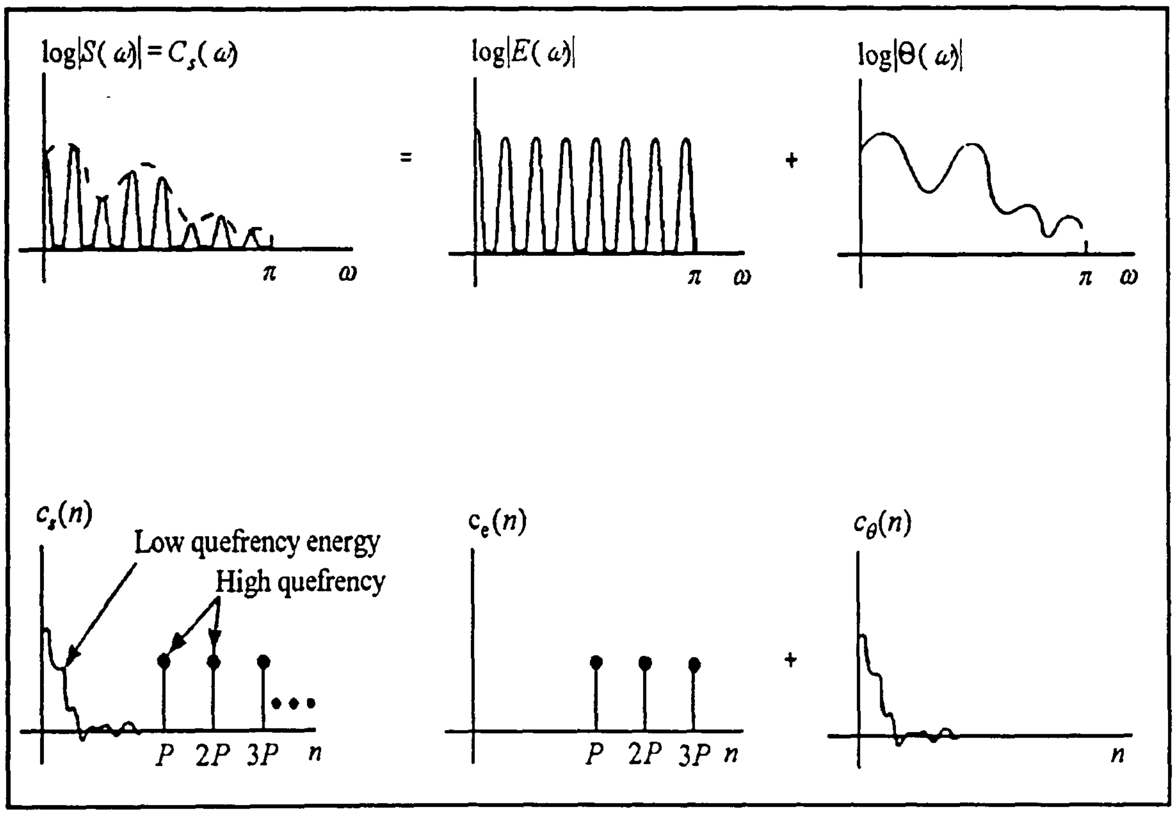


Figure 2.14 Separation of Excitation and Modulation

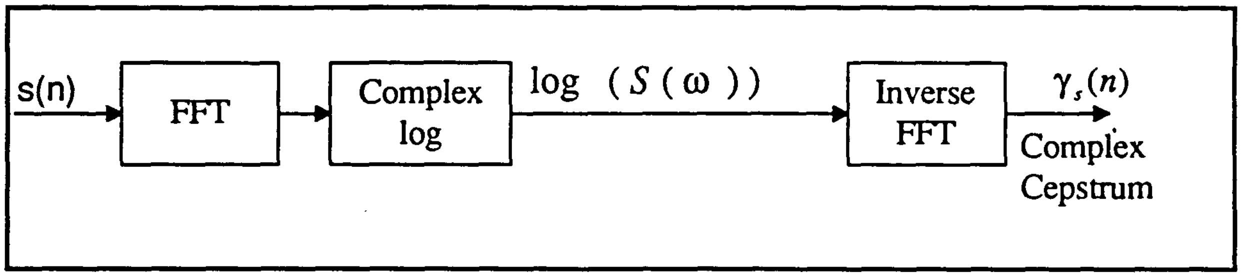


Figure 2.15 Computation of Complex Cepstrum

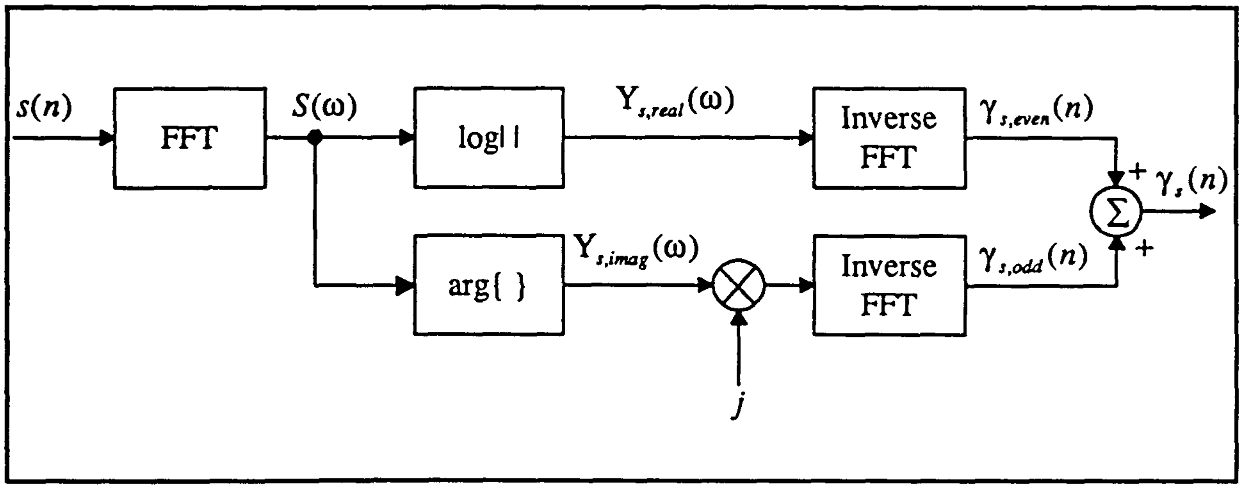


Figure 2.16 Decomposition of Complex Cepstrum into Even and Odd Parts

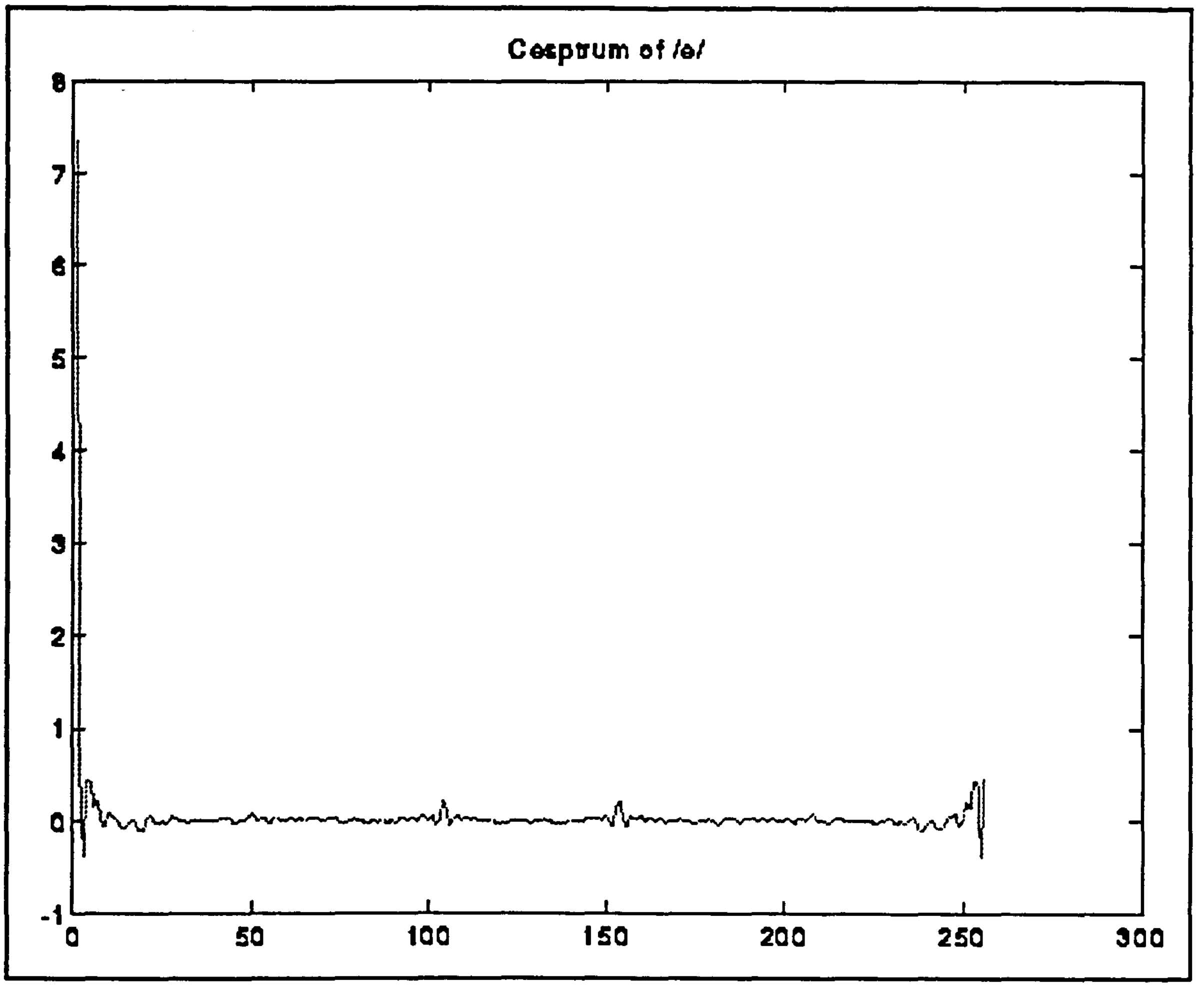


Figure 2.17 Complex Cepstrum of /e/

using the following equation:

$$c(i) = \sum_{j=(i-1)*375}^{i*375} s(j), \text{ for i} = 1, \mathrm{K} \ , 16$$

where c(i) is the feature coefficient and s(j) is the spectrum or cepstrum of the block. The summing of the frequency strengths yields the 16 coefficients that are required by the time delay neural network.

## Mel-Scale Filter

The mel is the unit of pitch, where pitch is defined relative to some reference frequency. The mel-scale is a logarithmic scale that relates the frequency (in Hz) to the pitch (in mels). One approximation of the mel-scale is: $y = 1000 \log_2\left(1 + \frac{f}{1000}\right)$. In Figure 2.18 pitch in mels is plotted versus frequency. To create the mel-scale filter a selected range in mels is taken and then all of the frequency components in that range is added together to generate the mel-scale coefficient. The mel-scale frequency ranges used to generate the mel-scale coefficients are given in Table 2.1 for a sampling rate of 12 kHz.

## Linear Prediction Coefficients

Linear prediction, when used for speech recognition, predicts the spectrum envelope of the speech signal. The goal of linear prediction is to predict the current output based on the previous outputs. The linear prediction estimate and prediction error are given in Figure 2.19.

There are two common ways of obtaining the optimum predictor coefficients, the autocorrelation and covariance methods. The autocorrelation and covariance names are derived from the fact that the data representation reduces to the short-term autocorrelation and the covariance matrix respectively. The autocorrelation method is used in this research, so only that method will be discussed. The autocorrelation method of obtaining the predictor coefficients using the Levinson-Durbin recursion is shown in Figure 2.20 [11]. Levinson-Durbin recursion solves a system of equations of order n assuming that there is already a solution for the predictor of order n-1.

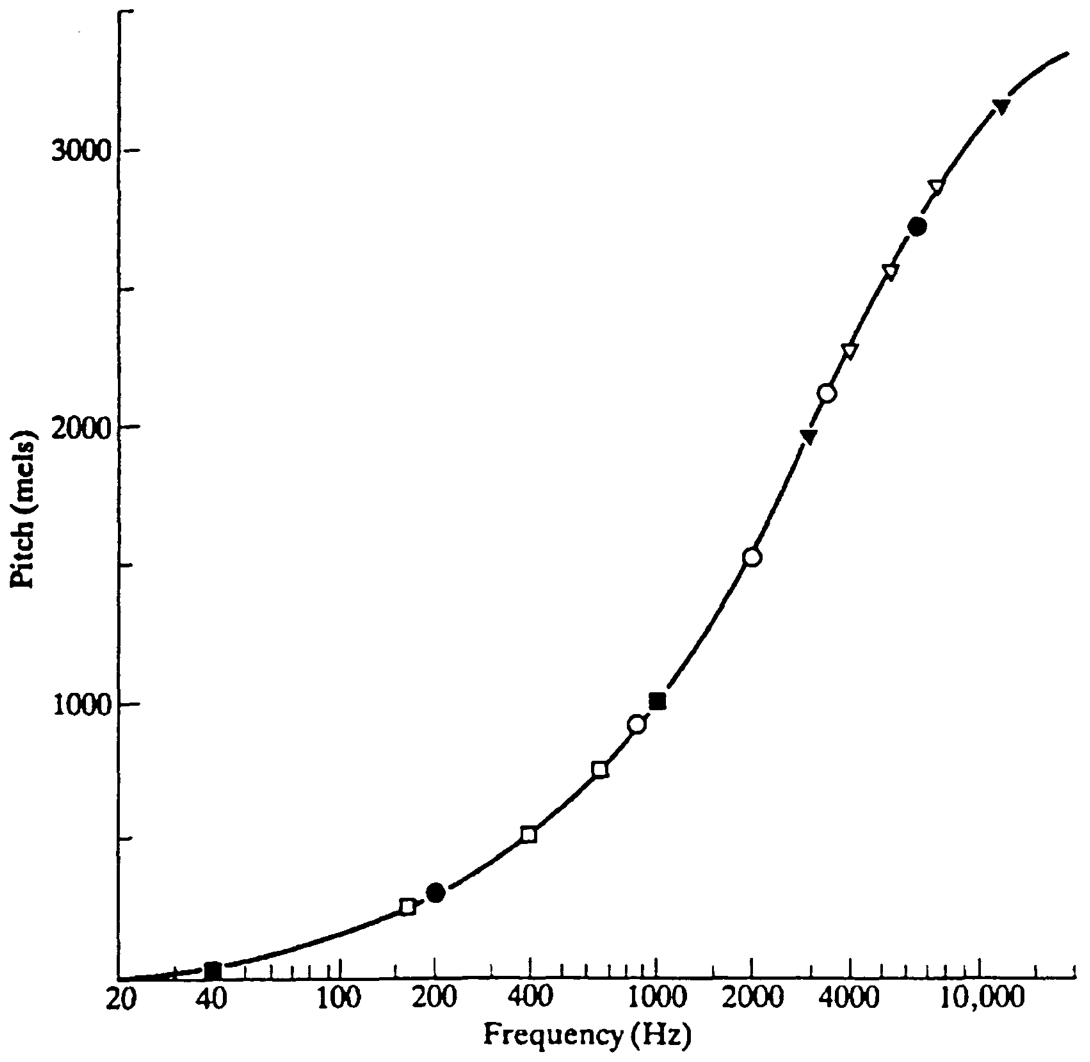


Figure 2.18 Mel-Scale

Table 2.1 Mel-Scale Coefficients

| Mel-Scale Coefficients for 12 kHz Sampling Rate | |
|---|---|
| Coefficient | Frequency Range |
| m(1) =s(1)+s(2)+s(3)/2 | 0- 141 |
| m(2) =s(3)/2+s(4)+...+s(6)+s(7)/2 | 141- 328 |
| m(3) =s(7)/2+s(8)+...+s(10)+s(11)/2 | 328- 516 |
| m(4) =s(11)/2+s(12)+...+s(14)+s(15)/2 | 516- 703 |
| m(5) =s(15)/2+s(16)+...+s(18)+s(19)/2 | 703- 891 |
| m(6) =s(19)/2+s(20)+...+s(22)+s(23)/2 | 891-1078 |
| m(7) =s(23)/2+s(24)+...+s(26)+s(27)/2 | 1078-1266 |
| m(8) =s(27)/2+s(28)+...+s(30)+s(31)/2 | 1266-1453 |
| m(9) =s(31)/2+s(32)+...+s(35)+s(36)/2 | 1453-1688 |
| m(10) =s(36)/2+s(37)+...+s(41)+s(42)/2 | 1688-1969 |
| m(11) =s(42)/2+s(43)+...+s(48)+s(49)/2 | 1969-2297 |
| m(12) =s(49)/2+s(50)+...+s(57)+s(58)/2 | 2297-2719 |
| m(13) =s(58)/2+s(59)+...+s(68)+s(69)/2 | 2719-3234 |
| m(14) =s(69)/2+s(70)+...+s(81)+s(82)/2 | 3234-3844 |
| m(15) =s(82)/2+s(83)+...+s(97)+s(98)/2 | 3844-4594 |
| m(16) =s(98)/2+s(99)+...+s(116)+s(117)/2 | 4594-5484 |

Linear Prediction Estimate

$$\hat{y}(n) = -\sum_{i=1}^{p} a(i)y(n-i),$$

where $-a(i)$ is the predictor coefficients and $p$ is the predictor order

Prediction Error

$$e(n) = y(n) - \hat{y}(n) = \sum_{i=0}^{p} a(i)y(n-1)$$

Figure 2.19 Linear Prediction Equation

Autocorrelation Recursive Algorithm

Step 1. For n=0, $E_0 = r_0$, where $E$ is the error term and

$$r_i = \sum_{n=i}^{N-1} y(n)y(n-i) \quad \text{(the autocorrelation)}$$

Step 2. For step n,

$$k^n = \frac{-1}{E^{n-1}} \sum_{i=0}^{n-1} a_i^{n-1} r_{n-i} \quad k \text{ is the reflection coefficient}$$

$$a_n^n = k^n$$

For $i = 1, K\ , n-1$

$$a_i^n = a_i^{n-1} + k^n a_{n-i}^{n-1}$$

$$E^n = E^{n-1}\left(1-\left(k^n\right)^2\right)$$

Step 3. Repeat Step 2 until n=p, the order of the prediction desired.

Figure 2.20 Autocorrelation Recursive Algorithm

## Wavelet Transform

The wavelet transform has many properties in common with the Fourier transform, but has some distinguishing differences that might give the wavelet transform an advantage over the Fourier transform for speech recognition [12]. The two key differences between the wavelet and Fourier transforms is the mapping and the number of functions. The Fourier transform maps the time domain to the frequency domain, while the wavelet transform maps the time domain to a scale domain. Fourier analysis decomposes a signal into individual frequency components, but does not tell when the frequencies occurred. The scale domain of the wavelet transform, however, contains information about the frequencies in the signal and when they occurred. There are many wavelet functions available while there is only a single function for the Fourier transform.

In wavelet analysis, a generating function, the wavelet, is selected and an associated transform gives a time-scale representation of functions. The wavelet function can be thought of as the impulse response of a band pass filter [14]. The time-scaling is accomplished by using contracted versions of the wavelet for fine temporal analysis and dilated versions for fine frequency analysis. The orthonormal wavelet bases constructed by Daubechies (used in this work), gives rise to a discrete, time-scale representation of finite energy signals [12].

The basic concept of the wavelet transform is varying the window size which yields better time-frequency resolution of signals. The uncertainty principle excludes the possibility of having arbitrarily high resolution in both time and frequency. By varying the window size, resolution in time can be traded for resolution in frequency. One way of achieving the trading of resolution is to have short high frequency basis functions and long low frequency basis functions. The trading is accomplished with the wavelet transform, where the basis functions are obtained from translating, dilating, and contracting a single wavelet. An example basis function is:

$$h_{a,b}(t) = \frac{1}{\sqrt{a}} h\left(\frac{t-b}{a}\right) \quad \text{with } a \in \mathrm{R}^{+},\ b \in \mathrm{R}$$

When $a$ is large, the basis function is a stretched version of the prototype wavelet (low frequency) and for small $a$ the basis function is a contracted version of the prototype wavelet (high frequency). Translation is obtained by adjusting the value of b. The wavelet transform is defined by the equation on the top of the next page.

$$X_w(a,b)=\frac{1}{\sqrt{a}}\int_{-\infty}^{\infty} h\left(\frac{t-b}{a}\right)x(t)dt$$

Daubechies wavelets are specified by a particular set of numbers, called wavelet filter coefficients. The Daubechies wavelets range from highly localized to highly smooth representations of the signals. The simplest (and most localized) of the Daubechies wavelets is called DAUB4 and has only four coefficients, $c_0,\ldots,c_3$. The DAUB4 wavelet was used in this work. The following is the transformation matrix that acts upon a column vector of data to its right.

$$\begin{bmatrix}
c_0 & c_1 & c_2 & c_3 & & & & & & & \\
c_3 & -c_2 & c_1 & -c_0 & & & & & & & \\
 & & c_0 & c_1 & c_2 & c_3 & & & & & \\
 & & c_3 & -c_2 & c_1 & -c_0 & & & & & \\
\vdots & \vdots & & & & & \ddots & & & & \\
 & & & & & & & c_0 & c_1 & c_2 & c_3 \\
 & & & & & & & c_3 & -c_2 & c_1 & -c_0 \\
c_2 & c_3 & & & & & & & & c_0 & c_1 \\
c_1 & -c_0 & & & & & & & & c_3 & -c
\end{bmatrix}$$

Here blank entries signify zeros. The first row of the matrix generates one component of the data convolved with the filter coefficients $c_0,\ldots,c_3$. Likewise so do the other odd rows. If the even rows followed this pattern, offset by one, then the matrix would be circular, that is an ordinary convolution that could be computed by an FFT. Instead of convolving with $c_0,\ldots,c_3$, however, the even rows perform a different convolution, with coefficients $c_3,-c_2,c_1,-c_0$. The action of the matrix is to perform two related convolutions, then to decimate each of them in half, and interleave the remaining halves. The values of the four coefficients are:

$$c_0=\left(1+\sqrt{3}\right)/4\sqrt{2} \qquad c_1=\left(3+\sqrt{3}\right)/4\sqrt{2}$$
$$c_2=\left(3-\sqrt{3}\right)/4\sqrt{2} \qquad c_3=\left(1-\sqrt{3}\right)/4\sqrt{2}$$

Figure 2.21 is a plot of the DAUB4 wavelet. The wavelet transform does not allow for a different number of outputs than inputs, therefore the 16 coefficients have to be generated in a slightly different manner for the 60 data samples. The 16 coefficients, for each 5 ms block, are generated by taking 4 wavelet transforms of size 16 and averaging the results. The range of data used are: 1-16, 15-30, 30-45, 45-60.

**Compression**

The coefficients were generated for the original 5 ms blocks of the words. To reduce the amount of data used to represent the speech signal the 5 ms blocks were compressed to 10 ms blocks. The compression was achieved by simply averaging adjacent blocks. Compressing the data also helped to reduce time alignment problems. When the data was averaged for 10 ms, instead of the original 5 ms, the resulting coefficients were smoother.

**Normalization**

There is a wide range of values that were determined for the feature coefficients. Having wide-spread coefficients can cause problems for neural networks, therefore the coefficients were normalized. To normalize the coefficients, for each representation of the words, the largest coefficient is determined. Then the coefficients for the given word were divided by the largest coefficient. Dividing by the largest coefficient provides a set of coefficients that were between 0 and 1. These smaller, less widely-spread coefficients seem to make the time delay neural network learn more quickly. This normalization also removes loudness information in the speech signals.

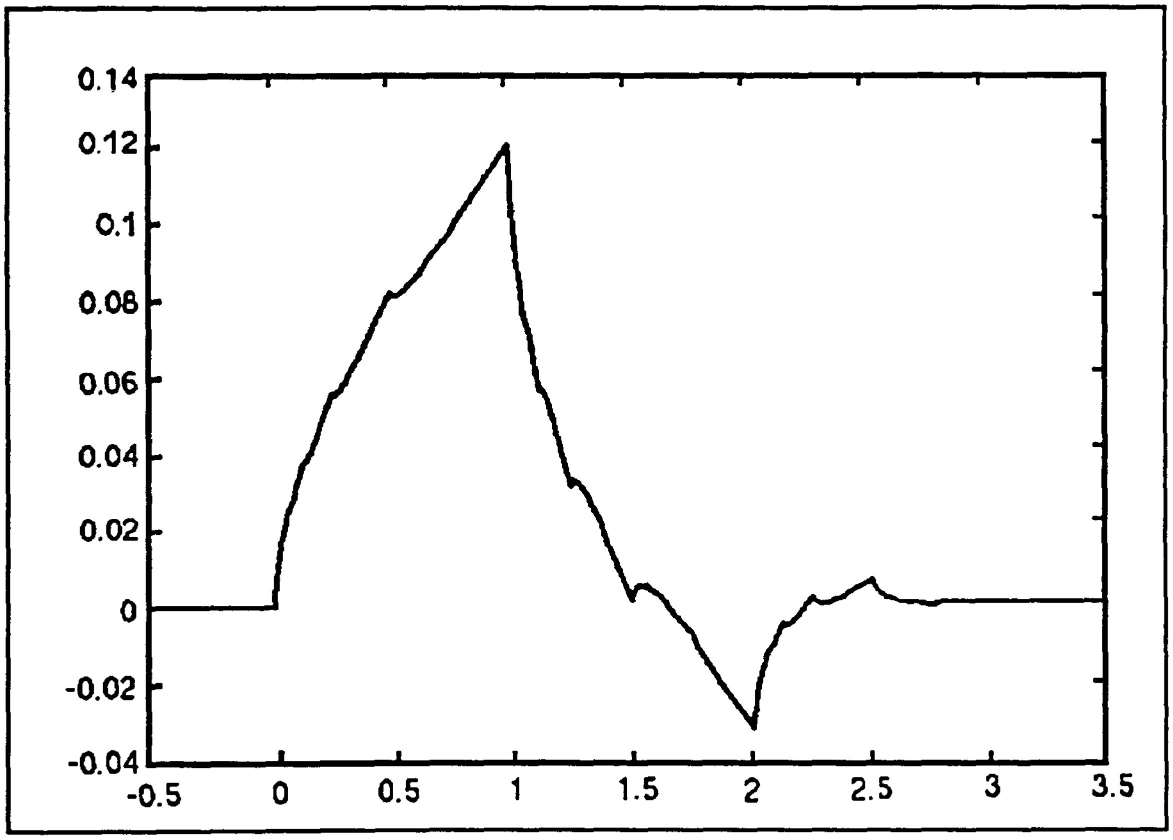


Figure 2.21 DAUB4 Wavelet

# CHAPTER 3. ARTIFICIAL NEURAL NETWORKS

This chapter gives a brief introduction to artificial neural networks (ANNs) in context of their use as pattern classifiers. Three models are discussed: the single layer perceptron, the multilayer perceptron, and the time delay neural network. An in-depth presentation of the time delay neural network is given, as this network was used for the recognition part of the speech system. For more detailed information on ANNs, consult [2,13], for example.

ANNs are simple mathematical models of the neurobiological structure (neural networks) of the brain. Figure 3.1 depicts a biological neuron and some local interconnections. Dendrites are fibers that transmit electrochemical pulses into the neuron. The neuron combines the inputs from all dendrites and if the combined signals are of the right strength, the neuron sends out an electrochemical pulse along axons, fibers that branch outward and meet other dendrites at junctions called synapses. As there are around 100 billion neurons in the brain, ANNs cannot hope to emulate biological neural networks, at least at this stage of development. However, ANNs have been shown to be quite good at solving certain pattern recognition problems, including those in speech recognition.

## Single Layer Perceptron

### Background

The perceptron is the first successful ANN architecture, developed by Rosenblatt in 1957 [2]. The perceptron grew out of Rosenblatt's research to model the neurological structure of the brain. The perceptron is successful only where decision boundaries between pattern classes are linear hyperplanes, and this severely narrows the class of problems the perceptron is useful for solving.

One main use of ANNs is for pattern classification. They also have the ability to determine function mapping, to deal with noisy data, to complete patterns, and to be adaptive in their solutions. For example, one of the first successful applications is handwritten character identification. Other areas that neural networks have been successful in providing solutions include: control systems, financial analysis, signal analysis, and pattern classification in biochemistry [1,13].

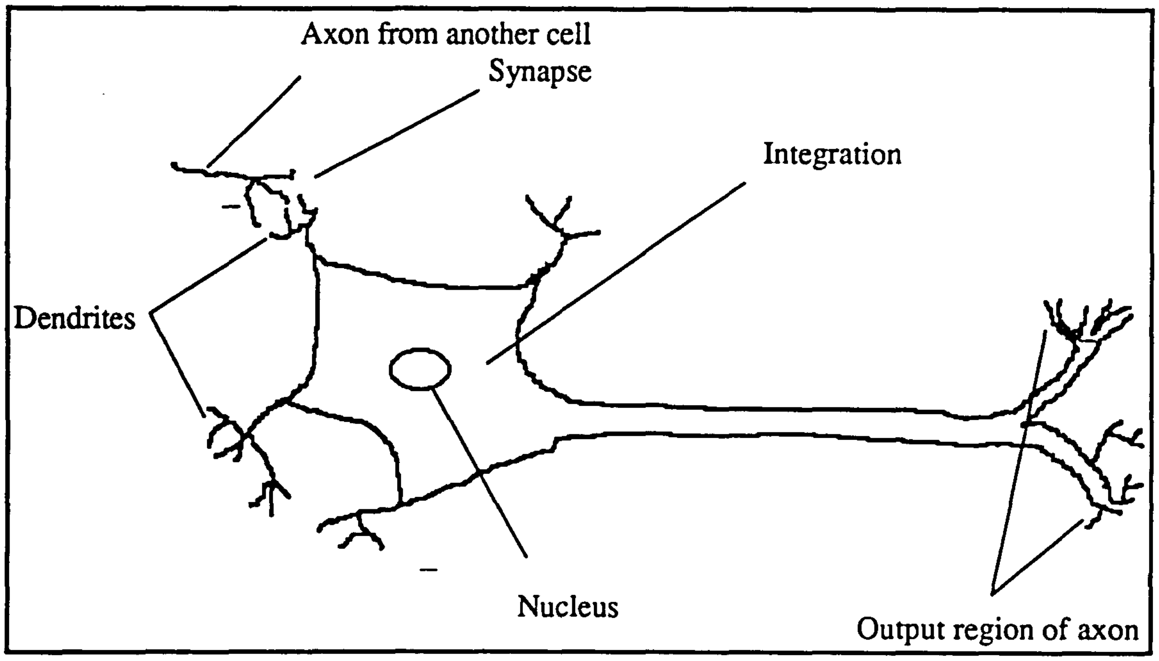


Figure 3.1 Biological Neuron

## Architecture

The architecture of an ANN is the set of "neurons," or nodes, which perform the numerical calculations, plus the connections linking certain nodes to others. The single layer perceptron (SLP) has a straight-forward architecture. See Figure 3.2. There are actually two layers in this network, the input layer of neurons, which performs no calculations, plus the output layer of neurons, where each neuron does computation. Each input neuron simply passes its numeric value along each connection to nodes in the upper layer to which it is connected. Each connection has a weight value associated with it: $w_{ji}$ is the weight value from lower node i to upper node j. At output node j, the value of the input node at location i is multiplied by the corresponding weight $w_{ji}$, summed, and then passed through a nonlinear activation function f. The final value at each output node is passed out of the network, and gives an indication of the pattern class represented by the input data. For M outputs, we have M pattern classes. Formally, the combining operation at each output node is

$$y_j = f\left(\sum_{i=1}^{N} x_i w_{ji}\right).$$

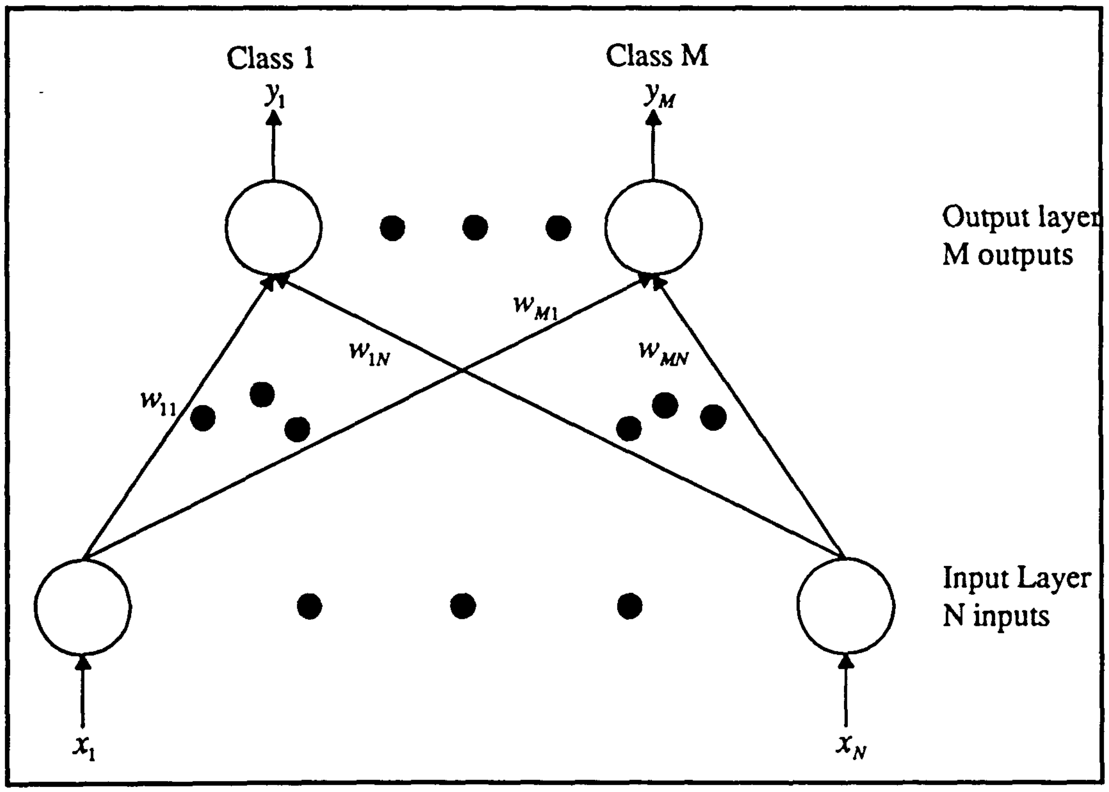


Figure 3.2 Single Layer Perceptron

## Activation Functions

Nonlinear activation functions give the neural network the ability to learn complex data sets. The hardlimiter and bipolar hardlimiter are often used for integer valued data sets, while the sigmoid and hyperbolic tangent are typically used for continuous data sets. These common activation functions are graphed in Figure 3.3. The equations of these four activation functions are as follows:

hardlimiter $f(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases}$

bipolar hardlimiter $f(x)=\begin{cases}1 & x\geq 0\\ -1 & x<0\end{cases}$

sigmoid $f(x)=\dfrac{1}{1+e^{-\alpha x}}$

hyperbolic tangent $f(x)=\tanh(x)$

The most commonly used activation function for the single layer perceptron is the hardlimiter.

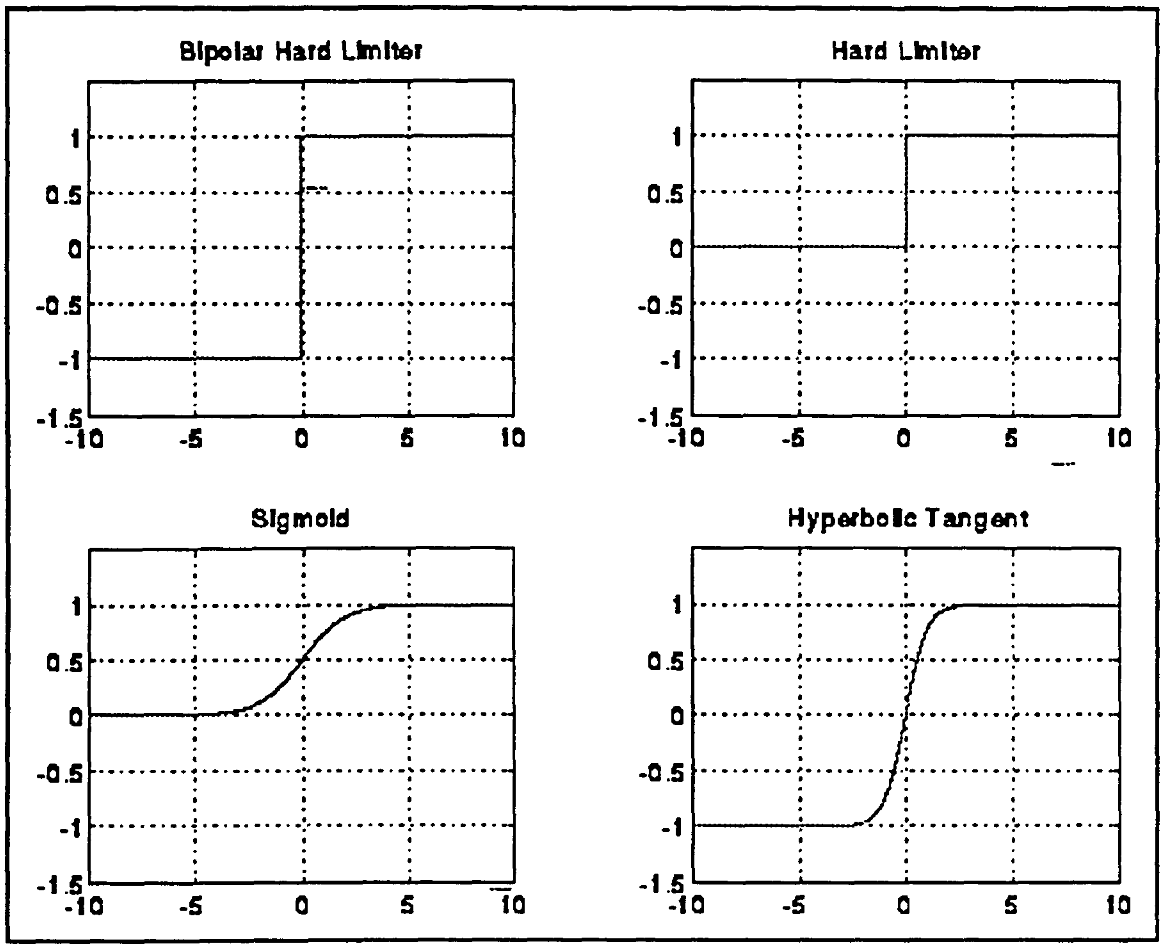


Figure 3.3 Nonlinear Activation Functions

## Decision Regions

ANNs can be viewed as complex dynamical nonlinear systems, mapping an N-vector of input data to an M-vector of outputs. Correct pattern classification occurs when the M-vector lies inside a certain region of M-space. Thus, the information about the mapping properties or classifications is contained in the weight values. As mentioned previously, the main drawback of the single layer perceptron is its linear decision boundary. The single layer perceptron is able to classify only linearly separable data sets. The requirement for the data to be linearly separable implies that the single layer perceptron can not even solve the XOR (Exclusive-or) decision boundary. An example, for two inputs and two outputs, of data sets that the single layer perceptron can distinguish is given in Figure 3.4. The light gray is the area that will be classified as region A and the dark gray is classified at region B. The XOR decision boundary that can not be solved by the single layer perceptron is given in Figure 3.5. There is a comparison of solvable decision regions for the perceptron based on the number of layers given in Figure 3.8, which is located in the section on the multilayer perceptron.

## Convergence Procedure

The term learning refers to the neural network discovering the association between inputs and desired outputs in terms of the weights $\{w_{ji}\}$. There are two types of learning: supervised and unsupervised. Supervised learning is based on presenting both the inputs and desired outputs to the ANN. The ANN uses the inputs and calculated outputs to adjusts the weights so on the next pass they classify the inputs correctly to the desired outputs. This presentation of inputs followed by a correction in the weight values is done iteratively, typically thousands of times. For unsupervised learning, the ANN is given the inputs and decides by the use of pre-defined heuristic algorithms how to separate the data into different classes. A supervised method is used in this work. The original supervised learning procedure developed by Rosenblatt is outlined in Figure 3.6.

Rosenblatt proved that if two classes of data are linearly separable, then the perceptron converges (using the method outlined in Figure 3.6) with a hyperplane separating the two data sets. If the two classes of data are not linearly separable then it is possible for the perceptron convergence procedure to oscillate continuously [6]. A procedure developed by Widrow and Hoff minimizes the least mean squared error to determine an optimal solution for data that is not linearly separable. This algorithm is simply called the Widrow-Hoff or LMS algorithm.

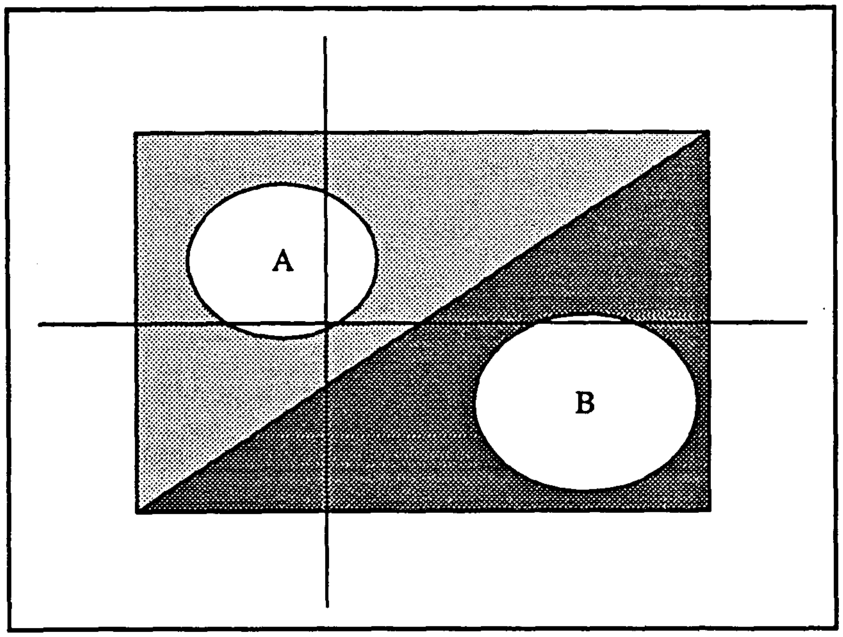


Figure 3.4 Solvable Decision Region for SLP

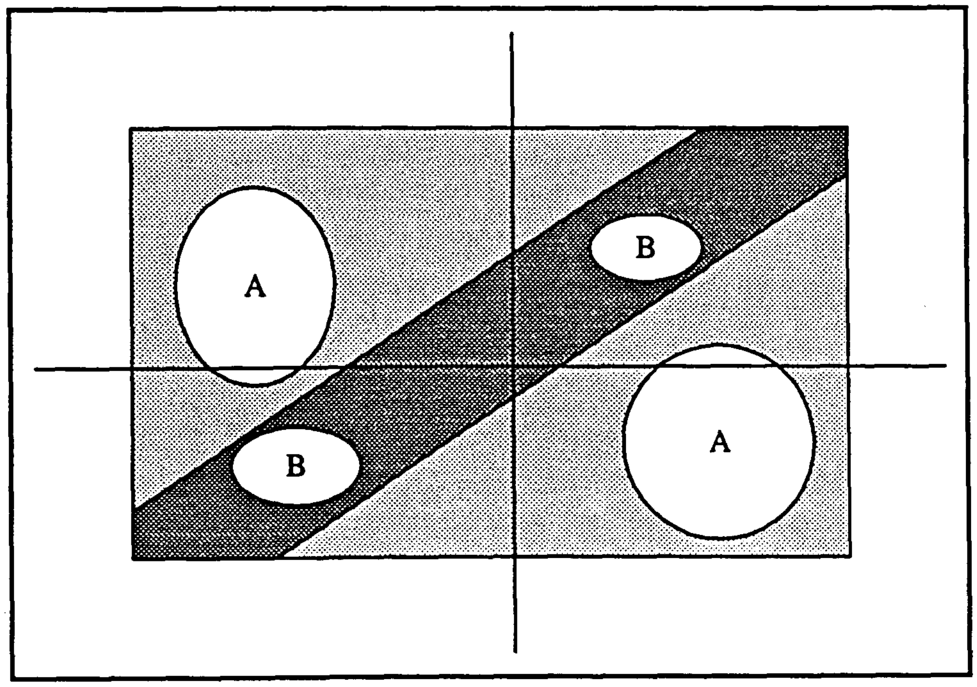


Figure 3.5 XOR Decision Region that is not Solvable by SLP

Learning Algorithm for Single Layer Perceptron

Step 1 Initialize the Weights and Threshold

Set the initial weights ($w_{ji}(0)$, $(0 \le i \le N-1, 0 \le j \le M)$) and threshold ($\theta$) to small random values, where $w_{ji}(t)$ is the weight from input node i to output node j at time t and $\theta$ is the threshold for the output node.

Step 2 Present New Input and Desired Output

Present a new input set $x_i(k)$, $0 \le i \le N-1$, where k is the index of the data sets. Along with the input give the desired output $d(k)$.

Step 3 Calculate the Actual Output

The calculation for the actual output is

$$y_j(k) = f\left(\sum_{i=0}^{N-1} w_{ji}(t)x_i(k) - \theta\right),$$

where f is a nonlinear activation function, $w_{ji}(t)$ is the weight from input node i to output node j, $x_i(k)$ is the input of node i, and $\theta$ is a threshold or offset.

Step 4 Adapt Weights

The weights are adapted in the following manner. It should be noted that if the calculated and desired output are the same the weights are not updated.

$w_{ji}(t+1) = w_{ji}(t) + \eta(d(k) - y(k))x_i(k)$, $0 \le i \le N-1$, and $\eta$ is the learning rate or gain factor $0 < \eta < 1$.

Step 5 Repeat by going to Step 2.

This process is completed when there is one complete presentation of the training data without having to adapt the weights.

Figure 3.6 Learning Algorithm for the SLP

The Widrow-Hoff follows Rosenblatt's procedure except the hardlimiter is replaced by a ramp function, shown in Figure 3.7. Both learning algorithms require the weights to be corrected on every trial by an amount that is related to the difference of the desired and calculated outputs.

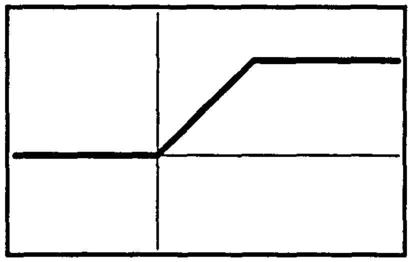


Figure 3.7 Ramp Function

## Multilayer Perceptron

### Architecture

The multilayer perceptron (MLP) takes the simple structure of the single layer perceptron and combines several SLPs. The architecture of the MLP has the ability to learn complex decision regions. There are multiple layers in the MLP, see Figure 3.8; the layers between the input and output layers are called hidden layers. It has been shown that no more than two hidden layers are required to learn data of any complexity [2]. The MLP is a fully interconnected network; full interconnection implies that every node of layer 1 has weighted connections to layer 2, every node of layer 2 is connected to layer 3, etc. There is no restriction on the number of input nodes, hidden nodes, and output nodes. There have been many attempts to give a general rule to determine the number of nodes, but no rule works for all cases. Usually the number of input nodes is determined by the input data set, and the number of output nodes is the number of desired classes. Lippmann makes the following suggestion on determining the number of nodes that are required for a particular data set [6]. In Lippmann's statement the first layer refers to the first or lowest hidden layer and these second layer is the second hidden layer or the one below the output layer.

"The number of nodes in the second layer must be greater than one when decision regions are disconnected or meshed and cannot be formed from one convex area. The number of second layer nodes required in the worst case is equal to the number of disconnected regions in input distributions. The number of nodes in the first layer must typically be sufficient to provide three or more edges for each convex area generated by every second-layer nodes. There should thus typically be more than three times as many nodes in the first as in the second layer."

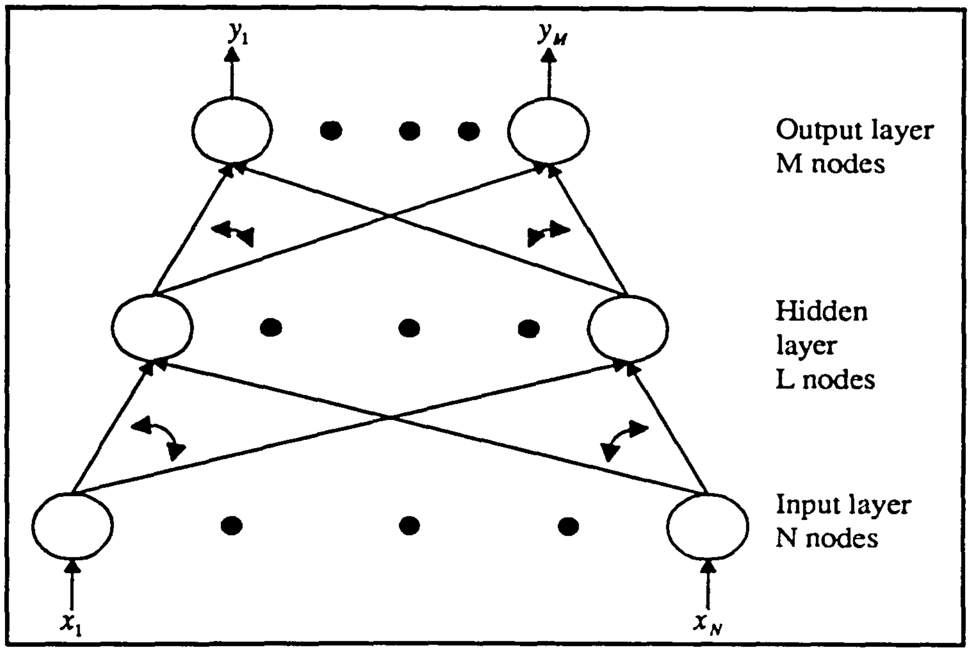


Figure 3.8 Multilayer Perceptron

## Decision Region

The architecture of the multilayer perceptron allows the MLP to learn more abstract data sets than the SLP. A comparison of the complexity of the decision regions for the single layer perceptron, a two layer perceptron, and a three layer perceptron are given in Figure 3.9 [6]. The areas Ⓐ and Ⓑ are the regions that are desired to be classified as class A or B. The gray and white regions are those that the ANN has chosen to be classes A and B. As depicted in Figure 3.9 the two layer perceptron has the ability to learn either convex open or convex closed regions. The convex regions can have no more sides than the number of nodes in the input layer. Knowing that the complexity of the convex region is dependent on the number of input nodes helps to determine the number of input nodes to be used in the neural network. If there are too many input nodes, however, the weights can not adjust properly to give reliable outputs. The three layer perceptron has the ability to classify arbitrarily complex decision regions. The 3 layer MLP even can separate meshed regions like that in the bottom right of Figure 3.9.

**Back-Propagation Learning Algorithm**

The back-propagation training algorithm was used to train the time delay neural network, and thus this training algorithm is presented in detail. Until the advent of the back-propagation learning algorithm, the multilayer perceptron was not used much, since there were no effective/efficient way to train the network. It is possible to train the MLP with the algorithm given in Figure 3.6, but the weights in some of the layers are required to be fixed which makes learning and training more difficult. With recent development of more advanced algorithms the popularity of the MLP has grown. The back-propagation (sometimes referred to as back-prop) learning algorithm is one of the most common methods for training the MLP and several other ANNs. The back-prop is an iterative gradient descent learning method designed to minimize the mean square error between the desired output and calculated output. An outline of the back-propagation learning algorithm is given in Figure 3.10. The basic procedure of the back-prop algorithm is to calculate the actual output given an input data set and the initial weights, find the mean square error between the desired and calculated outputs, adjust the weights between the highest two levels, use the error term to generate an error term for the next lower level, update the weights at the next lower level, and continue this process until reaching the weights between the lowest hidden layer and the input layer. The back-propagation algorithm requires a continuous activation function; usually either the sigmoid or hyperbolic tangent are used.

Considering the possibility that the back-propagation method may find a local minimum instead of a global minimum it usually provides very good results. The momentum term discussed in Step 4 of Figure 3.10 may sometime alleviate the problem of local minimums by bouncing the gradient descent out of a local minimum. Also, having random valued starting weights may help to jump over or out of some local minimum. The biggest drawback of this algorithm is the fact that many presentations of the training data are usually required for the network to learn the data.

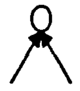
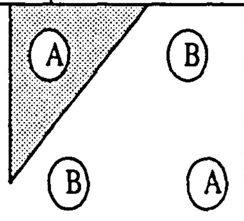
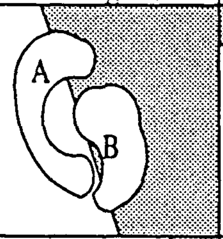
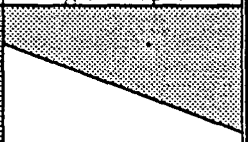
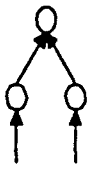
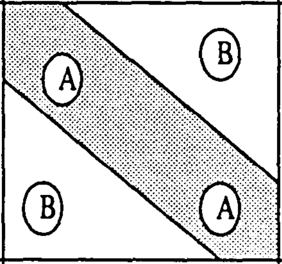
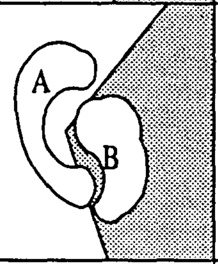
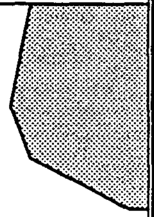
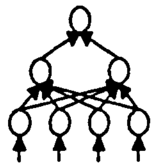
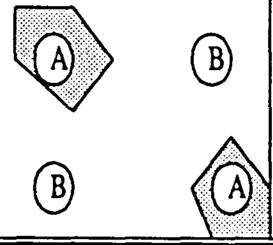
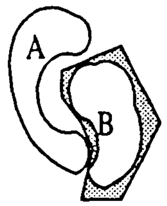
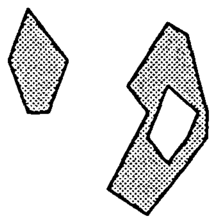
| Structure | Types of decision regions | Exclusive OR problem | Classes with meshed regions | Most general region shapes |
|---|---|---|---|---|
| Single Layer  | Half Plane Bounded by Hyperplane |  |  |  |
| Two Layer  | Convex Open or Closed Regions |  |  |  |
|  | Arbitrary (Complexity limited by number of nodes) |  |  |  |

Figure 3.9 Decision regions of the perceptron for various number of layers

Back-Propagation Learning Algorithm

Step 1 Initialize the Weights and Offsets

Set all of the weights to small random numbers (usually between -1 and 1). This includes initializing the weights from the offsets to the nodes above.

Step 2 Present Inputs and Desired Outputs

Present a new input set $x_i(k)$, $0 \le i \le N-1$, where k is the index of the data sets. Along with the input specify the desired output $d(k)$. If the network is being used as a classifier then generally the desired outputs are all 0 except for the node corresponding to the desired class which is given the value of 1.

Step 3 Calculate the Actual Outputs

Calculate the outputs at each layer starting at hidden layer 1 and working toward the output layer. The output equitation is $y_j(k) = f\left(\sum_{i=0}^{N-1} w_{ji}(t)x_i(k) - \theta\right)$, where f is the nonlinear activation function with the sigmoidal function typically being used. $y_j(k)$ is the output of node j with input set k, $w_{ji}(t)$ is the weight between node i and node j, $x_j(k)$ is the output of node j, $\theta$ is a threshold or offset. and N is the number of nodes in the current layer.

Step 4 Adapt Weights

Start by updating the weights between the output layer and the highest hidden layer and recursively work down to the weights between the lowest hidden layer and the input layer. Adjust the weights with the following equation. $w_{ji}(t+1) = w_{ji}(t) + \eta\delta_j x_i'$, with $w_{ji}(t)$ being the weight from hidden node i or from an input node i to node j at time t. $x_i'$ is the output of node i or an input

Figure 3.10 Back-Propagation Learning Algorithm

node i. $\eta$ is the gain term or learning rate. $\delta_j$ is the error term for node j. If node j is an output node, then $\delta_j = y_j(1-y_j)(d_j - y_j)$ with $d(j)$ the desired output at node j and $y_j$ is the calculated output. If node j is a hidden node, then $\delta_j = x'_j(1-x'_j)\sum_k \delta_k w_{jk}$ where k is over all nodes in the layers above node j. Convergence can sometimes be faster if a momentum term is included in the adaptation of the weights. The adaptation of weights including the momentum term is given below.
$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_j x'_i + \alpha(w_{ji}(t) - w_{ji}(t-1))$, where $0 < \alpha < 1$.

Step 5 Repeat by going Step 2.

Continue until the system has converged for the training data.

Figure 3.10 Continued

## Time Delay Neural Network

### Background

The time delay neural network (TDNN) was introduced by Alex Waibel in 1988 [7]. The following reasons are given by Waibel for using the TDNN for speech recognition:

- Multilayer neural networks can represent arbitrary complex decision regions.
- The time delay structure gives the network the ability to learn the temporal structure of speech.
- The network is translation invariant, therefore it has the ability to learn the speech pattern without precise alignment.

The learning procedure used by Waibel and in this research is the back-propagation algorithm. The first recognition task performed by Waibel were on the letters "B", "D", and "G" with average recognition rates between 97.5% and 99.1% for speaker dependent speech samples [7].

## Architecture of the Time Delay Neural Network

This network was used for the pattern classification part of the speech system in this work. An important structure of the time delay neural network is the delay included at each node. Figure 3.11 is a diagram that depicts the delays in the TDNN. The delay node takes the weighted versions of N previous inputs and the current input to calculate the nodes output. This delaying unit gives the time delay neural net the ability to learn speech data without accurate time alignment.

The overall TDNN used by Waibel and this research is shown in Figure 3.12. This TDNN takes 15 sets (150ms) of 16 nodes (i.e. the 16 coefficients generated by the feature extraction) for the input and ends up with an output of 4 nodes (3 nodes in Waibel's case). There are two hidden layers, which contain 8 nodes in the lower hidden layer and 4 nodes in the upper hidden layer. A delay of N=2 between the input and hidden layer 1 leaves hidden layer 1 with 13 sets of eight nodes. The delay of N=4 between hidden layer 1 and hidden layer 2 gives hidden layer 2 9 sets of 4 nodes. Finally the delay between hidden layer 2 and the output layer is N=9 leaving only 1 set of the four output nodes. Increasing the delay size (or window size) when ascending the layers of the TDNN gives the TDNN the ability to learn more abstract relationships between the data.

To calculate the output of a node the following calculations are performed: multiply the value of the lower nodes (and delayed versions) by their corresponding weights, sum these multiplications, and pass the summation through a nonlinear activation function (in this work the sigmoid function). This procedure works for all but the output layer. For the output layer a weighted sum of all node 1's is performed followed by passing the summation through the nonlinear activation function to generate the output for node 1 of the output layer. This is continued for each of the other three nodes. The reason for this final calculation is to include the information for the entire 150ms of data.

## Modifications to the Back-Propagation Algorithm

The structure of the TDNN has many similarities to the perceptron so almost all implementations of the TDNN use a modified back-propagation algorithm. Using a back-prop algorithm is also suggested by Waibel [7]. This research thus follows suit and has developed a modified back-prop algorithm.

The structure of the time delay neural network imposes some problems for the back-propagation algorithm. These problems include the calculation of the output nodes being non standard, the dependence of outputs on shifted versions of inputs, and the different delays or window sizes used for various layers. Therefore there must be modifications made to the back-prop algorithm to overcome these problems. The method used in this work is outlined in Figure 3.13. The two main modifications that were required are incorporating the delayed information and calculating the final output. Since the output from a node is used several times a more complex scheme for keeping track of the nodes is needed. The manner that the final output is calculated requires that the algorithm is modified.

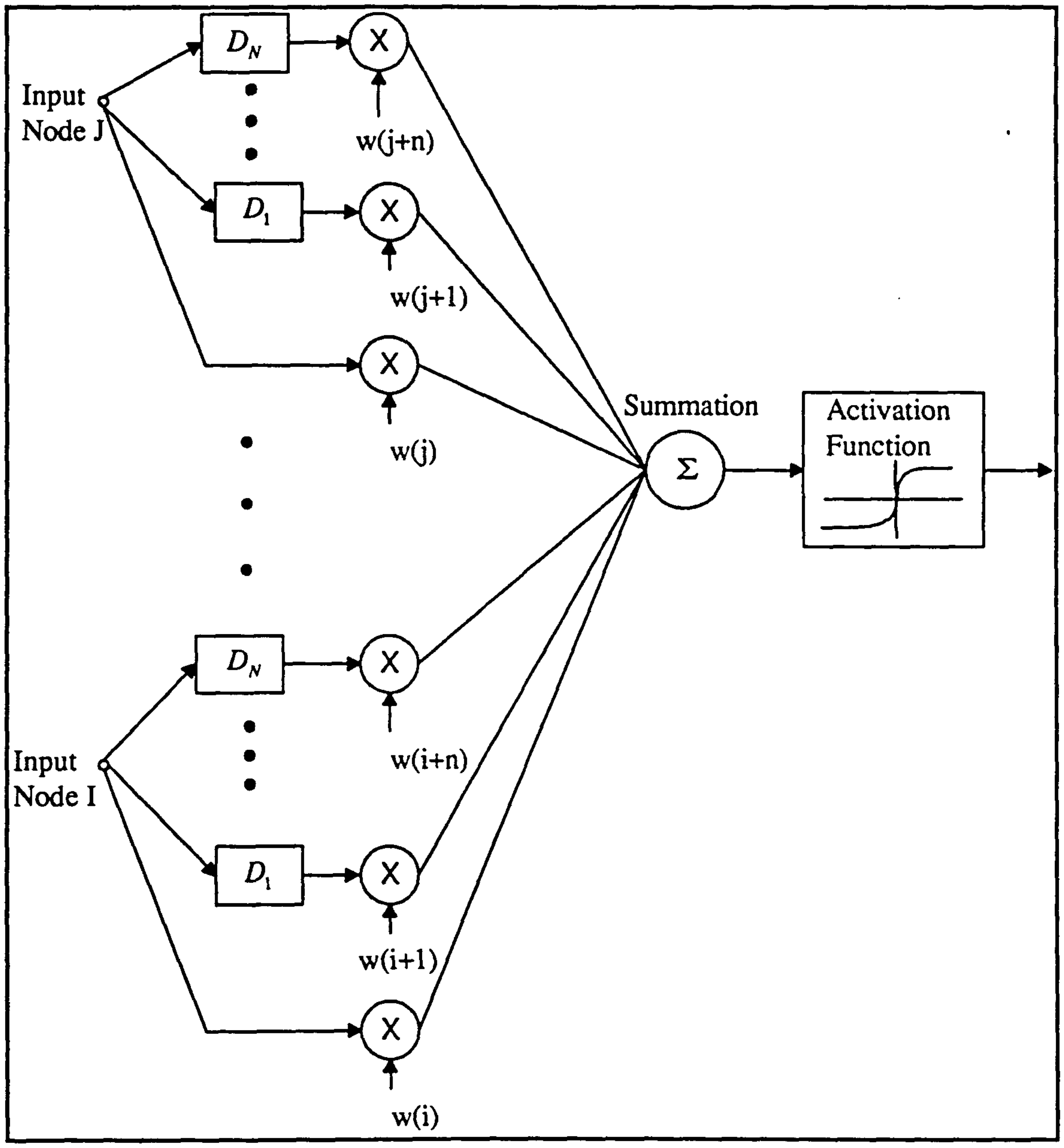


Figure 3.11 Single Delay Unit for the TDNN

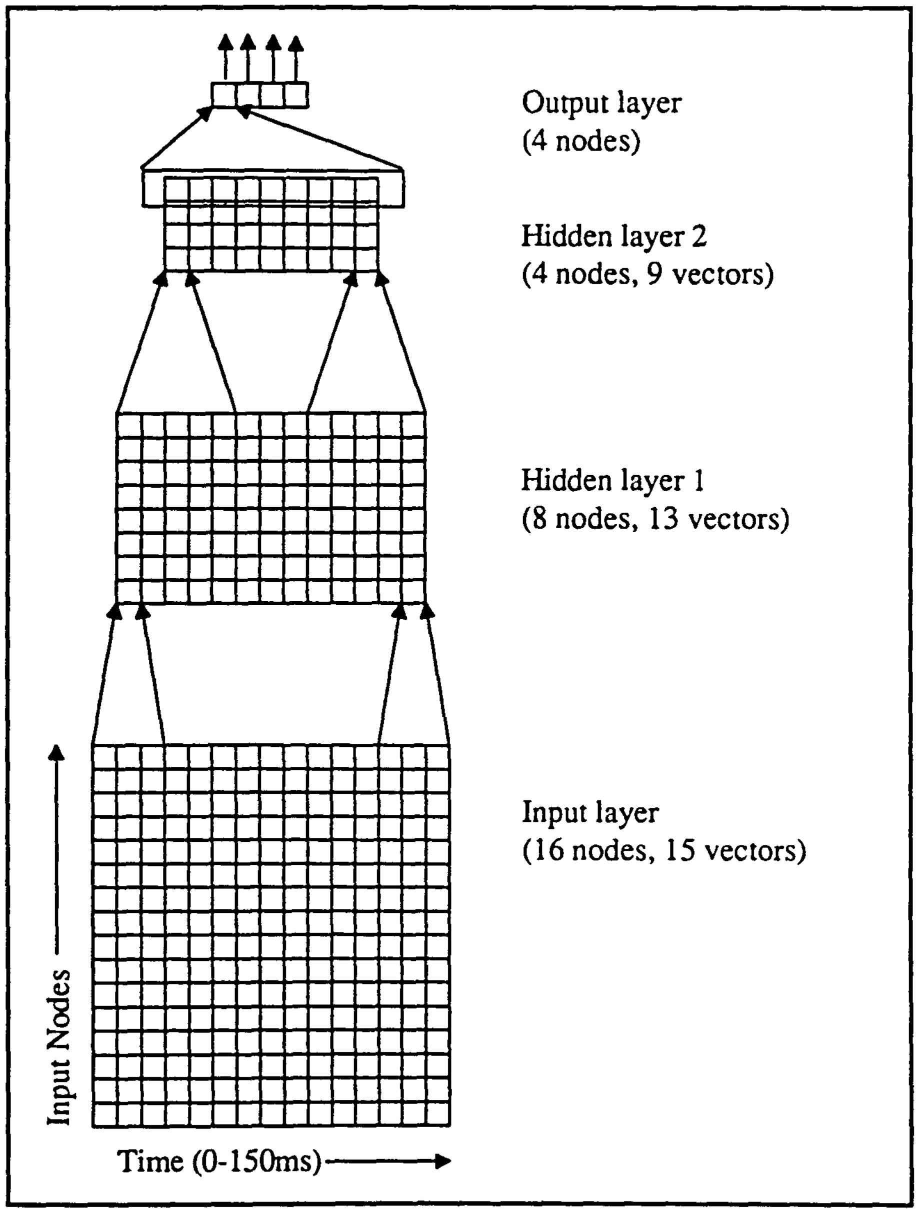


Figure 3.12 Time Delay Neural Network

Time Delay Neural Network Modified Back-Prop Algorithm

Step 1. Initialize Weights

Set the initial weights to random numbers between -0.5 and +0.5.

Step 2. Present Inputs and Desired Outputs

Present a new input set $x_i(j)$, $0 \le i \le N-1$, where j is the index of the data sets. Along with the input specify the desired output $d(j)$. If the network is being used as a classifier then generally the desired outputs are all 0 except for the node corresponding to the desired class which is given the value of 1.

Step 3. Calculate Outputs

a) Calculate the output at hidden layer 1.

$y(i,j) = \sum_{l=1}^{16} \sum_{m=1}^{3} w(i,m,j,l) * x(i+(m-1),l)$, where i is the number of vectors in hidden layer 1, j is the number of nodes in hidden layer 1, m is the size of the delay, $l$ is the number of input nodes, w is the weight between the input layer and hidden layer 1, and x is the inputs from the input layer.

b) Calculate the output at hidden layer 2.

$y(i,j) = \sum_{l=1}^{8} \sum_{m=1}^{5} w(i,m,j,l) * x(i+(m-1),l)$, where i is the number of vectors in hidden layer 2, j is the number of nodes in hidden layer 2, m is the size of the delay, $l$ is the number of nodes in hidden layer 1, w is the weight between hidden layer 1 and hidden layer 2, and x is the output from hidden layer 1.

Figure 3.13 Modified Back-Propagation Algorithm

c) Calculate the output at output layer.

$$y(i) = \sum_{j=1}^{4} \sum_{l=1}^{9} w(l,i,j) * x(l,j), \text{ if } i = j, \text{ where i is the number}$$

of nodes in the output layer, j is the number of nodes in hidden layer 2, $l$ is the size of the delay, w is the weight between hidden layer 2 and the output layer, and x is the output from hidden layer 2.

Step 4. Adapt Weights

a) Adapt weights between output layer and hidden layer 2.

$$delta(j) = y(j) * (1 - y(j)) * (d(j) - y(j))$$
$$w(k,j,i) = w(k,j,i) + \eta * delta(j) * x(k,i)$$
$$+\alpha * (w(k,j,i) - w_{old}(k,j,i))$$

where j is the number of nodes in the output layer, i is the number nodes in hidden layer 2, and k is the delay.

b) Adapt weights between hidden layer 2 and hidden layer 1.

$$delta(k,j) = y(k,j) * (1 - y(k,j)) * \sum_{i=1}^{4} delta(i) * w(k,i,j)$$

where i is the number of nodes in the output layer.

$$w(k,l,j,i) = w(k,l,j,i) + \eta * delta(k,j) * y(k + (l - 1),i)$$
$$+\alpha * (w(k,l,j,i) - w_{old}(k,l,j,i))$$

where k is the number of vectors in hidden layer 2, $l$ is the number of delays, j is the number of nodes in hidden layer 2, and i is the number nodes in hidden layer 1.

Figure 3.13 Continued

$$w(l,k,j,i) = \frac{1}{5}\sum_{k=1}^{5} w(l,k,j,i)$$

c) Adapt weights between hidden layer 1 and input layer.

$$delta(k+(l-1),j) = y(k,j)*(1-y(k,j))+$$
$$\sum_{i=1}^{4} delta(k,i)*w(k,l,i,j)$$

where k is the number of vectors in hidden layer 2, $l$ is the number of delays, i is the number of nodes in hidden layer 2, and j is the number of nodes in hidden layer 1.

$$w(k,l,j,i) = w(k,l,j,i) + \eta * delta(k,j) * x(k+(l-1),i)$$
$$+\alpha * (w(k,l,j,i) - w_{old}(k,l,j,i))$$

where k is the number of vectors in hidden layer 1, j is the number nodes in hidden layer 1, i is the number of nodes in the input layer, and $l$ is the number of delays.

$$w(l,k,j,i) = \frac{1}{3}\sum_{k=1}^{3} w(l,k,j,i)$$

Step 5. Reduce the Size of $\eta$, and $\alpha$

Reduction of $\eta$, and $\alpha$ often helps to stay in a minimum once found.

Step 6. Go to Step 2

Repeat until there is one presentation of all of the data without updating the weights.

Figure 3.13 Continued

# CHAPTER 4. RESULTS

The complete results for this research are given in Table 4.1. The information contained in Table 4.1 is the recognition rates for each of the 4 words, the average recognition rates, and the recognition rates for the training data set. The recognition rates for the training data provides information about how well the training data was learned by the TDNN. Ideally the TDNN should recognize the entire training set without error; the training time for the data was approximately 6 cpu hours and it was decided that the recognition rate achieved was good enough. Since the training set was not learned completely the TDNN can not be expected to recognize untrained words with a higher recognition rate than that of the training set.

Table 4.1 Recognition Rates for 6 Feature Vectors

| Recognition Rates | | | | | | |
|---|---|---|---|---|---|---|
| Feature Vectors | Word Recognition Rate | | | | Overall | Training |
| | /b/ | /d/ | /e/ | /v/ | Recognition | Recognition |
| Mel-Scale FFT | 74.0% | 100.0% | 84.0% | 88.0% | 86.5% | 97.5% |
| Filter Bank FFT | 74.0% | 84.0% | 74.0% | 98.0% | 79.5% | 96.75% |
| Mel-Scale Cepstrum | 86.0% | 96.0% | 96.0% | 98.0% | 94.0% | 98.5% |
| Filter Bank Cepstrum | 83.0% | 94.0% | 92.0% | 93.0% | 90.5% | 98.0% |
| Linear Prediction | 80.0% | 94.0% | 92.0% | 92.0% | 89.5% | 97.75% |
| Wavelet Transform | 88.0% | 94.0% | 96.0% | 96.0% | 93.5% | 98.25% |

It can be seen that the recognition rates for the wavelet transform and the mel-scaled cepstrum are fairly high. The reason for their good performance is their ability to separate the high and low frequency content of the signal. The techniques, other than the mel-scaled cepstrum and wavelet transform, do not supply as much information about the low frequency content of the signals. Since the mel-scale cepstrum and the wavelet transform obtain nearly the same recognition rate it is important to see if there is any advantages of one or the other technique. The wavelet transform has the advantage of a more simple implementation than the mel-scale cepstrum. The wavelet transform could also be implemented in hardware, which would make the processing speed faster than the mel-scale cepstrum.

Notice that the performance of the filter bank cepstrum is not as good as the mel-scaled cepstrum and likewise the performance of the filter bank FFT is not as good as the mel-scaled FFT. Apparently the nonlinear filtering of the cepstrum (or FFT) to generate the coefficients works better than the linear filtering of the cepstrum (or FFT). The main reason that this nonlinear filtering works better than the linear filtering is that the nonlinear filter emphasizes the lower frequencies, which is where most of the distinguishing speech information is located.

The recognition rates for the word /b/ are not as high as the other words. Many times /b/ was recognized as /e/; apparently the distinguishing information in /b/ is not much different than /e/. Most often the other words were not recognized as any of the others when an error occurred. The fact that the wavelet transform and the mel-scale cepstrum both do a better job of recognizing /b/, than the other techniques reemphasizes the fact that these two techniques are the best for speech recognition using the TDNN.

The work was also compared with results obtained by other researchers. This insured that the implementation of the time delay neural network was correct. A paper by Lang and Waibel [7] used the same words, and the mel-scaled cepstrum for the feature vector. The sampling rate used by Lang and Waibel was 12 kHz. The recognition rate obtained for their work was approximately 94%, which compares favorably with the results obtained in this work for the mel-scaled cepstrum feature vector, therefore the implementation of the time delay neural network was assumed to be correct.

Table 4.2 gives the methods of implementation of the different feature vectors. Most of the feature vectors were generated with FORTRAN programs, but the complex cepstrum used a MatLab function. These programs can be found in any speech processing or signal processing book [3,10]. The MatLab function was used for the complex cepstrum because of the difficulty in implementing the complex logarithm. The program for the wavelet transform is included in the appendix, although it could have been just as easily implemented in MatLab.

Table 4.2 Implementation Techniques for Feature Vectors

| Feature Vector | Implementation Technique |
|---|---|
| Mel-Scale FFT | Program |
| Filter Bank FFT | Program |
| Mel-Scale Cepstrum | MatLab |
| Filter Bank Cepstrum | MatLab |
| Linear Prediction | Program |
| Wavelet Transform | Program |

The most interesting and important aspect of this work is the use of the wavelet transform. The wavelet transform has not been applied to speech recognition using neural networks. The fact that the recognition rate of the wavelet transform is so high is very promising for future work. One other item that should be noted is that currently neural networks do not perform the recognition task quite as well as the hidden Markov model. The use of the wavelet transform may give neural networks the ability to perform at a higher recognition rate than the hidden Markov model.

# CHAPTER 5. CONCLUSION AND FUTURE WORK

This work has found that both the popular mel-scaled cepstrum feature vector and a wavelet transform feature vector provide good recognition rates. Both of these feature vectors were capable of producing recognition rates of around 94% for the four phonemes /b/, /d/, /e/, and /v/. Since the wavelet transform has not been used before to generate feature vectors for speech recognition this gives a possible new method for speech recognition.

The next step in this work should be looking at different wavelet transforms and comparing their capabilities for speech recognition. This should be continued by increasing the number of phonemes that are used. Increasing the number of phonemes will allow more realistic speech recognition to be performed. Once the vocabulary is increased the 'good' feature vectors should be compared again to guarantee that they work well for the larger vocabularies. At this point a processor to gather the phonemes to generate words should be developed. The final stage would be creating a language processor to determine the sentence structure.

## REFERENCES

[1] Paolo Antognetti and Veljko Milutinovic, *Neural Networks Concepts, Applications, and Implementations*, Prentice Hall, Englewood Cliffs, New Jersey (1991).

[2] Judith E. Dayhoff, *Neural Network Architectures*, Van Nostrand Reinhold, New York, New York (1990).

[3] John R. Deller, Jr., John G. Proakis, and John H. L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan, New York, New York (1993).

[4] Sadaoki Furui, *Digital Speech Processing, Synthesis, and Recognition*, Marcel Dekker Inc., New York, New York (1989).

[5] Nobuo Hataoka, and Alex H. Waibel, "Speaker-Independent Phoneme Recognition on TIMIT Database Using Integrated Time-Delay neural Networks (TDNNs)," *IEEE ICNN*, 1990.

[6] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, 4-22, April, 1987.

[7] Kevin J. Lang and Alex H. Waibel, "A Time-Delay Neural Network Architecture for Isolated Word Recognition," *Neural Networks*, vol. 3, 23-43, 1990.

[8] Robert J. Mayhan, *Discrete-Time and Continuous-Time Linear Systems*, Addison Wesley, Reading, Massachusetts (1984).

[9] Satru Nakamura, Hidefumi Sawai, and Masahide Suigyama, "Speaker-Independent Phoneme Recognition Using Large-Scale Neural Networks," *ICASSP-92*, 409-412, 1992.

[10] Alan V. Oppenheim, and Ronald W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey (1989).

[11] Thomas Parsons, *Voice and Speech Processing*, McGraw-Hill, New York (1986).

[12] Oliver Rioul and Martin Vetterli, "Wavelets and Signal Processing," *IEEE Signal Processing Magazine*, 14-38, October 1991.

[13] Françoise Fogelman Soulié and Jeanny Hérault, *Neurocomputing - Algorithms, Architectures, and Applications*, Springer-Verlag, New York (1990).

[14] Martin Vetterli and Cormac Herley, "Wavelets and Filter Banks: Theory and Design," *IEEE Trans. on ASSP*, vol 40, 2207-2232, September 1992.

[15] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. on ASSP*, vol 37, 328-339, March 1989.

# APPENDIX

```
*******************************************************************************
*                                                                             *
*     Program TDNN                                                            *
*     Main program for the TDNN                                               *
*     Written by : Brian Schmidt                                              *
*     Date Created : 7/26/93                                                  *
*******************************************************************************

*******************************************************************************
*                                                                             *
*     Subroutines Used :                                                      *
*                       Initialize           (init.f)                         *
*                       Features             (feature.f)                      *
*                       Output               (output.f)                       *
*                       Write_Weights        (weight.f)                       *
*                       Update               (update.f)                       *
*                       Results              (results.f)                      *
*                                                                             *
*     Functions Used :                                                        *
*                       Sigmoid              (sigmoid.f)                      *
*                                                                             *
*******************************************************************************

*******************************************************************************
*                                                                             *
*     Files used :                                                            *
*                       mode.dat             (main.f / input)                 *
*                       feature.dat          (feature.f / input)              *
*                       test.dat             (feature.f / input)              *
*                       weights.dat          (feature.f / input)              *
*                       weights.dat          (weight.f / output)              *
*                       results.dat          (results.f / output)             *
*                                                                             *
*******************************************************************************

***************************************************************************
*                                                                             *
*     Parameters :                                                            *
*                   max_words       maximum number of words to be used        *
*                   max_features    maximum number of features                *
*                   max_vectors     maximum number of feature vectors         *
*                   in_nodes        number of input nodes                     *
*                   hidden1_nodes   number of hidden nodes at layer 1         *
*                   hidden2_nodes   number of hidden nodes at layer 2         *
*                   out_nodes       number of output nodes                    *
*                   delay1          time delay between input and layer 1      *
*                   delay2          time delay between layer 1 and layer 2    *
*                   delay3          time delay between layer 2 and output     *
*                   vec1            number of feature vectors at layer 1      *
*                   vec2            number of feature vecotrs at layer 2                 *
*                                                                                        *
*     For the structure of the TDNN vec2 = delay3, out_nodes =                           *
*                   hidden2_nodes, in_nodes = num_feautres                               *
*                                                                                        *
*     Variables :                                                                        *
*                   num_words       number of words to be used                           *
*                   num_features    number of features to be used                        *
*                   num_vectors     number of feature vectors to be used                 *
*                   training        logical to tell if in training mode                  *
*                   feature         the training data set                                *
*                   test            the classifying data set                             *
*                   yo              calculated outputs for all input words               *
*                   yd              desired outputs for all input words                  *
*                   y1              output at layer 1                                    *
*                   y2              output at layer 2                                    *
*                   y3              output at layer 3 (output layer)                     *
*                   w1              weights from input layer to layer 1                  *
*                   wold1           previous weights from input to layer 1               *
*                   w2              weights from layer 2 to layer 1                      *
*                   wold2           previous weights from layer 2 to 1                   *
*                   w3              weights form layer 3 to layer 2                      *
*                   wold3           previous weights from layer 3 to 2                   *
*                   cont            logical to tell if this is a continuation of a previously started  *
*                                   training period                                      *
*                   eta             learning rate of the back-prop                       *
*                   alpha           learning rate for the momentum term                  *
*                                                                                        *
******************************************************************************************


      Program TDNN

      Integer max_words, max_features, max_vectors, in_nodes, hidden1_nodes
      Integer hidden2_nodes, out_nodes, delay1, delay2, delay3
      Integer vec1, vec2

      Parameter (max_words = 450, max_features = 16, max_vectors = 15)
      Parameter (in_nodes = 16, hidden1_nodes = 8, hidden2_nodes = 4)
      Parameter (out_nodes = 4, delay1 = 3, delay2 = 5, delay3 = 9)
      Parameter (vec1 = 13, vec2 = 9)

      Real feature(max_words,max_vectors,max_features)
      Real test(max_words,max_vectors,max_features)
      Real yo(max_words,out_nodes), yd(max_words,out_nodes)
      Real y1(vec1,hidden1_nodes), y2(vec2,hidden2_nodes), y3(out_nodes)
      Real w1(vec1,delay1,hidden1_nodes,in_nodes)
      Real wold1(vec1,delay1,hidden1_nodes,in_nodes)
      Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real wold2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real w3(delay3,out_nodes,hidden2_nodes)
      Real wold3(delay3,out_nodes,hidden2_nodes), eta, alpha
      Integer num_words, num_features, num_vectors
      Logical training, cont

      Call Initialize(max_words, max_features, max_vectors, in_nodes,
&          hidden1_nodes, hidden2_nodes, out_nodes, delay1, delay2, delay3,
&          vec1, vec2, feature, test, num_words, num_features, num_vectors,
&          training, y1, y2, y3, yd, yo, w1, wold1, w2, wold2, w3, wold3)

      Open (Unit=10, File='mode.dat',Status='Unknown')
      Read (10,*) training
      If (training .EQ. .True.) then
          Read (10,*) cont
      Endif
      Read (10,*) eta, alpha
      Close (10)

      Call Features(max_words, max_features, max_vectors, in_nodes,
&          hidden1_nodes, hidden2_nodes, out_nodes, delay1, delay2,
&          delay3, vec1, vec2, feature, test, num_words, num_features,
&          num_vectors, training, cont, yd, w1, w2, w3)
      write(12,*) 'entering Output'

      Call Output(max_words, max_features, max_vectors, in_nodes,
&          hidden1_nodes, hidden2_nodes, out_nodes, delay1, delay2,
&          delay3, vec1, vec2, feature, test, num_words, num_features,
&          num_vectors, training, cont, y1, y2, y3, yd, yo, w1, wold1, w2,
&          wold2, w3, wold3, eta, alpha)
      write(12,*) 'entering Results'
      Call Results(max_words, out_nodes, training, yd, yo, num_words, eta
&          alpha, cont)
      close(12)
      close(14)
      End
```

```
****************************************************************************
*                                                                          *
*     Subroutine Initialize                                                *
*     Subroutine to initialize all of the variables                        *
*     Written by : Brian Schmidt                                           *
*     Date Created : 7/26/93                                               *
*                                                                          *
****************************************************************************

****************************************************************************
*                                                                          *
*     Parameters :                                                         *
*                   max_words       50                                     *
*                   max_features    20                                     *
*                   max_vectors     20                                     *
*                   in_nodes        16                                     *
*                   hidden1_nodes   8                                      *
*                   hidden2_nodes   4                                      *
*                   out_nodes       4                                      *
*                   delay1          3                                      *
*                   delay2          5                                      *
*                   delay3          9                                      *
*                   vec1            13                                     *
*                   vec2            9                                      *
*                                                                          *
*     Variables :                                                          *
*                   num_words       number of words to be used             *
*                   num_features    number of features to be used          *
*                   num_vectors     number of feature vectors to be used   *
*                   training        logical to tell if in training mode    *
*                   feature         the training data set                  *
*                   test            the classifying data set               *
*                   yo              calculated outputs for all input words *
*                   yd              desired outputs for all input words    *
*                   y1              output at layer 1                      *
*                   y2              output at layer 2                      *
*                   y3              output at layer 3 (output layer)       *
*                   w1              weights from input layer to layer 1    *
*                   wold1           previous weights from input to layer 1 *
*                   w2              weights from layer 1 to layer 2        *
*                   wold2           previous weights from layer 2 to 1     *
*                   w3              weights from layer 3 to layer 2        *
*                   wold3           previous weights from layer 3 to 2     *
*                   i               loop index                             *
*                   j               loop index                             *
*                   k               loop index                             *
*                                                                          *
****************************************************************************

      Subroutine Initialize(max_words, max_features, max_vectors, in_nodes,
     &       hidden1_nodes, hidden2_nodes, out_nodes, delay1, delay2, delay3,
    &      vec1, vec2, feature, test, num_words, num_features, num_vectors,
    &      training, y1, y2, y3, yd, yo, w1, wold1, w2, wold2, w3, wold3)

       Integer num_words, num_features, num_vectors, max_words, max_features
       Integer max_vectors, in_nodes, hidden1_nodes, hidden2_nodes, out_nodes
       Integer delay1, delay2, delay3, vec1, vec2
       Integer i, j, k
       Real y1(vec1,hidden1_nodes), yo(max_words,out_nodes)
       Real y2(vec2,hidden2_nodes), y3(out_nodes)
       Real yd(max_words,out_nodes), w1(vec1,delay1,hidden1_nodes,in_nodes)
       Real wold1(vec1,delay1,hidden1_nodes,in_nodes)
       Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
       Real wold2(vec2,delay2,hidden2_nodes,hidden1_nodes)
       Real w3(delay3,out_nodes,hidden2_nodes)
       Real wold3(delay3,out_nodes,hidden2_nodes)
       Real feature(max_words,max_vectors,max_features)
       Real test(max_words,max_vectors,max_features)
       Logical training

       num_words = 0
       num_features = 0
       num_vectors = 0
       training = .True.

       Do 100 i = 1, max_words
          Do 200 j = 1, max_vectors
             Do 300 k = 1, max_features
                feature(i,j,k) = 0.0
                test(i,j,k) = 0.0
300          Continue
200       Continue
100    Continue

       Do 350 l = 1, vec1
          Do 400 i = 1, delay1
             Do 500 j = 1, hidden1_nodes
                Do 600 k = 1, in_nodes
                   w1(l,i,j,k) = (Ran(1)-0.5)
                   wold1(l,i,j,k) = 0.0
600             Continue
500          Continue
400       Continue
350    Continue

       Do 650 l = 1, vec2
          Do 700 i = 1, delay2
             Do 800 j = 1, hidden2_nodes
                Do 900 k = 1, hidden1_nodes
                   w2(l,i,j,k) = (Ran(1)-0.5)
                   wold2(l,i,j,k) = 0.0
900             Continue
```

```
800             Continue
700         Continue
650     Continue

        Do 1000 i = 1, delay3
            Do 1100 j = 1, out_nodes
                Do 1200 k = 1, hidden2_nodes
                    If (i .EQ. j) Then
                        w3(i,j,k) = (Ran(1)-0.5)
                    Else
                        w3(i,j,k) = 0.0
                    Endif
                    wold3(i,j,k) = 0.0
1200            Continue
1100        Continue
1000    Continue

        Do 1300 i = 1, vec1
            Do 1400 j = 1, hidden1_nodes
                y1(i,j) = 0.0
1400        Continue
1300    Continue

        Do 1500 i = 1, vec2
            Do 1600 j = 1, hidden2_nodes
                y2(i,j) = 0.0
1600        Continue
1500    Continue

        Do 1700 i = 1, out_nodes
            y3(i) = 0.0
1700    Continue

        Do 1800 i = 1, max_words
            Do 1900 j = 1, out_nodes
                yd(i,j) = 0.0
                yo(i,j) = 0.0
1900        Continue
1800    Continue

        Return

        End
```

```
**********************************************************************************
*                                                                                *
*     Subroutine Features                                                        *
*     Subroutine to read the feature vectors                                     *
*     Written by : Brian Schmidt                                                 *
*     Date Created : 7/26/93                                                     *
*                                                                                *
**********************************************************************************

**********************************************************************************
*                                                                                *
*     Parameters :                                                               *
*                    max_words        50                                         *
*                    max_features     20                                         *
*                    max_vectors      20                                         *
*                    in_nodes         16                                         *
*                    hidden1_nodes    8                                          *
*                    hidden2_nodes    4                                          *
*                    out_nodes        4                                          *
*                    delay1           3                                          *
*                    delay2           5                                          *
*                    delay3           9                                          *
*                    vec1             13                                         *
*                    vec2             9                                          *
*                                                                                *
*     Varialbes :                                                                *
*                    num_words        number of words to be used                 *
*                    num_features     number of features to be used              *
*                    num_vectors      number of feature vectors to be used       *
*                    training         training data set                          *
*                    test             classifying data set                       *
*                    yo               calculated outputs                         *
*                    yd               desired outputs for all input words        *
*                    y1               output at layer1                           *
*                    y2               output at layer2                           *
*                    y3               output at layer3 (output layer)            *
*                    w1               weights from input layer to layer 1        *
*                    wold1            previous weights from input to layer 1     *
*                    w2               weights from layer2 to layer 1             *
*                    wold2            previous weights from layer2 to layer1     *
*                    w3               weights from layer3 to layer2              *
*                    wold3            previous weights from layer3 to layer2     *
*                    i                loop index                                 *
*                    j                loop index                                 *
*                    k                loop index                                 *
*                    l                loop index                                 *
*                                                                                *
**********************************************************************************

      Subroutine Features(max_words, max_features, max_vectors,in_nodes,
     &      hidden1_nodes, hidden2_nodes, out_nodes,
  &      delay1, delay2, delay3, vec1, vec2, feature, test, num_words,
  &      num_features, num_vectors, training, cont, yd, w1, w2, w3)

      Integer max_words, max_features, max_vectors, in_nodes, hidden1_nodes
      Integer delay1, delay2, delay3, vec1, vec2, hidden2_nodes, out_nodes
      Real feature(max_words,max_vectors,max_features)
      Real yd(max_words,out_nodes)
      Real test(max_words,max_vectors,max_features)
      Real w1(vec1,delay1,hidden1_nodes,in_nodes)
      Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real w3(delay3,out_nodes,hidden2_nodes)
      Integer num_words, num_features, num_vectors
      Integer i, j, k, l
      Logical training, cont

      If (training .EQ. .True.) Then

         Open (Unit=10, File='feature.dat',Status='Unknown')

         Read(10,*) num_words
         Read(10,*) num_features
         Read(10,*) num_vectors

         Do 50 k = 1, num_words
            Read(10,*) (yd(k,l), l=1, out_nodes)
            Do 100 i = 1,num_vectors
               Do 200 j = 1, num_features
                  Read(10,*) feature(k,i,j)
200            Continue
100         Continue
50       Continue

         Close(10)

      Else

         Open (Unit=10,File='test.dat',Status='unknown')

         Read(10,*) num_words
         Read(10,*) num_features
         Read(10,*) num_vectors

         Do 300 k = 1, num_words
            Do 400 i = 1, num_vectors
               Do 500 j = 1, num_features
                  Read (10,*) test(k,i,j)
500            Continue
400         Continue
300      Continue

         Close (10)
      Endif

      If ((cont .EQ. .True.) .OR. (training .EQ. .False.)) Then
         Open (Unit=10,File='weights.dat',Status='Unknown')

         Do 550 l = 1, vec1
            Do 600 k = 1, delay1
               Do 700 j = 1, hidden1_nodes
                  Do 800 i = 1, in_nodes
                     Read(10,*)  w1(l,k,j,i)
800               Continue
700            Continue
600         Continue
550      Continue

         Do 850 l = 1, vec2
            Do 900 k = 1, delay2
               Do 1000 j = 1, hidden2_nodes
                  Do 1100 i = 1, hidden1_nodes
                     Read(10,*) w2(l,k,j,i)
1100              Continue
1000           Continue
900         Continue
850      Continue

         Do 1200 k = 1, delay3
            Do 1300 j = 1, out_nodes
               Do 1400 i = 1, hidden2_nodes
                  Read(10,*) w3(k,j,i)
1400           Continue
1300        Continue
1200     Continue

         Close(10)

      Endif

      Return

      End
```

```
**********************************************************************************
*                                                                                *
*     Subroutine Output                                                          *
*     Subroutine to calculate outputs at hidden and output nodes                 *
*     Written by : Brian Schmidt                                                 *
*     Date Created : 7/27/93                                                     *
*                                                                                *
**********************************************************************************

**********************************************************************************
*                                                                                *
*     Parameters :                                                               *
*                   max_words       50                                           *
*                   max_features    20                                           *
*                   max_vectors     20                                           *
*                   in_nodes        16                                           *
*                   hidden1_nodes   8                                            *
*                   hidden2_nodes   4                                            *
*                   out_nodes       4                                            *
*                   delay1          3                                            *
*                   delay2          5                                            *
*                   delay3          9                                            *
*                   vec1            13                                           *
*                   vec2            9                                            *
*                                                                                *
*     Varialbes :                                                                *
*                   num_words       number of words to be used                   *
*                   num_features    number of features to be used                *
*                   num_vectors     number of feature vectors to be used         *
*                   training        logical to tell if in training mode          *
*                   feature         training data set                            *
*                   test            classifying data set                         *
*                   yo              calculated outputs for all input words       *
*                   yd              desired outputs for all input words          *
*                   y1              output at layer 1                            *
*                   y2              output at layer 2                            *
*                   y3              output at layer 3                            *
*                   w1              weigths from layer1 to input layer           *
*                   wold1           previous weights from layer1 to input        *
*                   w2              weights from layer2 to layer1                *
*                   wold2           previous weights from layer2 to layer1       *
*                   w3              weights form output layer to layer 2         *
*                   wold3           previous weights from output to layer2       *
*                   i               loop index                                   *
*                   j               loop index                                   *
*                   k               loop index                                   *
*                   l               loop index                                   *
*                   m               loop index                                   *
*                   done            logical to tell if updating is complete      *
*                   up              logical to tell if weights need updating     *
*                   eta             learning rate                                *
*                    alpha              learning rate                                     *
*                                                                                         *
*******************************************************************************************

      Subroutine Output (max_words, max_features, max_vectors, in_nodes,
   &      hidden1_nodes, hidden2_nodes, out_nodes, delay1, delay2, delay3,
   &      vec1, vec2, feature, test, num_words, num_features, num_vectors,
   &      training, cont, y1, y2, y3, yd, yo, w1, wold1, w2, wold2, w3,
   &      wold3, eta, alpha)

      Integer max_words, max_features, max_vectors, delay1, delay2, delay3
      Integer vec1, vec2, in_nodes, hidden1_nodes, hidden2_nodes, out_nodes
      Real w1(vec1,delay1,hidden1_nodes,in_nodes)
      Real wold1(vec1,delay1,hidden1_nodes,in_nodes)
      Real y1(vec1,hidden1_nodes)
      Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real wold2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real y2(vec2,hidden2_nodes)
      Real w3(delay3,out_nodes,hidden2_nodes), y3(out_nodes)
      Real wold3(delay3,out_nodes,hidden2_nodes)
      Real yd(max_words,out_nodes), yo(max_words,out_nodes)
      Real feature(max_words,max_vectors,max_features)
      Real test(max_words,max_vectors,max_features)
      Real eta, alpha
      Integer i, j, k, l, m, num_words, num_features, num_vectors
      Integer count, max_count
      Logical training, done, up, cont

      If (training .EQ. .True.) Then
          Write(*,*) 'What is the maximum number of training iterations?'
          Read(*,*) max_count
      Endif

      done = .False.
      count = 0
      Do 25 While ((done .NE. .True.) .AND. (training .EQ. .True.))

          done = .True.

*     outputs at hidden layer 1
          count = count + 1
          Do 50 k = 1, num_words
              Do 100 i = 1, vec1
                  Do 200 j = 1, hidden1_nodes
                      y1(i,j) = 0.0
                      Do 300 l = 1, num_features
                          Do 400 m = 1, delay1
                              y1(i,j) = w1(i,m,j,l) * feature(k,i+(m-1),l) + y1(i,j)
400                       Continue
300                   Continue
                      y1(i,j) = sigmoid(y1(i,j))
```

```
200                 Continue
100             Continue

*       outputs a hidden layer 2

                Do 600 i = 1, vec2
                    Do 700 j = 1, hidden2_nodes
                        y2(i,j) = 0.0
                        Do 750 l = 1, hidden1_nodes
                            Do 800 m = 1, delay2
                                y2(i,j) = w2(i,m,j,l)*y1(i+(m-1),l) + y2(i,j)
800                         Continue
750                     Continue
                        y2(i,j) = sigmoid(y2(i,j))
700                 Continue
600             Continue

*       outputs at output layer

                Do 1000 i = 1, out_nodes
                    y3(i) = 0.0
                    Do 1100 j = 1, hidden2_nodes
                        Do 1200 l = 1, delay3
                            If (i .EQ. j) Then
                                y3(i) = w3(l,i,j) * y2(l,j) + y3(i)
                            Endif
1200                    Continue
1100                Continue
                    y3(i) = sigmoid(y3(i))
1000            Continue

*       check to see if weights need to be updated during training mode

                up = .False.
                Do 1300 i = 1, out_nodes
                    If ((yd(k,i) .LE. (y3(i)-.15)).AND.(yd(k,i) .LE. 0.00001)) then
                        up = .True.
                    Endif
                    If ((yd(k,i) .GE. (y3(i)+.15)).AND.(yd(k,i) .GE. 0.99999)) then
                        up = .True.
                    Endif
1300            Continue

                If (up .EQ. .True.) Then
                    Call Update(max_words, in_nodes, hidden1_nodes, hidden2_nodes,
  &                     out_nodes, delay1, delay2, delay3, vec1, vec2, w1,
  &                     wold1, w2, wold2, w3, wold3, y1, y2, y3, yd, k, eta,
  &                     alpha, feature, max_vectors, max_features, count)
                    done = .False.
                Endif
50          Continue
```

```
*reducing the size of eta and alpha

            eta = eta - 0.001 * eta
            alpha = alpha - 0.001 * alpha

            If (count .ge. max_count) Then
                done = .true.
            Endif

25      Continue

        If (training .EQ. .True.) Then
            Call Write_weights(in_nodes, hidden1_nodes, hidden2_nodes,
   &            out_nodes,delay1, delay2, delay3, w1, w2, w3, vec1, vec2)
            Open (Unit=10, File='mode.dat', Status='unknown')
            Write(10,*) training
            Write(10,*) cont
            Write(10,*) eta
            Write(10,*) alpha
            Close(10)
        Endif

*Calculating final outputs for the training or classifying data sets

        Do 1400 k = 1, num_words

*Calulating outputs at layer 1

            Do 1500 i = 1, vec1
                Do 1600 j = 1, hidden1_nodes
                    y1(i,j) = 0.0
                    Do 1700 l = 1, num_features
                        Do 1800 m = 1, delay1
                            If (training .EQ. .True.) Then
                                y1(i,j) = w1(i,m,j,l) * feature(k,i+(m-1),l) + y1(i,j)
                            Else
                                y1(i,j) = w1(i,m,j,l) * test(k,i+(m-1),l) + y1(i,j)
                            Endif
1800                    Continue
1700                Continue
                    y1(i,j) = sigmoid(y1(i,j))
1600            Continue
1500        Continue

*Calculating outputs at layer 2

            Do 1900 i = 1, vec2
                Do 2000 j = 1, hidden2_nodes
                    y2(i,j) = 0.0
                    Do 2100 l = 1, hidden1_nodes
                Do 2200 m = 1, delay2
                    y2(i,j) = w2(i,m,j,l) * y1(i+(m-1),l) + y2(i,j)
2200            Continue
2100        Continue
            y2(i,j) = sigmoid(y2(i,j))
2000      Continue
1900    Continue

*Calculating outputs at output layer

        Do 2300 i = 1, out_nodes
          yo(k,i) = 0.0
          Do 2400 j = 1, hidden2_nodes
            Do 2500 l = 1, delay3
                If (i .EQ. j) Then
                    yo(k,i) = w3(l,i,j) * y2(l,j) + yo(k,i)
                Endif
2500        Continue
2400      Continue
          yo(k,i) = sigmoid(yo(k,i))
2300    Continue
1400  Continue

      Return

      End
```

```
**************************************************************************************
*                                                                                    *
*     Subroutine Update                                                              *
*     Subroutine to update weights using the back propagation learning method, including momentum term *
*     Written by : Brian Schmidt                                                     *
*     Date Created : 7/27/93                                                         *
*                                                                                    *
**************************************************************************************

**************************************************************************************
*                                                                                    *
*     Parameters :                                                                   *
*                    max_words       50                                              *
*                    in_nodes        16                                              *
*                    hidden1_nodes   8                                               *
*                    hidden2_nodes   3                                               *
*                    out_nodes       3                                               *
*                    delay1          3                                               *
*                    delay2          5                                               *
*                    delay3          9                                               *
*                    vec1            13                                              *
*                    vec2            9                                               *
*                                                                                    *
*     Varialbes :                                                                    *
*                    cur_word        current word being processed                    *
*                    yd              desired outputs for all input words             *
*                    y1              output at layer 1                               *
*                    y2              output at layer 2                               *
*                    y3              output at layer 3                               *
*                    w1              weigths from layer1 to input layer              *
*                    wold1           previous weights from layer1 to input           *
*                    w2              weights from layer2 to layer1                   *
*                    wold2           previous weights from layer2 to layer1          *
*                    w3              weights form output layer to layer 2            *
*                    wold3           previous weights from output to layer2          *
*                    i               loop index                                      *
*                    j               loop index                                      *
*                    k               loop index                                      *
*                    l               loop index                                      *
*                    eta             learning rate                                   *
*                    alpha           learning rate                                   *
*                    new             temporary variable for the new weight           *
*                                                                                    *
**************************************************************************************

      Subroutine Update(max_words, in_nodes, hidden1_nodes, hidden2_nodes,
  &       out_nodes, delay1, delay2, delay3, vec1, vec2, w1, wold1, w2,
  &       wold2, w3, wold3, y1, y2, y3, yd, cur_word, eta, alpha, feature,
  &       max_vectors, max_features, count)

      Integer max_words, max_vectors, max_features, count
      Integer in_nodes, hidden1_nodes, hidden2_nodes, out_nodes
      Integer delay1, delay2, delay3, vec1, vec2, cur_word
      Real w1(vec1,delay1,hidden1_nodes,in_nodes)
      Real wold1(vec1,delay1,hidden1_nodes,in_nodes)
      Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real wold2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real w3(delay3,out_nodes,hidden2_nodes)
      Real wold3(delay3,out_nodes,hidden2_nodes)
      Real y1(vec1,hidden1_nodes), y2(vec2,hidden2_nodes)
      Real y3(out_nodes), yd(max_words,out_nodes)
      Real del1(13,8), del2(9,4), feature(max_words,max_vectors,max_features)
      Real del3(4), eta, alpha, temp, new
      Integer i, j, k, l, m

*     Calulations for the weights between output layer and layer 2

      Do 100 j = 1, out_nodes
         del3(j) = y3(j) * (1 - y3(j)) * (yd(cur_word,j) - y3(j))
100   Continue

      Do 200 j = 1, out_nodes
         Do 300 i = 1, hidden2_nodes
            If (j .EQ. i) Then
               Do 400 k = 1, delay3
                  new = w3(k,j,i) + eta * del3(j) * y2(k,i) +
     &               alpha * (w3(k,j,i) - wold3(k,j,i))
                  wold3(k,j,i) = w3(k,j,i)
                  w3(k,j,i) = new
400            Continue
            Endif
300      Continue
200   Continue

*     updating weights between layer 2 and layer 1

      Do 800 k = 1, vec2
         Do 900 j= 1, hidden2_nodes
            temp = 0.0
            Do 1000 i = 1, out_nodes
               If (i .EQ. j) Then
                  temp = del3(i) * w3(k,i,j) + temp
               Endif
1000        Continue
            del2(k,j) = temp * (y2(k,j)) * (1 - y2(k,j))
900      Continue
800   Continue

      Do 1200 k = 1, vec2
         Do 1300 j = 1, hidden2_nodes
            Do 1400 i = 1, hidden1_nodes
               Do 1500 l = 1, delay2
```

```
                        new = w2(k,l,j,i) + eta * del2(k,j) * y1(k+(l-1),i) +
     &                       alpha * (w2(k,l,j,i) - wold2(k,l,j,i))
                        wold2(k,l,j,i) = w2(k,l,j,i)
                        w2(k,l,j,i) = new
1500                Continue
1400            Continue
1300        Continue
1200    Continue

        Do 1600 j = 1, hidden2_nodes
            Do 1650 i = 1, hidden1_nodes
                Do 1700 l = 1, vec2
                    new = 0.0
                    Do 1750 k = 1, delay2
                        new = w2(l,k,j,i) + new
1750                Continue
                    Do 1800 k = 1, delay2
                        w2(l,k,j,i) = new/delay2
1800                Continue
1700            Continue
1650        Continue
1600    Continue

*       updating weights between layer 1 and input layer

        Do 1840 k = 1, vec1
            Do 1860 j = 1, hidden1_nodes
                del1(k,j) = 0.0
1860        Continue
1840    Continue

        Do 1900 k = 1, vec2
            Do 1950 l = 1, delay2
                Do 2000 j = 1, hidden1_nodes
                    temp = 0.0
                    Do 2100 i = 1, hidden2_nodes
                        temp = del2(k,i) * w2(k,l,i,j) + temp
2100                Continue
                    del1(k+(l-1),j) = temp * (y1(k,j)) * (1 - y1(k,j)) + del1(k+(l-1),j)
2000            Continue
1950        Continue
1900    Continue

        Do 2300 k = 1, vec1
            Do 2400 j = 1, hidden1_nodes
                Do 2500 i = 1, in_nodes
                    Do 2600 l = 1, delay1
                        new = w1(k,l,j,i) + eta * del1(k,j) *
     &                  feature(cur_word,k+(l-1),i) + alpha * (w1(k,l,j,i) -
     &                  wold1(k,l,j,i))
                        wold1(k,l,j,i) = w1(k,l,j,i)
                        w1(k,l,j,i) = new
2600                Continue
2500            Continue
2400        Continue
2300    Continue

        Do 2700 j = 1, hidden1_nodes
            Do 2800 i = 1, in_nodes
                Do 2900 l = 1, vec1
                    new = 0.0
                    Do 3000 k = 1, delay1
                        new = w1(l,k,j,i) + new
3000                Continue
                    Do 3100 k = 1, delay1
                        w1(l,k,j,i) = new/delay1
3100                Continue
2900            Continue
2800        Continue
2700    Continue

        Return

        End
```

```
***************************************************************************************
*                                                                                     *
*     Subroutine Write_weights                                                        *
*     Subroutine to write the trained weight to the file weights.dat                  *
*     Written by : Brian Schmidt                                                      *
*     Date Created : 7/27/93                                                          *
*                                                                                     *
***************************************************************************************

***************************************************************************************
*                                                                                     *
*     Parameters :                                                                    *
*                   in_nodes        16                                                *
*                   hidden1_nodes   8                                                 *
*                   hidden2_nodes   4                                                 *
*                   out_nodes       4                                                 *
*                   delay1          3                                                 *
*                   delay2          5                                                 *
*                   delay3          9                                                 *
*                                                                                     *
*     Variables :                                                                     *
*                   w1              weights from layer1 to input layer                *
*                   w2              weights form layer2 to layer1                     *
*                   w3              weights form layer3 (output) to layer2            *
*                   i               loop index                                        *
*                   j               loop index                                        *
*                   k               loop index                                        *
*                   l               loop index                                        *
*                                                                                     *
***************************************************************************************

      Subroutine Write_weights(in_nodes, hidden1_nodes, hidden2_nodes,
     &  out_nodes, delay1, delay2, delay3, w1, w2, w3, vec1, vec2)

      Integer in_nodes, hidden1_nodes, hidden2_nodes, out_nodes, delay1
      Integer delay2, delay3, vec1, vec2
      Real w1(vec1,delay1,hidden1_nodes,in_nodes)
      Real w2(vec2,delay2,hidden2_nodes,hidden1_nodes)
      Real w3(delay3,out_nodes,hidden2_nodes)
      Integer i, j, k, l

      Open(Unit=10,File='weights.dat',Status='Unknown')

      Do 50 l = 1, vec1
          Do 100 k = 1, delay1
              Do 200 i = 1, hidden1_nodes
                  Do 300 j = 1, in_nodes
                      Write(10,*) w1(l,k,i,j)
300               Continue
200           Continue
100       Continue
```

```
50    Continue

      Do 350 l = 1, vec2
         Do 400 k = 1, delay2
            Do 500 i = 1, hidden2_nodes
               Do 600 j = 1, hidden1_nodes
                  Write(10,*) w2(l,k,i,j)
600            Continue
500         Continue
400      Continue
350   Continue

      Do 700 k = 1, delay3
         Do 800 i = 1, out_nodes
            Do 900 j = 1, hidden2_nodes
               Write(10,*) w3(k,i,j)
900         Continue
800      Continue
700   Continue

      Close(10)

      Return

      End
```

```
***************************************************************************
*                                                                         *
*     Subroutine Results                                                  *
*     Subroutine to write the results of the TDNN to an output file       *
*     Written by : Brian Schmidt                                          *
*     Date Created : 7/27/93                                              *
*                                                                         *
***************************************************************************

***************************************************************************
*                                                                         *
*     Parameters :                                                        *
*                 max_words      50                                       *
*                 out_nodes      4                                        *
*                                                                         *
*     Variables :                                                         *
*                 training       logical to tell if in training mode      *
*                 yd             desired output                           *
*                 yo             calculated output                        *
*                 k              loop index                               *
*                 num_words      number of words                          *
*                                                                         *
***************************************************************************

      Subroutine Results(max_words, out_nodes, training, yd, yo, num_words)

      Integer max_words, out_nodes, num_words, i, j, k
      Real yd(max_words,out_nodes)
      Real yo(max_words,out_nodes)
      Logical training

      Open(Unit=10,File='results.dat',Status='Unknown')

      If (training .EQ. .True.) then
         Write (10,*) 'Training Mode'
         Write (10,*) 'Desired Output Calculated Output'
         Do 100 k = 1 , num_words
            Write(10,*) (yd(k,i), i=1, out_nodes), (yo(k,j), j=1, out_nodes)
100      Continue

      Else

         Write (10,*) 'Classifying Mode'
         Write (10,*) 'Calculated Output'
         Do 200 k = 1, num_words
            Write(10,*) (yo(k,j), j = 1, out_nodes)
200      Continue
      Endif

      Return
      End
```

```
*****************************************************************************************
*                                                                                       *
*       Function Sigmoid                                                                *
*       Function to calculate the sigmoid of a given number                             *
*       Written by : Brian Schmidt                                                      *
*       Date Created : 7/27/93                                                          *
*                                                                                       *
*****************************************************************************************

*****************************************************************************************
*                                                                                       *
*       Variables :                                                                     *
*                       x                       number to take the sigmoid of           *
*                                                                                       *
*****************************************************************************************

        Real Function Sigmoid (x)

        Real x

        If (x .lt. -18.0) then
            Sigmoid = 0.0
        Else If (x .gt. 18.0) then
            Sigmoid = 1.0
        Else
            Sigmoid = 1/(1+exp(-x))
        Endif

        Return

        End
```

```
************************************************************************************
*                                                                                  *
*     DAUB4 implementation of 4-coefficient Daubechies Wavelet                     *
*                                                                                  *
************************************************************************************

      Subroutine Daub4(a, n, isign)
      Integer n, isign, NMAX
      Parameter (C0=0.4829629131445341, C1=0.8365163037378079, C2=0.2241438680420134,
                  C3=-0.1294095225512604, NMAX=1024)
      Real wksp(NMAX)
      Integer nh, hn1,i, j

      If (n .LT. 4) Return
      If (n .GT. NMAX) pause 'wksp too small in daub4'
      nh = n/2
      nh1 = nh + 1
      If (isign .GE. 0) Then
            i = 1
            Do j = 1, n-3, 2
                  wksp(i) = C0*a(j)+C1*a(j+1)+C2*a(j+2)+C3*a(j+3)
                  wksp(i+nh) = C3*a(j)-C2*a(j+1)+C1*a(j+2)-C0*a(j+3)
                  i = i+1
            EndDo
            wksp(i) = C0*a(n-1)+C1*a(n)+C2*a(1)+C3*a(2)
            wksp(i+nh) = C3*a(n-1)-C2*a(n)+C1*a(1)-C0*a(2)
      Else
            wksp(1) = C2*a(nh)+c1*a(n)+C0*a(1)+C3*a(nh1)
            wksp(2) = C3*a(nh)-C0*a(n)+C1*a(1)-C2*a(nh1)
            j = 3
            Do i = 1, nh-1
                  wksp(j) = C2*a(i)+C1*a(i+nh)+C0*a(i+1)+C3*a(i+nh1)
                  wksp(j+1) = C3*a(i)-C0*a(i+nh)+C1*a(i+1)-C2*a(i+nh1)
                  j = j+2
            EndDo
      Endif

      Do i = 1, n
            a(i) = wksp(i)
      EndDo

      Return

      End
```