VICTORIA UNIVERSITY OF WELLINGTON

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Speech Recognition by Machine

Andrew Kingston

Technical Report CS-TR-92/2
October 1992

Abstract

Two popular approaches to isolated word speech recognition include the template matching and the neural network approach. An implementation of each of these systems is presented in this report, and a comparison between the two is given.

The recognition task is limited to digits only, and experiments involving a single speaker and a more difficult multiple speaker task were performed. In constructing the input for the recognition systems, various digital signal processing routines that transform the raw speech signal into a comparable word template format are examined. Word boundary detection is investigated, and an algorithm was developed to automatically separate each digit from the collected speech samples. A "dynamic time warping" procedure provides the template matching system with the ability to time align two templates, in contrast to the "temporal flow" neural network model, which contains a network of delay links to represent temporal relationships.

Recognition rates of 100% are reported for both approaches using a single speaker, but the neural network model achieves consistently higher rates for the multiple speakers obtaining a highest recognition rate of 92.5%.

Publishing Information

This report was originally prepared as part of B.Sc.(Hons) Degree in Computer Science under the supervision of Steve Cassidy (1991).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

"Whilst we are accustomed to interacting with machines by means of knobs and keys, we interact with each other mainly by speech. The question arises, therefore, as to whether it would be more natural to employ speech as a means of interacting with machines." [1]

Speech is potentially the fastest form of human to machine communication as speaking rates are still comparable to the fastest typing rates. There are also many advantages to speech as a form of communication with machines: people learn and are familiar with speech from an early age, people can naturally think and speak simultaneously, hands and eyes become free for other activities.

Speech recognition has been an active area of investigation for many years; the goal of recognising natural, connected speech from multiple speakers has not yet been achieved but significant progress has been made.

A number of different classes of speech recognition algorithms can be identified. *Template Matching* algorithms attempt to store an idealised speech template for each word and match an incoming speech signal of a word with the stored templates; the template with the best match is taken to be the one spoken. Alternatively the speech signal can be segmented into smaller units, for example syllables, and match against stored syllable templates. This has the advantage of more compact storage of a large vocabulary as only a finite set of syllable templates must be stored. The major problem with this approach is segmenting the speech signal into syllable units even when given a segmentation algorithm. Confusion can also arise since two syllables that are the same can sound different depending on the context in which the word is spoken.

The most recent classes of algorithm are *Neural Networks*, and *Hidden Markov Models*. A neural network is essentially a kind of associative memory which can be used to associate the speech signal input with a syllable or word output; there are learning algorithms to train the network on a set of associations that have produced interesting results for this type of application. Hidden Markov models use a statistical model to accommodate for the variability which is present in the speech waveform. Here each word or syllable is represented by a set of states, together with the probabilities of the transitions from state to state. High recognition rates have been achieved by using the group of acoustic speech frames that comprise a word or a syllable to determine which state transitions to make.

1.2 Project Goals

This project seeks to investigate a number of algorithms that have been used for isolated word recognition, with the aim of understanding the strengths of each algorithm. The goal of this project is to construct two recognition systems each using a different method, and provide an indepth comparison. Limited experimentation with each system will also be performed so that varied approaches to each system can be discussed. It must also be emphasized that the aim is not to design these systems to operate in *real time*, although they may incorporate some features found in real time speech recognition machines.

In this project both a template matching and a neural network speech recognition system were constructed. Various digital signal processing routines and transformation functions operating on the collected speech samples were implemented, to provide a variety of input types for each system.

The constructed systems achieved very reasonable recognition rates and compare well with other speech recognition systems. This report gives an introduction to the theory behind the techniques used, and explains the components of each system.

Chapter 2 covers the method used to collect the speech data, and discusses the theory and implementation of the digital signal processing and segment functions. Chapter 3 discusses general techniques to detect word boundaries from a speech sample containing a group of spoken words, and presents details of the algorithm used to automatically segment each digit. Chapter 4 introduces the Template Matching approach which forms the first speech recognition system. In Chapter 5 the Neural Network approach to speech recognition is discussed, including the operation of the neural network software that was chosen, which forms the second recognition system. Chapter 6 discusses the methodology used by the suite of speech recognition experiments, and presents their results. In Chapter 7 the concluding remarks are given, along with some comments on improving the systems that were developed.

Chapter 2

Collecting and Processing the Speech Data

In this chapter the nature of the speech data to be collected and the method used to record each subject is discussed. Raw speech samples must be transformed into a more usable format through the use of digital signal processing techniques which are described in the second part of this chapter.

2.1 Speech Data Collection

The systems that have been developed for this project are restricted to operating on isolated words only. To simplify the collection of the speech database, and to provide a moderately difficult task, the *digits* (0-9) are used. This task has been used before to assess speech recognition systems, so comparisons of recognition performance can be made with these published results.

Collecting the speech samples from the subjects was done using predefined *digit sets* consisting of *five* digits each. The digit order was varied in each digit set. For the speaker dependent recognition experiments one subject was presented with 32 digit sets, giving 160 digits so that each digit is spoken 16 times. For the speaker independent experiments, four subjects were used; two males and two females. Each subject was presented with 8 digit sets, for a total of 160 digits; each subject therefore speaks each digit 4 times. The samples were all collected under quiet conditions, and the subjects were asked to leave slight pauses between each spoken digit.

The data was collected using an Apple Macintosh LC at a sampling rate of 22kHz (22,000 samples per second). An 8-bit sampling resolution is provided by the Macintosh hardware. Using the Nyquist theorem, the maximum frequency that can be sampled without aliasing of the sampled signal is $f = s/2$ where s is the sampling rate (22kHz). Human speech contains little information above 9kHz so a 22kHz sampling rate (allowing frequencies up to 11kHz only) is adequate.

2.2 Digital Signal Processing Techniques

Digital signal processing functions are used to transform the raw signal (in this case the speech waveform) into other formats to extract meaningful features from the original signal. There are two categories: one that operates on the entire raw waveform such as segmenting and digital filtering, and another category that operates on individual segments of the waveform.

2.2.1 Normalization and Segmenting the Raw Data

Sampled Macintosh output consists of a stream of 8-bit values. Each 8-bit sample has a range of 0-255 with the zero voltage value set close to 128. An observation after recording the subjects was that this zero voltage value appeared to fluctuate slightly across recording sessions; the same microphone was used for all subjects but different Macintosh machines were used.

It was found that by averaging a period of background noise at the start of each digit set this zero value can be accurately determined. The first 500 samples (22.7ms of speech at 22kHz) was averaged and the zero values were found to range from 126 to 129 across digit sets. It is important to obtain the correct zero voltage value so that the positive peaks and negative troughs of the signal are not skewed.

After the word boundaries in the speech sample have been determined a large group of sample points exist for each word - typically around 10,000 sample values (approximately half a second) when 22kHz sampling is used. To analyze the frequency components over a whole word, the word must be broken into some number of segments. This is done so that segment functions such as the *Fast Fourier Transform* (FFT), or *Linear Predictive Coding* (LPC) analysis can be applied to each of these short intervals to construct *feature vectors*. The segment size needs to be chosen carefully; it must be long enough so that each segment contains enough information for the segment functions to compute a feature vector, while being short enough so that the set of vectors can detect changes in the features across a word.

Previous speech recognition systems have used a segment size of sample points ranging from 20ms to 50ms of speech; this group of sample points is often called a *data window*. The number of points inside a data window is called the *data window width*. One requirement for the input to these FFT routines is that the segment size must be a power of two. Given this, the data window width has been chosen to be 512 points. At 22kHz sampling this represents 23.3ms of speech.

There are also other issues related to data windowing. Taking the raw data exactly as it is sampled for each data window has problems when determining the frequency spectrum because of the sharp on/off effects (also known as *end effects*) that are present at the edges of the data window. The sudden onset of the signal at the start of the data window and the sudden drop off at the end of the data window introduces an additional unwanted high frequency component in the signal.

A remedy for this situation is to multiply each value in the data window by a window function that changes more gradually from zero to a maximum and then back to zero again. Rather than use a square window (the raw samples) a scaling function is used to decrease the effect of the initial and final window values. Some widely used smoothing window functions are the Parzen, and Hamming functions [2]. There is very little difference between the performance of these various windowing functions — as long as the square window is avoided.

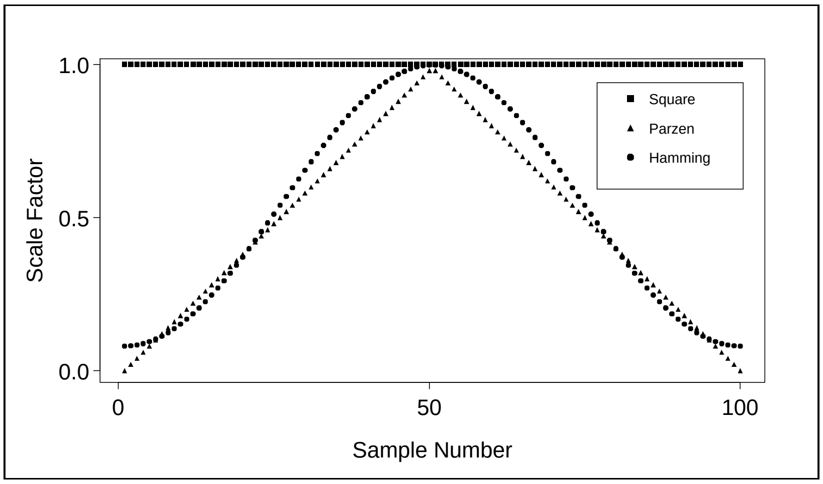

Figure 2.1: Square, Parzen and Hamming Window Smoothing Functions

Figure 2.1 illustrates the square, Parzen and Hamming window functions giving the scale factor on the Y axis for the corresponding sample number on the X axis. One interesting feature of the Hamming window is that the edges of the window never quite reach zero. A segment size of 100 samples was used in the illustration.

Data windows must be *overlapped* when smoothing each window so that the information in the scaled down portion of each window is not lost. Overlapping of the windows has also been shown to keep the spectral variance per data point to a minimum as discussed in [2]. It has been demonstrated that the optimal (or very nearly optimal) size for the window overlap is exactly half the window size, and this has been used in the digital signal processing routines for this project. This gives a data window overlap of 256 points resulting in a data window every 11.6ms.

2.2.2 Digital Filtering

Digital filtering is an important technique that can be used to remove unwanted frequency components from a previously sampled waveform. There are three basic types of filter, a *low-pass filter* that filters out all frequencies above a given cutoff frequency and allows the low frequencies to pass through the filter. Similarly, a *high-pass filter* filters out all frequencies below a given cutoff frequency. A *band-pass* filter is a combination of both a low-pass and the high-pass filter, so that the filtered signal contains frequencies between a specified interval [3].

For this project a low-pass digital filter is needed to remove some of the higher frequencies in the signal of the speech waveform. For speech recognition the frequencies in human speech above 5-6kHz are not usually needed in the frequency analysis routines, as these frequency components contain little important information.

The easiest method of low-pass filtering the raw 22kHz samples is to decrease the sampling rate which is called *decimation*. Furthermore, an integer factor decrease simplifies this process, as after the signal has been filtered every nth sample can be discarded; n is the integer factor of the sampling rate decrease. A factor of two has been used so this gives an effective sampling rate of 11kHz and needs to use a low-pass digital filter with a cutoff frequency set at 5.5kHz.

There are two types of digital filter; FIR (finite duration impulse response) and IIR (infinite duration impulse response). An FIR filter has been chosen for this project over the IIR filter as the response function is easy to design.

Digital filtering is performed through a process called *convolution*. Convolution of two finite duration sequences is equivalent to multiplying the DFT's (Discrete Fourier Transforms) of the two sequences and then applying an inverse DFT.

The filtered signal $y(n)$ can be expressed as a convolution of the input signal $x(n)$ and the unit response function $h(n)$ in the following way

$$y(n) \;=\; h(n) * x(n) \;=\; \sum_{m=-\infty}^{+\infty} h(m)\; x(n-m) \tag{2.1}$$

where $*$ stands for the discrete convolution [4]. For a FIR filter with a finite response window of width $2M+1$ the following equation may be used

$$y(n) \;=\; \sum_{m=-M}^{+M} h(m)\; x(n-m) \tag{2.2}$$

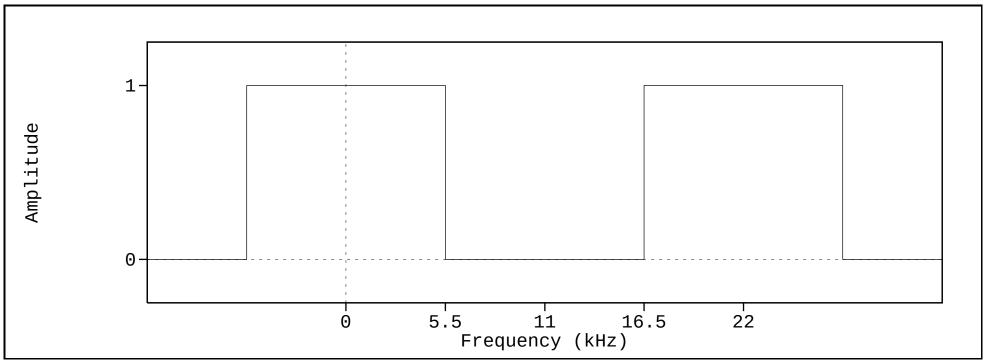

Figure 2.2: Frequency Response of the Filter Function

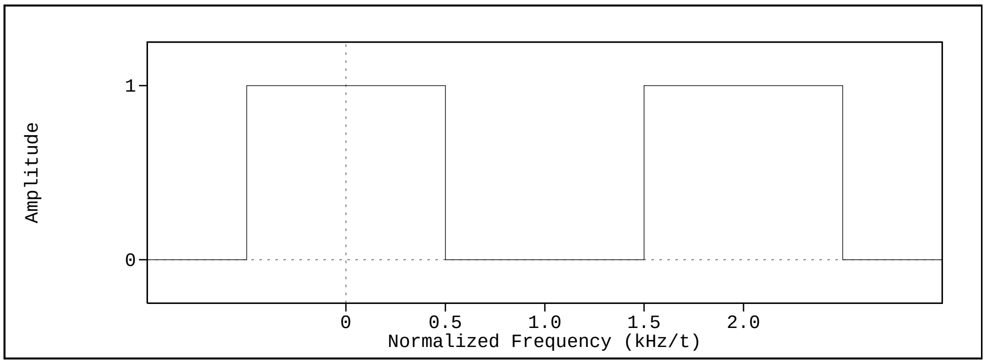

Figure 2.3: Frequency Response of the Filter Function on a Normalized Frequency Scale

The last consideration is the design of the filter function $h(m)$ and its $2M+1$ coefficients. Firstly, the frequency response of the desired low-pass filter must be determined. This is illustrated in *Figure 2.2* where the function is expressed on the sampling frequency scale. It can be seen that this response function allows only frequencies below 5.5kHz to pass through the filter, and that this function repeats above 11kHz because of the requirement that the function is periodic in frequency. Usually this function is expressed on a normalized frequency scale normalized to half the original sampling rate, $t = s/2$, and this is shown in *Figure 2.3*.

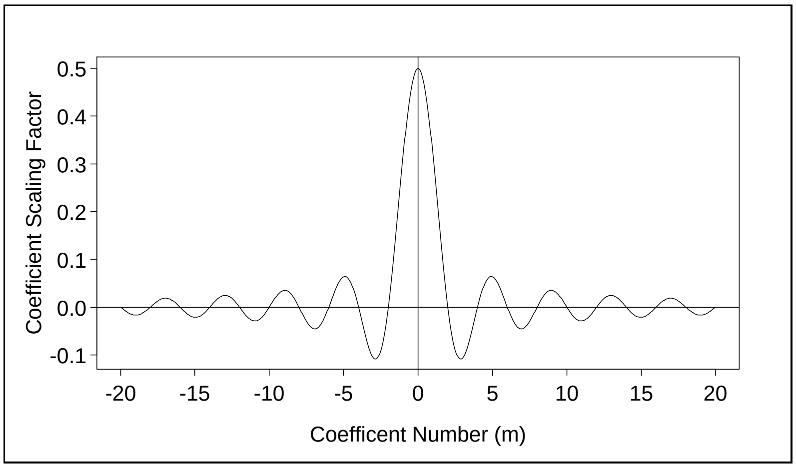

Figure 2.4: Response Function Coefficients for a Digital FIR Filter

As discussed in [3] the coefficients for such a function can be worked out from a finite Fourier series given by

$$h_m = \frac{\sin(v\ \pi m)}{\pi m} = \frac{\sin(0.5\pi m)}{\pi m} \tag{2.3}$$

where v is the ratio of the low pass cutoff frequency to the original sampling frequency (in this case 0.5), m is the index from $-M$ to $+M$ corresponding to each coefficient. A plot of the response function coefficients for a window size of $M = 20$ is given in *Figure 2.4*. Note that there are 41 coefficients in total.

Producing the coefficients of the response function by the above method uses a finite Fourier series expansion of the desired filter frequencies. Therefore the abrupt termination arising from the finite number of terms can cause poor convergence of the resulting series [3]. Windowing the group of coefficients through a finite-width window function ($2M + 1$ width) such as a Hamming window provides a better approximation to the original response function.

The results of the above digital filtering routine using a Hamming window on the response function coefficients is given in *Figure 2.5*. The initial high frequency component of a spoken "three" has been used as the sample waveform. The raw data is at the top, filtering to half the sampling rate (a 5.5kHz cutoff) in the centre, and filtering to a quarter of the sampling rate (a 2.75kHz cutoff) at the bottom. A window width of $M = 20$ was used.

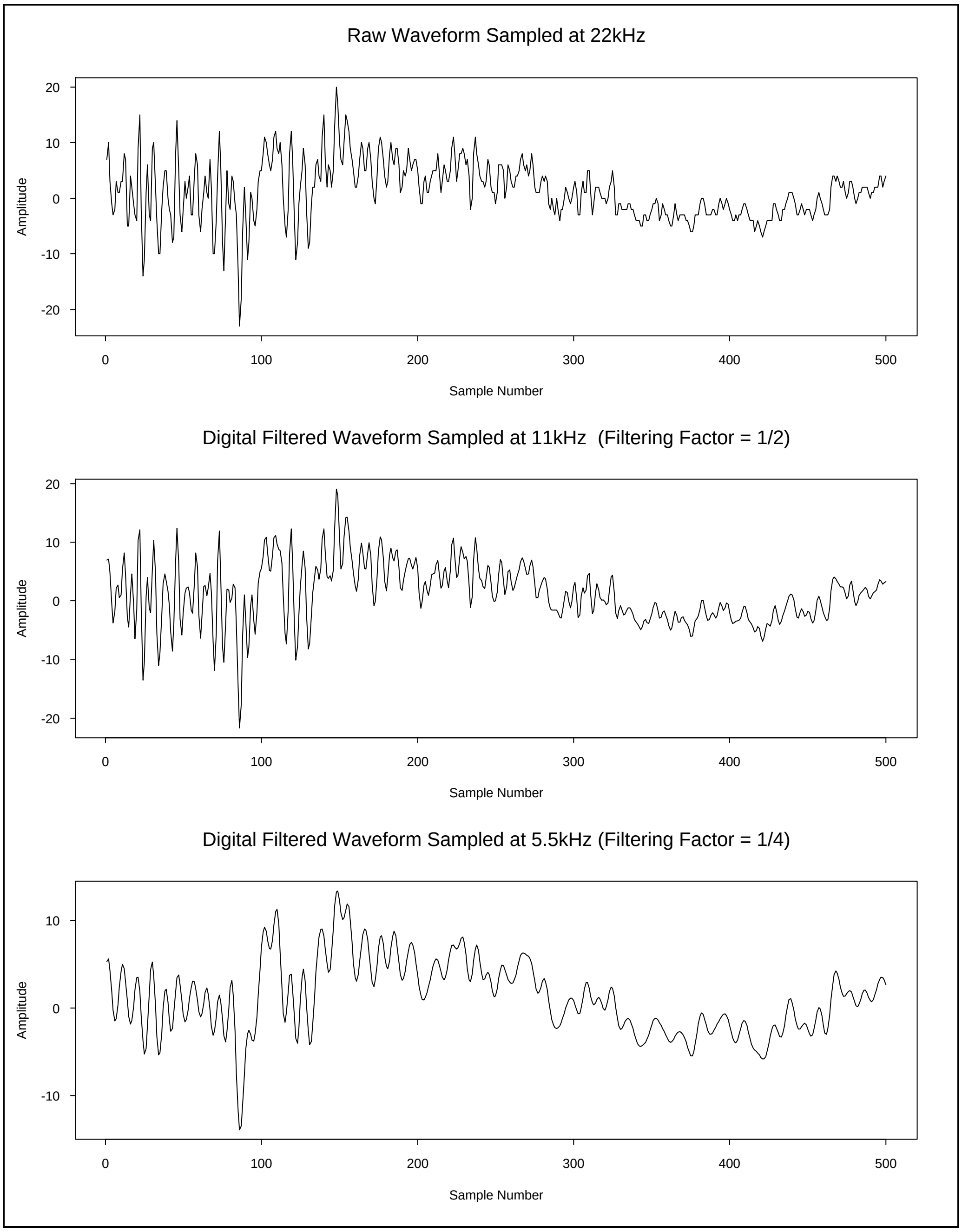

Figure 2.5: Examples of Digital Filtering for a Portion of a "three"

2.3 Segment Processing

This section describes the construction of filter banks which are obtained from the output of the fast Fourier transform, and LPC and cepstral processing which are two predominant techniques for speech processing.

The fast Fourier transform, LPC, and cepstral processing are all routines that operate on short segments (data windows) of the input waveform. The segments are obtained from using a windowing process on the entire waveform as described in the previous section. These routines are used to extract meaningful information by processing each segment of N sample points, and returning some measure or characterisation of the frequency content in the waveform. Usually the processed values are in the form of a vector spanning across the frequency range of the signal. A very important reason why these transformation routines are used is that the length of this vector is much smaller than the N sample points allowing comparisons to be easily made between these vectors. As an example a waveform segment of 256 or 512 sample points can be easily transformed into only 16 filter bank energy values.

2.3.1 The Fast Fourier Transform (FFT) algorithm

In speech analysis, fast Fourier transforms are used to extract the frequency amplitudes for discrete frequency intervals from a sampled signal. The FFT is an optimized version of the discrete Fourier transform (DFT), with the requirement that the number of samples must be a power of two. The DFT assumes that an input signal $s(n)$ is periodic with an arbitrary number of samples, N. However, for speech analysis, it is permissible to consider the input signal not as strictly periodic but as a windowed portion of a much longer signal with an infinitely long period. The FFT routine can be used in spectral analysis, filter bank simulation and digital filtering.

An FFT routine has been obtained which computes the complex FFT for a *real valued* input signal. Given N sample points as input, $N/2$ complex values are computed corresponding to each of the $N/2$ points in the FFT. There are only half the number of points in the complex FFT to input samples because the highest frequency that can exist is half the sampling rate. The frequency amplitudes for each discrete filter ranging between 0Hz and half the sampling rate can be easily computed from this complex FFT by taking the square root of the sum of the squared complex and squared real component for each point in the FFT. Therefore an N sample point real FFT results in $N/2$ filter magnitudes, because the real and complex components for each point are combined together. The magnitudes of these discrete filters can then be used to examine the spectral components, and simulate filter banks for each segment within a word.

Filter Banks

Filter banks are used to measure the magnitude of a signal in a discrete frequency interval, and are suitable for monitoring the frequency changes between speech segments of a word and for comparing speech segments for different words. For real time signal processing filter banks can be implemented in hardware by using band-pass filters. Taking the power of these waveforms that pass through each bandpass filter gives a measure of the signal magnitude for each frequency interval.

For the purposes of this project the discrete filter outputs from the FFT are used as the initial set of band pass filters which are then collapsed into a smaller number of filter banks. The number of filter banks that have been used in previous speech recognition systems range from 12 to 16 banks spanning across the frequency range of the sampled speech. Two methods are described below that obtain a fixed number of filter banks from the FFT output, and these are linearly spaced and logarithmically spaced filter banks.

Linearly Spaced Filter Banks

Linearly spaced filter banks are constructed by combining the available band pass filters into some smaller number of filters corresponding to equally spaced discrete frequency intervals.

As discussed above an N sample FFT produces $M = N/2$ filter magnitudes which are linearly spaced from 0Hz to f_s, half the sampling frequency. M is usually 512, 256 or 128 depending on the segment size chosen. To obtain n linear spaced filter banks the following relationship is used

$$f_m = \frac{f_s m}{(n+1)} \tag{2.4}$$

where f_m is the centre frequency of each filter bank, and m is the filter bank index from $1..n$

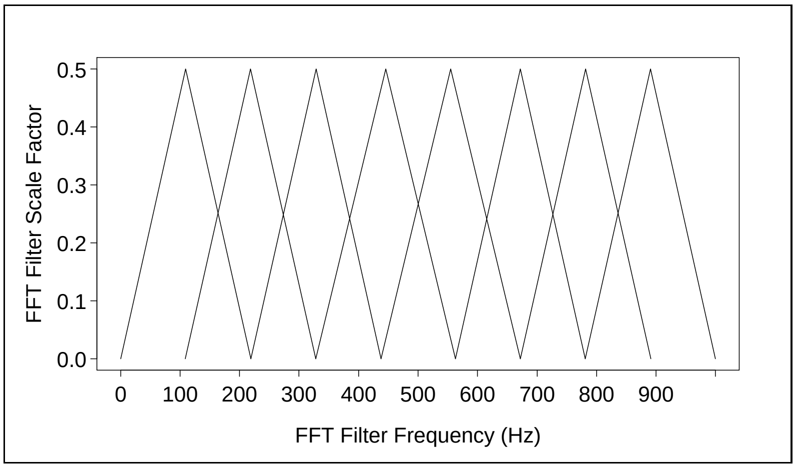

Figure 2.6: Example of Linear Filter Bank Spacing

From these centre frequencies, we can obtain an interval for each linearly spaced filter. An overlap should be taken so that the frequency span of each filter is from f_{m-1} to f_{m+1}, and a Parzen (triangle) window can then be applied to the FFT filter magnitudes inside this interval to obtain each linearly spaced filter bank. The frequency interval from f_{m-1} to f_{m+1} can be easily mapped to the M FFT filters by obtaining the range of FFT filters that correspond to this calculated frequency interval. This interval of FFT filters is then combined through a triangular window which scales each FFT filter magnitude. This is illustrated in *Figure 2.6* that shows the frequency intervals and scale factors for 8 linearly spaced filter banks ranging from 0Hz to 1kHz.

Each linearly spaced filter bank is calculated by summing the scaled FFT filters within its interval, and then normalizing this sum to the total number of filters. This normalization is needed because the total number of FFT filters used for each linear bank may vary depending on the frequency interval. The normalization is performed by dividing the sum by $(w-1)/2$, where w is the width of the linear bank in FFT filters.

The Mel-Frequency Scale for Filter Banks

A variant on linearly spaced filter banks are logarithmically spaced banks. This log scale is also known as the *mel-frequency* scale [5]. The mel-frequency scale consists of a number of linearly spaced filter banks up to a cutoff frequency (about 1kHz), and then a logarithmic filter bank spacing from this cutoff to the maximum frequency (about 5-6kHz). Usually there are an equal number of linear and log banks. The motivation behind this particular method of dividing the banks is that the lower frequency components of human speech (up to 1kHz) contain more useful formant information than the higher frequencies (above 1kHz). Therefore as the frequency of the FFT filters gets higher they are combined into fewer filter banks.

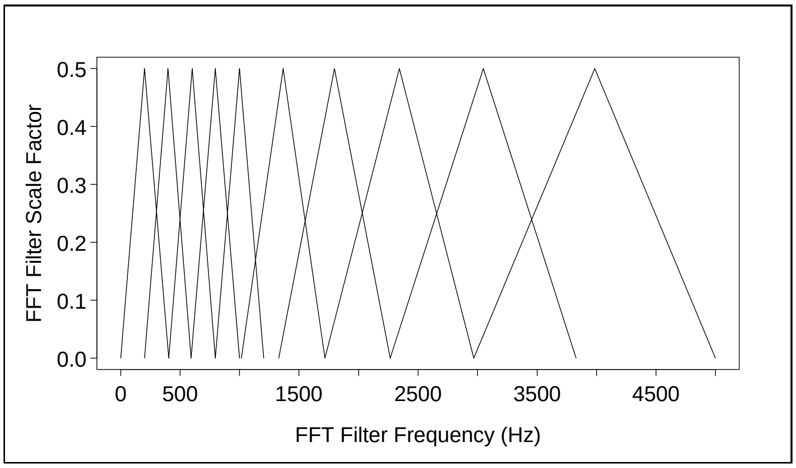

Figure 2.7: Example of Mel-Frequency Filter Bank Spacing

To obtain n logarithmically spaced filter banks the following relationship can be used

$$f_m = f_i + (\mathrm{e}^{\omega m} - 1)\omega_0 \tag{2.5}$$

Again f_m is the centre frequency of each filter bank, and m is the filter bank index from $1..n$. f_i is the initial frequency that the log banks are to start from. This is usually 1kHz and is also the frequency at which the linear banks end. ω_0 is the desired initial spacing for the log banks, and is usually set so the initial spacing is fractionally higher than the linear bank spacing. ω is a constant that defines the exponential increase for the log bank spacing and this is determined by

$$\omega = \frac{\log(1 + \frac{\Delta f}{\omega_0})}{n + 1} \tag{2.6}$$

Here Δf represents the frequency span of the log banks and is given by $f_e - f_i$, the desired end and initial frequencies of the log bank filters. As mentioned usually f_i is set to 1kHz and f_e to about 5kHz giving a frequency span of 4kHz.

In the same fashion as the linear filter banks the range of FFT filters can be obtained that correspond to each log bank, for triangular windowing.

An example of a mel-frequency scale is given in *Figure 2.7* which spans to 5kHz, containing 5 linearly spaced filter banks and 5 logarithmically spaced filter banks.

2.3.2 Linear Predictive Coding (LPC)

Linear predictive coding (LPC) is the second method that was investigated, and is used in estimating the basic parameters of a speech waveform including formant estimation and pitch estimation. The basic idea behind linear prediction is that a periodic signal value can be approximated from a linear combination of past signal values. The *predictor coefficients* are the weighting coefficients used in this linear combination.

Linear prediction is also equivalent to decomposing a signal into source and filter components which fit a source-filter model of speech production. This source-filter model is an acoustic model consisting of an excitation source driving a series of filters. An important simplification of this model is that the source and filters are considered to be independent of each other. These properties of linear prediction make it a particularly appropriate technique for analysing a speech waveform.

LPC assumes that the input waveform $x(t)$ has been digitally sampled. If $x(t)$ is continuous, then the current sample $x(n)$ can be predicted from the previous sample in the following way

$$x(n) = a_1 x(n-1) + e(n) \tag{2.7}$$

where the coefficient a_1 is chosen such that the error signal $e(n)$ is small.

This idea can be extended further by predicting $x(n)$ by now considering the last p samples using

$$x(n) = e(n) + \sum_{k=1}^{p} a_k x(n-k) \tag{2.8}$$

The entire waveform can now be predicted from the group of LPC coefficients, a_k and the error signal $e(n)$. LPC offers a fast and very compact form of coding a waveform, as a small series of coefficients are produced that describe a large number of waveform samples.

The number of coefficients, p is known as the LPC *order*. As discussed in Rabiner and Schafer [4] this order should be chosen carefully. The choice of p is dependent primarily on the sampling rate. Two LPC coefficients are needed per kHz due to the vocal tract contribution, with an additional 3-4 coefficients needed to adequately represent the source excitation. Thus for an 11kHz sampling rate 11 coefficients are needed to adequately represent the vocal tract (2×5.5kHz), including 3-4 extra coefficients giving 14-15 coefficients in total.

To compute the LPC coefficients a routine has been obtained that uses the *autocorrelation technique* (from Tony Robinson at Cambridge, UK) to compute the LPC coefficients, along with the reflection and autocorrelation coefficients. The reflection and autocorrelation coefficients can be used in further formant analysis but for the scope of this project, only the LPC coefficients will be used. The LPC routine operates on an input speech segment of any size and computes a fixed order of LPC coefficients.

2.3.3 Cepstral Processing

Cepstral transforming an input signal was also investigated as a technique to extract meaningful features from a speech signal. The motivation for this transformation is that a speech production mechanism can be considered as a convolution of the excitation function with the vocal tract impulse response [6]. The cepstrum, $\mathcal{C}$ is defined as

$$\mathcal{C} = \mathrm{IDFT}(\log \mid \mathrm{DFT}(x(n)) \mid) \tag{2.9}$$

which is the inverse discrete Fourier transform of the logarithm of the discrete transform of an input signal $x(n)$. The property of convolution is used by cepstral processing to separate the excitation from the vocal tract impulse response in the following way

$$x(n) = u(n) * s(n) \tag{2.10}$$

$u(n)$ is the vocal tract impulse response, $s(n)$ is the excitation signal and the $*$ represents discrete convolution. The steps taken to compute the cepstrum can be paralleled to this equation in the following way. Taking a DFT of the signal $x(n)$ produces a signal that is the product of the DFT of both $u(n)$ and $s(n)$. Taking the log magnitude of this resultant signal gives a new signal corresponding to the sum of the log magnitudes of the DFT's of $s(n)$ and $u(n)$. Now an IDFT (inverse discrete Fourier transform) of this signal is the sum of the cepstra of the excitation and the vocal tract impulse response. This is known as the cepstrum of the original signal. The horizontal axis of a cepstrum, which has the dimension of time, is called *quefrency* (an anagram of frequency).

To show that the cepstral transformation does in fact separate the excitation and the vocal tract impulse response these two components can be considered separately. The excitation signal can be viewed as a sequence of quasi-periodic pulses whose Fourier transform consists approximately of a line spectrum with the lines spaced at *harmonics* of the fundamental frequency. Taking the log magnitude does not affect this characteristic, and the IDFT operation yields another quasi-periodic waveform with the pulses spaced at the fundamental period. Thus the cepstrum of this excitation should consist of pulses around $n = 0, T, 2T, ...$ where T is the pitch period. Now consider the vocal tract impulse response. It is a sequence that generally lasts for only 20 to 30ms and has a Fourier transform that is a slowly varying function of frequency [7]. The process of then taking its log magnitude followed by an IDFT yields a sequence that lasts for only a small number of samples (generally less than the number of samples in a pitch period). Thus the cepstrum can differentiate the excitation information (stored in the late part of the cepstrum) from the vocal tract impulse response (stored in the early part of the cepstrum).

Usually the cepstrum is computed by the method described above, by taking the logarithm of each FFT filter and then performing an inverse FFT on this result to build the cepstrum. However, an alternative way of obtaining the cepstrum coefficients is available which uses a recursive process involving a group of previously computed LPC coefficients for the input signal. This type of cepstrum is called an LPC cepstrum. One restriction is that the order of the resulting cepstrum coefficients should be a few less than the order of the LPC coefficients. Computing the cepstrum in this way has been chosen as it is also a much faster process than performing the computationally expensive FFT and inverse FFT steps.

Chapter 3

Word Boundary Detection

3.1 Introduction

Word boundary detection is one of the major problems for speech recognition systems, especially when real-time continuous speech recognition is the goal. For segments of speech larger than words the speech stream must be segmented into smaller units (e.g. words, phonemes, syllables, diphones). However, this segmentation is difficult as many speech sounds merge together with no apparent boundary. It is still a matter of debate as to whether words, syllables, phonemes, or some other quantity should be used as the basic unit in speech recognition [1].

For this project the unit of recognition is words, and since the speech data was obtained from digit sets consisting of five digits each, each word or digit must be separated from the speech sample. A simple solution to the word boundary problem in the context of this project is to separate each word manually with a simple waveform editor. However, this is a rather time consuming approach especially when using a large group of speech samples. A speech recognition system is of little use unless it can automatically determine where each word starts and ends from a larger speech sample. Subsequently, a fair proportion of time was given to investigating word boundary detection in the hope that a routine could be developed to automatically segment each digit.

In this chapter various techniques are discussed that can be used to process segments of the speech signal to gain a measure of how likely it is that the segment is silence (background noise) or spoken speech. Later in the chapter the method that was chosen to automatically segment the digits is discussed.

3.2 Word Boundary Detection Measures

Human speech sounds can be divided into two categories - *voiced* and *unvoiced* sounds. Voiced speech sounds are produced through the vocal tract which outputs sharp acoustic resonance frequencies called *formants*. These sounds are rich in harmonics, and include vowels, nasal consonants, liquids (/r/ and /l/), and glides (/w/ and /y/). A good example of digits using voiced sounds are the digits "one", and "nine". Unvoiced sounds are produced by forcing air through some constriction (such as between the tongue and teeth in "th" sounds), or by an abrupt release of pressure built up in the vocal tract (such as behind the lips in a /p/). The spectral peaks of these unvoiced sounds (called fricatives) generally lie between 2 and

7kHz, which is significantly higher than for voiced sounds. Unvoiced sounds also tend to have a lower energy (or amplitude) component than voiced sounds. To confuse matters further, voiced fricatives also exist such as /z/ and /v/, which contain both voiced and unvoiced excitation [8].

To detect the presence of speech for these different situations the raw speech must be examined, and a measure must be obtained which can accurately differentiate between the signal for spoken speech and the signal for gaps between this spoken speech. The suitability of three measures for word boundary detection are presented in this section. These measures all operate on short segments of speech, and are the *average magnitude*, the *average zero crossing rate* and the *average zero crossing with threshold rate.*

3.2.1 Energy or Average Magnitude

To gain a measure of the variation in amplitudes of speech segments, the *energy* or *average magnitude* of each segment can be used. This measure is important for detecting speech from background noise, and also for detecting different types of speech containing high or low overall energy components. The energy for a segment of speech is given by the following [4]

$$E = \sum_{k=0}^{n-1} x(N+k)^2 \tag{3.1}$$

where E is the energy in the segment of size n and $x(k)$ are the sample points of the waveform. N is the start sample number of the segment in the larger group of waveform samples. One difficulty with this energy measure is that it is very sensitive to large signal levels (since they enter the above computation as a square). One way of alleviating this problem is to use the average magnitude function given by [4]

$$E = \sum_{k=0}^{n-1} \mid x(N+k) \mid \tag{3.2}$$

The average magnitude has been chosen for this project also because it is computationally less expensive; absolute values are somewhat faster to compute than squares.

3.2.2 Average Zero Crossing Rate

Another very useful function is the *average zero crossing rate.* The rate at which zero crossings occur throughout a segment is a simple measure of the frequency content of a signal and can also be used to detect speech from background noise, and for pitch changes from segment to segment. The zero crossing rate can be thought of as the rate at which the sampled points on the waveform cross the zero voltage reference value. A simple way to compute this is to count the number of times the sign of the current sample is different from the sign of the previous sample.

3.2.3 Average Zero Crossing with Threshold Rate

One major problem with the zero crossing rate is that while it is good at detecting high frequency components, like the initial high frequency component in a "two", it has trouble distinguishing background noise from a spoken low frequency signal. Examining these two types of signal in more detail shows that they have comparable zero crossing rates because the background noise contains a very low powered signal that fluctuates around the zero voltage reference level. These fluctuations cause zero crossing counts to register because the standard zero crossing rate has no way of distinguishing that this crossing was triggered by a tiny peak/trough or a large peak/trough in the input signal.

Observation of this problem indicates that the zero crossing rate would be more useful if it could be modified to take into account the power of the signal before registering each crossing. An easy way to incorporate this into the usual zero crossing rate is to change the zero voltage reference value into a band which defines an interval that the signal must exceed to register a crossing [9]. A threshold value, T, is used to define this band so that the interval given between $-T$ and $+T$ is considered a dead area that crossings cannot be registered in. Crossings are now registered *only* if the signal makes a transition from a level above $+T$ to a level below $-T$. Low power transitions between the threshold band are ignored thus building a simple power mechanism into the zero crossing rate.

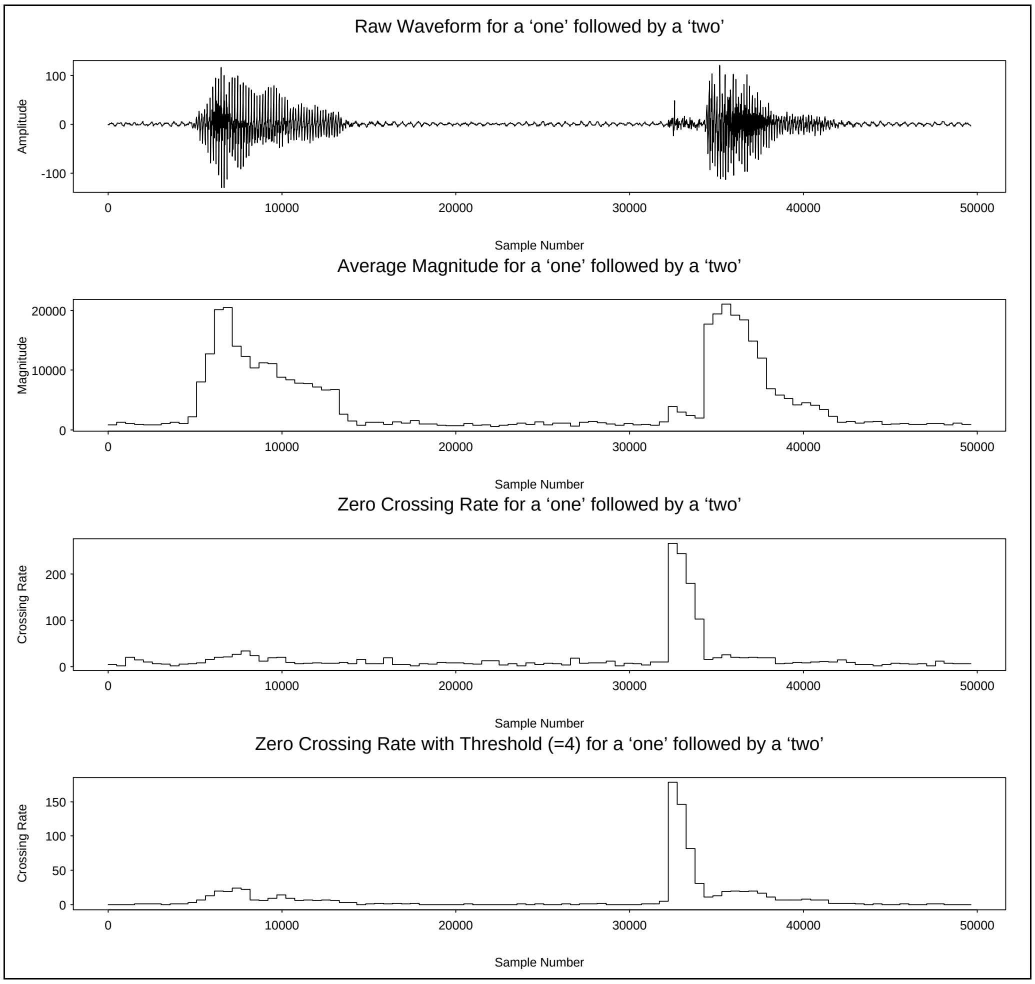

Figure 3.1: Examples of Word Boundary Detection Measures for a "one" followed by a "two"

Figure 3.1 shows three graphs, the first is a raw waveform of a "one" followed by a "two"; below this is a plot of its average magnitude for the corresponding segments, the zero crossing rate, and then the modified zero crossing rate with threshold at the bottom. It can be seen from these time series plots that the peak in the power graph is a very useful metric for detecting the presence of a spoken word. However, the power dips at the unvoiced sound of the "t" in "two" which has a lower power because of the softer nature of unvoiced sounds. The zero crossing rate appears to be of little use in detecting the boundaries for the "one", as there is a significant peak well before the "one" actually started. Also, during periods of silence the zero crossing rate is susceptible to background noise as shown by the numerous small peaks. For detecting the unvoiced sound of the "t" in "two" the zero crossing rate does have particular importance because these sounds contain a high frequency component that corresponds to the large peak. Examining the modified zero crossing with threshold plot shows very similar characteristics to the zero crossing plot. However, during periods of silence the zero crossing with threshold rate shows its superiority over the standard zero crossing rate as it keeps consistently lower due to the filtering out of the low power crossings. By combining the power and the zero crossing with threshold measures it appears viable to estimate word boundaries, and also gives a simple method of detecting voiced from unvoiced sounds.

3.3 Automatic Segmentation of the Digits

Detecting the digit boundaries was done automatically using two of the measures discussed above; the power and the zero crossing with threshold rate. The appropriate threshold levels to distinguish speech from background noise were determined for these two measures, and an algorithm was developed to scan across the measured values for each segment.

3.3.1 Choosing the Threshold Levels

The threshold levels for the two chosen measures were determined by hand by examining their values for the segments in the digit sets. The gaps between words could easily be seen as distinct lows in the measures, and the threshold levels were carefully chosen so words could be clearly distinguished from the background noise.

For the zero crossing with threshold rate, the threshold to eliminate background noise in the silence periods had to be determined first. This was done by examining the raw waveform after the normalization process. A threshold of 4 was chosen so that zero crossings must cross though the interval defined by -4 to $+4$ inclusive to register a crossing.

The segment size to use for the measures was the next consideration. A segment size of 128, 256, and 512 samples were tried, which corresponds to 5.8ms, 11.6ms, and 23.3ms of speech respectively. Initially it was thought that the smaller segment sizes would be more useful as the digit boundaries could be determined more accurately. However, this was not the case in practice because the range of values that the measures produced became smaller as the segment size decreased, especially for the zero crossing with threshold rate. A segment size of 512 was chosen as this gave an interval large enough to obtain a large range of values for the measures.

To determine the threshold level of the two measures to detect the digits from background speech, threshold levels of 4 were tried for the zero crossings, and the power level was set to 2500. These threshold levels were inclusive so that a zero crossing rate of 4 and above is considered significant. After trying the algorithm it was found that this power level was set too low as some of the gaps between digits contained power values of around 3000; this arose where the digits were spoken close together. To prevent the algorithm detecting two words as one, some experimentation was needed, and a level of 3300 was finally chosen as the threshold value for the power. The zero crossing threshold rate of 4 crossings was large enough to avoid most of the gaps, so this was adopted.

3.3.2 The Word Boundary Detection Algorithm

In this section the algorithm used to segment the digits will be discussed, including the parameters used for the algorithm.

The algorithm started as a simple scan across the values of the measures for the segments in the digit sets. If the power *or* zero crossings were greater or equal to the previously calculated thresholds then this was considered as a period of speech. Two problems were found. The first was that even though the thresholds were set at conservative levels, occasionally one of the measures exceeded the threshold during a digit gap. The second arose from chopping a word boundary too early because of a slight pause in a digit (e.g se-ven, fi-ve), resulting in two words being detected.

The first problem was solved by requiring that a word must consist of a minimum number of segments. This minimum number was set to 10 segments which corresponds to about one fifth of a second. As an extra precaution the average segment power was calculated over the segments for the supposed word and if less than some minimum level, then this speech segment is rejected as containing a spoken word. Setting the average segment power level to 2900 was used in conjunction with the 10 segment length which had the effect of rejecting breath or other disturbances in the recordings (e.g. mouse clicks) being detected as digits.

To solve the second problem of pauses in words it was not desirable to lower the power and zero crossing threshold parameters any further as this caused even more problems with the algorithm incorrectly determining words during periods of silence. Instead a *segment lookahead* was created that allows the algorithm to look ahead a number of segments for another significant zero crossing or power value. This allows two intervals of speech that are very close together to be concatenated into a single word. This segment gap was set at 6 segments or about one seventh of a second.

In the segmentation process warning messages were printed if exactly five digits were not found in a digit set, so that the above word boundary parameters could easily be finely tuned to develop a general word boundary routine. The parameters were customised so that given any digit set, exactly five digit boundaries were reliably determined.

Preliminary testing using these automatically segmented digits showed that consistently higher recognition rates were achieved by including one additional segment at the beginning and end of each word. This was also included into the word boundary detection algorithm.

3.4 Word Boundary Detection Examples

Examples of detecting the word boundaries for the digits of a "one" and a "two" are given in *Figure 3.2* and *Figure 3.3* respectively. The top plot shows the raw data and below this the power and the zero crossing rate for each segment are plotted. The dotted lines on the raw data plot show the digit boundaries as determined by the above algorithm, while the dotted lines on the plots for the word boundary detection measures indicate their threshold levels.

Examining *Figure 3.2* shows that the power and zero crossing rates both detect exactly the same boundary for this "one". Note that the dotted boundary lines on the raw waveform include the additional segment in the boundary. The algorithm has no problems determining

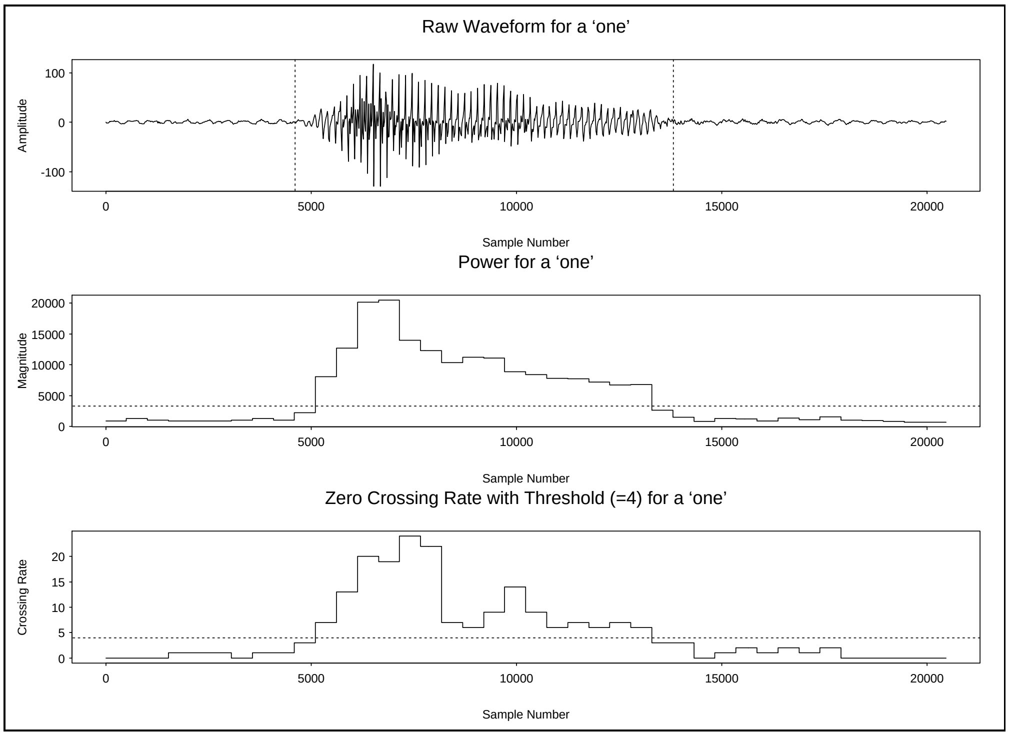

Figure 3.2: Word Boundary detection for a "one"

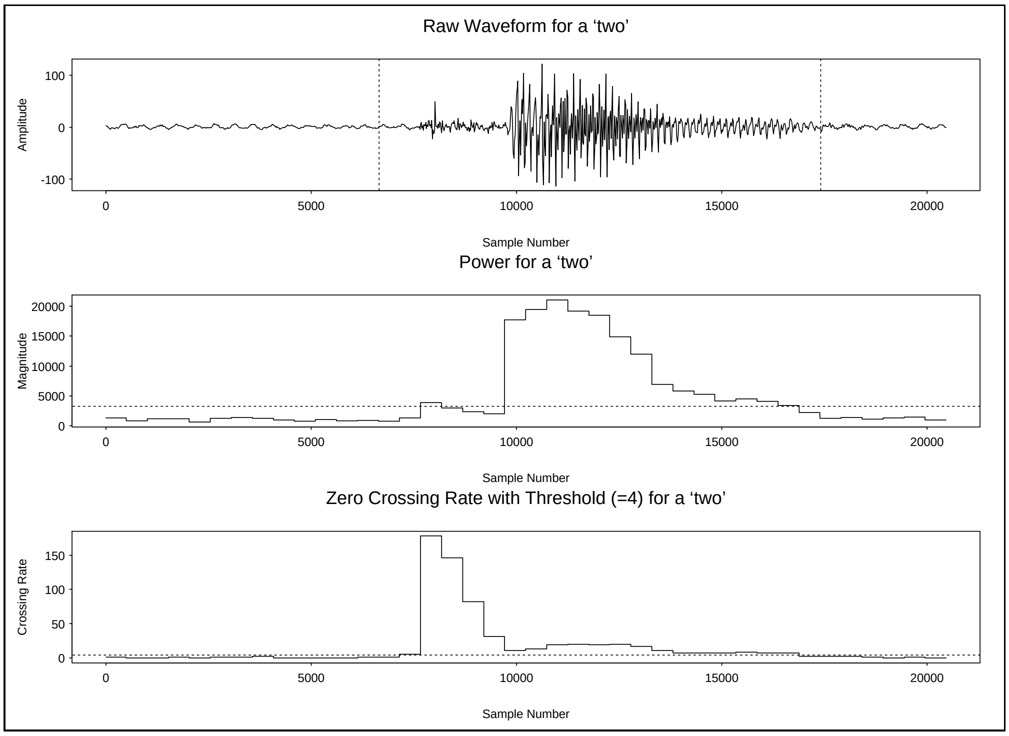

Figure 3.3: Word Boundary detection for a "two"

voiced portions of speech such as this “one”, as both the power and the zero crossing rate work well as measures on their own. *Figure 3.3* highlights the importance of the zero crossing with threshold measure as the peak for the unvoiced “t” sound in the “two” corresponds to the large peak. Possibly the start of the “two” was detected one segment too early, but as mentioned this appears to help rather than hinder the recognition rate. In this case the power measure on its own would still detect the boundary correctly, as the algorithm contains the segment lookahead which would join the two peaks together into one word. An observation from trying both of these measures separately showed that the zero crossing with threshold rate almost suffices as the only measure needed. This is almost certainly due to its property of being able to ignore most of the background noise enabling a tighter threshold level to be used. However the power measure was retained in the algorithm as it was useful in adding a small number of additional speech segments in some cases where the zero crossing rate had the value of 3.

Chapter 4

Speech Recognition using Template Matching

4.1 Introduction

The idea behind a template matching approach to speech recognition is simple : a collection of processed speech patterns (the templates) are carefully selected and stored as reference patterns representing the dictionary of candidate utterances. Recognition is then carried out by matching an unknown spoken utterance with each of these pre-stored patterns and selecting the category of the best matching pattern to be the spoken utterance. The speech unit used in template matching systems is usually chosen to be an entire word; this is so segmentation and classification of smaller speech units such as phonemes is then avoided.

Template matching systems become more expensive as the vocabulary size increases; processing and selecting the reference templates becomes a larger task, and of course matching to more templates takes longer. More confusion results in the matching process as the reference set grows because of similarities between templates for alternative words. In the context of this project the limited digit vocabulary avoids most of these problems, although for the speaker independent recognition task a high degree of variability in templates for the same word is to be expected from the different speakers.

It is well known that variation in speaking rate causes fluctuation over a speech pattern time axis, and it is desirable to remove this fluctuation by time-normalizing the speech pattern. Furthermore, linear normalization is not considered to be sufficient and to find the optimal nonlinear time alignment between two templates, dynamic time warping has been proposed by Sakoe and Chiba [10]. This algorithm attempts to align the templates in time by dynamically stretching or compressing the time axes between the two templates.

Selecting the reference templates from the total number of templates available is not a trivial task as it is important to choose the set of templates that best represent the words to be recognised. Multiple speaker speech recognition often requires template *clustering* techniques to achieve reasonable recognition rates. Groups of similar templates are combined into clusters and the best representative for each cluster is chosen. In the case where more than one cluster is found for a word, more than one reference template is stored for this word.

This chapter discusses the construction of the templates and includes examples of the four types of templates that were investigated. The dynamic time warping algorithm is presented, followed by the method used to select the reference template set.

4.2 Template Construction

Word templates consist of a set of vectors that are computed for short segments of speech which span across the word. Each vector contains a fixed number of coefficients that are obtained from processing the short segment that corresponds to this vector.

The raw waveform is first digitally filtered to 5.5kHz and downsampled so the sampling rate is halved to 11kHz. Next the entire speech waveform for the word is partitioned into segments of a fixed length, with each segment overlapping. Each segment is then windowed by a Hamming window and is passed to one of the segment processing functions discussed in Chapter 2.

Four segment processing functions were chosen to give four distinct template types; linearly spaced filter banks, mel-frequency spaced filter banks, LPC and cepstral processing. Using these different template types allow comparisons between the recognition rates for each processing function.

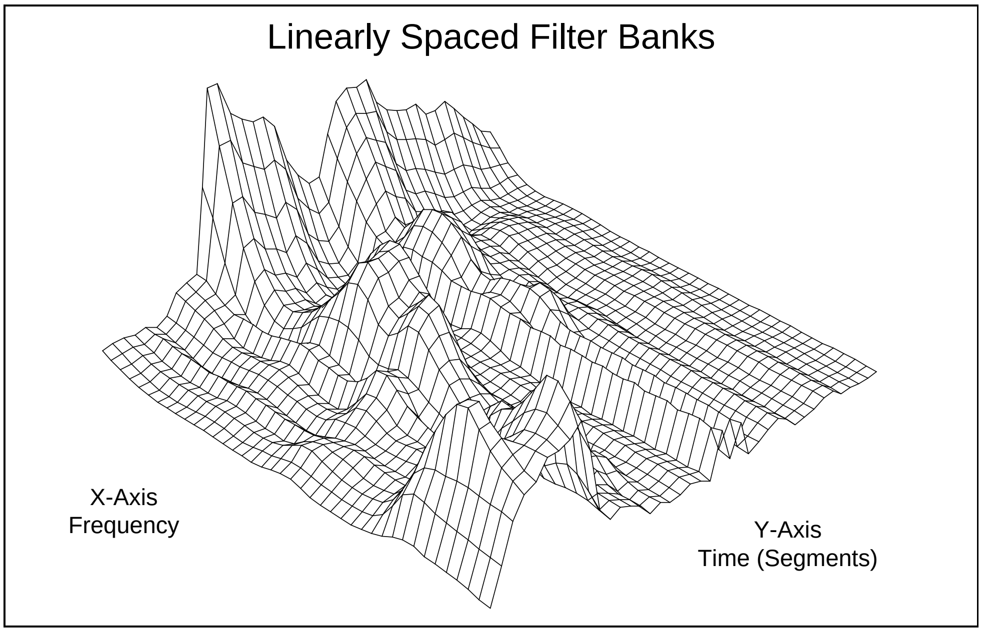

Figure 4.1: A Linearly Spaced Filter Bank Template for a "seven"

To visualise the word templates for these functions, *Figures 4.1* to *4.3* depict word templates for the same spoken "seven". The three dimensional perspective graphs show frequency (or quefrency for the cepstral template) on the X-Axis. The values increase from left to right so that the lowest frequency component is on the left of the X-Axis, and the highest frequency is on the right. Along the Y-Axis, the duration of the word is given, also increasing from left to right. The first segment of the word is on the left of the Y-Axis, with the last segment on the right. All of the word templates contain 42 segments across the Y-Axis. *Figure 4.1* and *Figure 4.2* for the filter banks both have 15 filter banks along the X-Axis, with *Figure 4.3* containing 13 quefrency coefficients. An LPC template was not plotted; the independent nature of each LPC coefficient does not produce a smooth function when they are plotted together as a set. Note that the perspective plots are generated on a 40 by 40 grid so interpolation is used to smooth the original function.

Comparing *Figure 4.1* and *Figure 4.2* for the filter bank templates, the linearly spacing has a

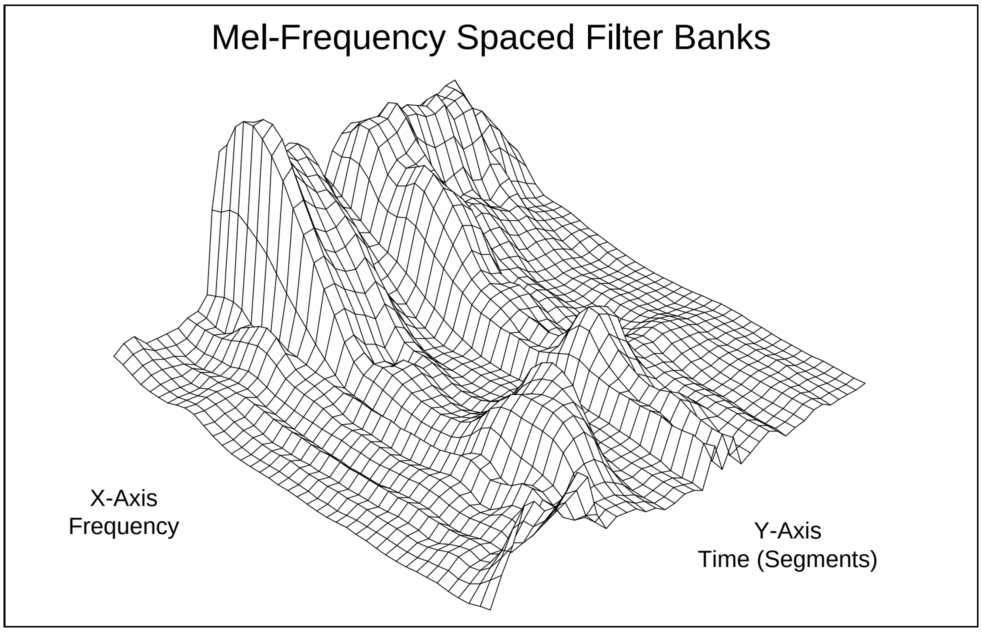

Figure 4.2: A Mel-Frequency Spaced Filter Bank Template for a "seven"

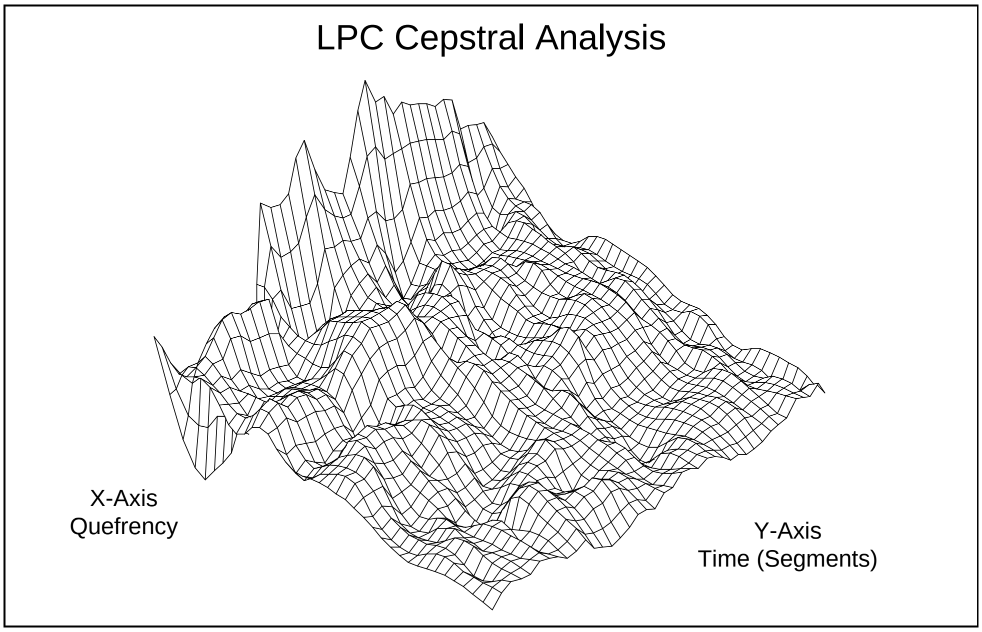

Figure 4.3: A Cepstrum Template for a "seven"

much larger initial high frequency peak for the "s" in seven. While this peak is still present on the mel-frequency scale, it is much less pronounced as fewer filter banks are used to represent this range. However, the mel-frequency scale template holds more low frequency information as it contains more filter banks for the lower frequency range. Looking closely at the two graphs shows that although the same overall frequency structure is present in both, each has detail over a different range, which makes them interesting to compare in a recognition system.

The last template, *Figure 4.3*, contains an LPC cepstral transform of each segment so the X-Axis represents quefrency. Examining the early portion of the cepstrum shows a strong relationship between the height of the low frequency peak in the filter bank plots. The high frequency "s" is represented by a negative trough in the early portion of the cepstrum.

4.3 Dynamic Time Warping (DTW)

A template matching speech recognition system relies on an efficient mechanism for comparing two templates and obtaining a measure of the "distance" between them. This distance is used both in the selection of the most suitable templates for the reference set, and in the recognition matching process to choose a spoken utterance from the reference set for an input template. Problems that arise are the dynamic stretching or compressing of the templates during the matching process to cope with different speaking rates. The algorithm must also be as fast and as accurate as possible, especially for a speech recognition task involving a large vocabulary. This section describes dynamic time warping (DTW) which is an efficient *dynamic programming procedure* for achieving an optimum non-linear time alignment between two templates.

An input template A, and a reference template B can be expressed as a sequence of feature vectors obtained from each segment.

$$\begin{aligned} A &= A_1, A_2, A_3, ..., A_i, ..., A_I \\ B &= B_1, B_2, B_3, ..., B_j, ..., B_J \end{aligned} \tag{4.1}$$

The input template consists of I segments, and the reference template consists of J segments.

Figure 4.4 shows the two templates A and B, with their feature vectors developing across the X and Y axis respectively. Note that the template lengths differ, 20 for A and 15 for B. A *time warping path* can be mapped onto the time axis which represents the timing difference between the two templates. If there is no timing difference between the two templates the time warping path will be given by the line $i = j$. However, the larger the timing difference the further the warping path will deviate from this diagonal line.

Every dot at (i, j) on *Figure 4.4* is a measure of the difference between the two feature vectors A_i, and B_j. The Euclidean distance between two vectors, both of length k, can be used [6].

$$d(i,j) = \sqrt{\sum_k [A_i(k) - B_j(k)]^2} \tag{4.2}$$

However, in this project to save on computation the Chebyshev or norm of the difference between two vectors is used instead [6].

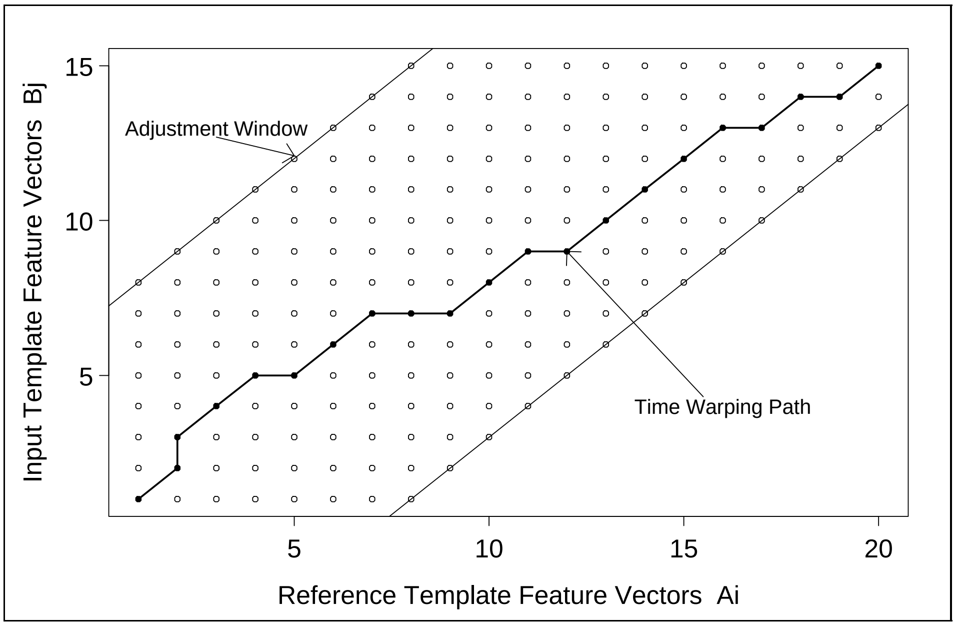

Figure 4.4: Time Warping Path and Adjustment Window Definition

$$d(i,j) = \sum_k \mid A_i(k) - B_j(k) \mid \tag{4.3}$$

Given the set of these minimum distances between the feature vectors for A and B all that remains is to select the time warping path from the points $(1,1)$ to (I,J) that represents the *minimum cost* of the distances along the path. This minimum cost provides the "distance" measure between the two templates.

To compute this minimum cost along a path from $(1,1)$ to (I,J) the minimum cost can be computed for each intermediate path from $(1,1)$ to the points (i,j) within the adjustment window. Each of these costs is kept in a separate cost matrix. The (i,j) costs are computed in a left to right fashion along the A_i columns, and then from bottom to top along the B_j rows. An interesting feature of propagating the minimum costs in this way is that although the minimum cost is known for the path from $(1,1)$ to (i,j) the path to achieve this minimum is not explicitly stored. However, it is the cost that provides the distance measure and not the path.

Before the constraints imposed on the time warping path are discussed it is important to consider the general properties that this algorithm assumes. The first assumption is that the information inside both templates has been obtained from a time-sampled process with a common and constant sampling period. Also, each feature vector inside a template is assumed to contain the same amount of linguistic information as any other feature vector.

4.3.1 Time Warping Path Restrictions

Obvious restrictions on the time warping path include the essential properties of a time axis for speech — continuity and monotonicity. There are two other interesting restrictions that also prune the possible paths that the algorithm has to search; the adjustment window, and the slope constraint.

The adjustment window allows a sliding window of a fixed length to be imposed on the time axis. This is illustrated in *Figure 4.4* where a window length of 8 has been used. This window prevents an excessive timing difference in the time warping path, and also prunes the number of feature vector distances that need to be computed.

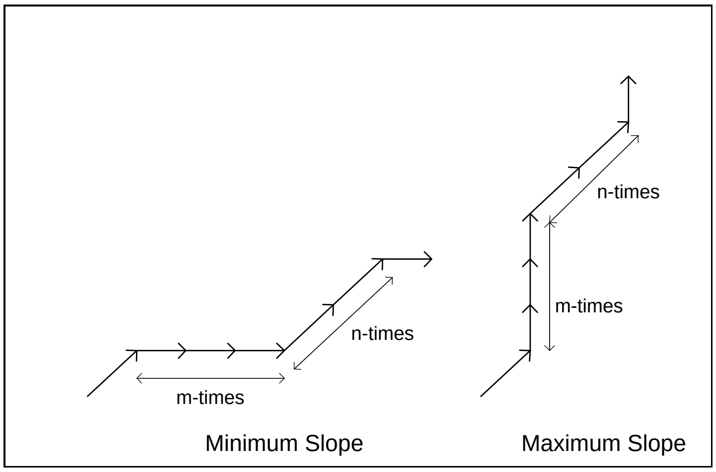

Figure 4.5: Minimum and Maximum Slope for the Time Warping Path

Computing the minimum cost of the path to (i, j) can be simplified by imposing a slope constraint on the gradient of the time warping path. This ensures that neither too steep nor too gentle a gradient is permitted, as this may represent an unrealistic stretching between the segments of A and B. The slope constraint specifies that if the path moves forward in the direction of the X (or Y) axis exactly m times, then the path cannot step further in this same direction before first stepping at least n times in the diagonal direction. The minimum and maximum paths are shown in *Figure 4.5* and are expressed in terms of m and n. The slope constraint can also be expressed as the following ratio which gives a measure of the slope intensity

$$P = n/m \tag{4.4}$$

The larger the P measure, the more restricted the slope is. $P = 0$ gives no restrictions on the time warping path, and $P = \infty$ restricts the path to the line $i = j$. This constraint allows the number of available paths to a point (i, j) to be reduced.

Given slope constraints of $P = 1$ and $P = 1/2$ *Figure 4.6* shows the only permissible paths to reach the point (i, j). Looking at the paths for slope $P = 1$ shows the three permissible paths from the lower marked points (the cost matrix values) to (i, j). Note that the two outer paths pass through the intermediate points $(i - 1, j)$ or $(i, j - 1)$. To calculate the new cost at (i, j) for each of these paths, the distances (obtained from the distance matrix) at each intermediate point including (i, j) are totalled and this is added to the cost matrix value. A weight coefficient is also used that weights these intermediate distances depending on the current slope of the path at the intermediate point. Three paths are available to reach (i, j) using slope constraint $P = 1$, and five paths for $P = 1/2$. Thus the minimum cost of the path from $(1, 1)$ to (i, j) is simply the minimum cost of these three or five permissible paths. Not every path will be valid as some of the cost matrix points may be outside the adjustment window.

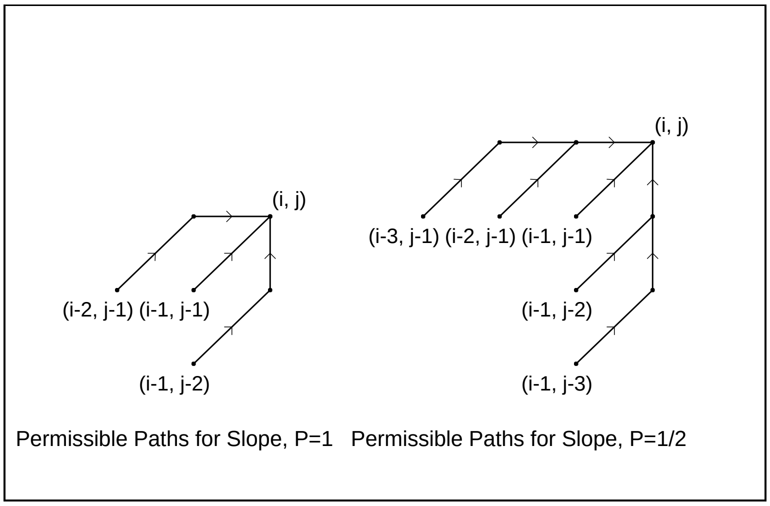

Figure 4.6: Possible Paths from Cost Matrix Points to the Point (i, j)

Initially the cost at $(1,1)$ is set to $2d(1,1)$. The weighting coefficient for this initial distance is 2. From the initial cost the minimum costs propagate up through the adjustment window until the minimum cost is calculated for the point (I, J). It is this cost at (I, J) that gives a measure of the distance between the two templates.

4.4 Selecting the Reference Template Set

Selecting the reference templates must be done carefully, as a chosen template must match all the templates for the word it represents with a small DTW distance, as well as matching all the other templates for the alternative words with the largest possible DTW distance.

Dynamic time warping forms the major part of the template selection routine. Each template for a given digit is tested with every other available template for this *same* digit. These results are averaged to produce the average distance, A_s, for this template. Next this template is tested with every other available template for the digits that are different. These results are also averaged into a distance measure, A_d. Also, the minimum DTW distance found from the different digits is retained in A_m. Now the average separation $(A_d - A_s)$ and the minimum separation $(A_m - A_s)$ can be combined to give an overall measure, S :

$$S = (A_d - A_s) + w(A_m - A_s) \tag{4.5}$$

w is a weighting factor used to increase the importance of the minimum separation and was set at 2. The template with the largest S can now be chosen as the template that best matches the digit, while still maintaining a high separation from the templates of all the other different digits.

Experimentation using this DTW procedure has shown that a slope constraint of $P = 1$ works best when selecting the reference templates. A slope constraint of $P = 2$ defines too rigid a path; the same digit spoken at twice the speed will never match (this is not uncommon in the collected data!). $P = 1/2$ gives the opposite problem; templates that are not close to the average length of of the digit may be chosen as a reference digit. However, $P = 1/2$ can be successfully used in matching an input template with the reference templates because it allows for greater variation in the matching process than a slope of $P = 1$.

Chapter 5

Speech Recognition using Neural Networks

5.1 Introduction

Humans recognize speech by applying an enormous body of knowledge to rapidly interpret the incoming audio signals. Even powerful supercomputers cannot compete with the human brain when the task is understanding a conversation involving many speakers over a considerable amount of background noise. All this interrelated knowledge used by humans ranges from the low-level acoustic features to high-level facts about the world and the speaker's intent.

A speech recognition system to mimic the functions of the human brain, which consists of several billion simple, inaccurate and slow processors is still well beyond the realms of current systems. The human ability to learn important speech knowledge automatically is also another major deficiency in today's systems. These ideas though, have led to the development of the *neural network* or *connectionist model* which has recently been applied to speech recognition tasks.

Neural network models represent knowledge *distributed* across many simple processing units rather than as individual units and rules. These simple processing units do not have any knowledge programmed into their functions; instead the knowledge is contained in the large number of connections or weights that link the processing units together. Each processing unit has its own internal function that examines its local information (i.e. weights and unit output levels) to compute its own activation or output level. This type of structure, although roughly based on a human neural model, is far from being as complex as a real neuron model. These simplified networks often contain features that are not biologically plausible, such as one of the methods used to train the neural network.

A particularly attractive feature of neural network models are their ability to implement very complex nonlinear mapping functions between the input and output data. Through altering the structure of the network (the layout and number of its processing units), some control is provided over the complexity of this mapping function.

Training a neural network to give a satisfactory output response for the set of training tokens is a very time consuming process, especially for the *back propagation model* which has been used for this project. The back propagation learning process incrementally adjusts the network's weights with each recognition pass of the training patterns. Training tokens are presented to the input layer, and after a forward pass of the network the output response of each of the processing units in the output layer is checked with the desired response. The difference between the actual response and the desired response of the output layer is propagated back by adjusting the weight structure of the entire network, so that next time a similar input token is presented the output layer response will be closer to the correct response. This is known as *supervised learning*, and it is this *external* weight adjustment procedure that contains global information on the entire network that is a biologically questionable training model. The computational requirements for training a neural network are proportional to the size of the network and the number of training tokens presented to the network [11][12].

In this chapter the neural network structure used for the project experiments is discussed, followed by the features and operation of the neural network package used for this project, "GRADSIM".

5.2 The Temporal Flow Model

Speech communication is an inherently *temporal* process; time is the independent variable that orders the process of speech production and perception. Consequently, "the representation of time in a connectionist model designed to be used for speech recognition is a crucial issue" [13].

In the temporal flow model, the temporal relationship between the processing units is represented by a propagation delay. This propagation delay is encoded into the network by representing a single connection between two processing units as a *set* of links with each link specifying a different delay value. Through the use of these delay links, relationships between events in time can be expressed because of the range of time samples available to each processing unit. Formation of general filters which can be used for feature detection and pattern matching are also possible in this temporal model [13].

5.3 The Neural Network Structure

Neural networks consist of many interconnected processing units that each contain a local processing ability. This local function is normally an identical function for every unit in the network. A typical unit function is the *sigmoid* output function [13] :

$$o_j(n) = (1 + \mathrm{e}^{-p_j(n)+\theta_j})^{-1} \tag{5.1}$$

where $o_j(n)$ is the unit output at time division n, $p_j(n)$ is its net excitation or the potential function, and θ_j is the unit threshold. The neural network structure in this project uses the sigmoid function for each unit. The potential function $p_j(n)$ examines the interconnection characteristics of the network with respect to unit u_j and for the temporal model is given by a function such as [13] :

$$p_j(n) = \sum_{i,d} w_{ijd} o_i(n-d) \tag{5.2}$$

where w_{ijd} is the connection weight from unit u_i to unit u_j, and d represents the propagation delay(s) along the connection.

Processing units in neural networks are often organised into *multiple layers*. Three types of layer exist; the input layer, the hidden layer or layers, and the output layer. As mentioned, the input layer contains the units on which the training tokens are presented, and the output layer contains the units that emit the final response of the network to the training token. Hidden layers are the layers between the input and output layers. Depending on the complexity of the network there can be zero, one or a number of hidden layers. In a standard back propagation network these layers are *feed-forward* — that is the links can only connect a lower layer to a higher layer, where the lowest layer is the input layer and the highest is the output layer. Some networks also allow *recurrent* links that specify a connection back to the same unit, or to another unit in the same layer.

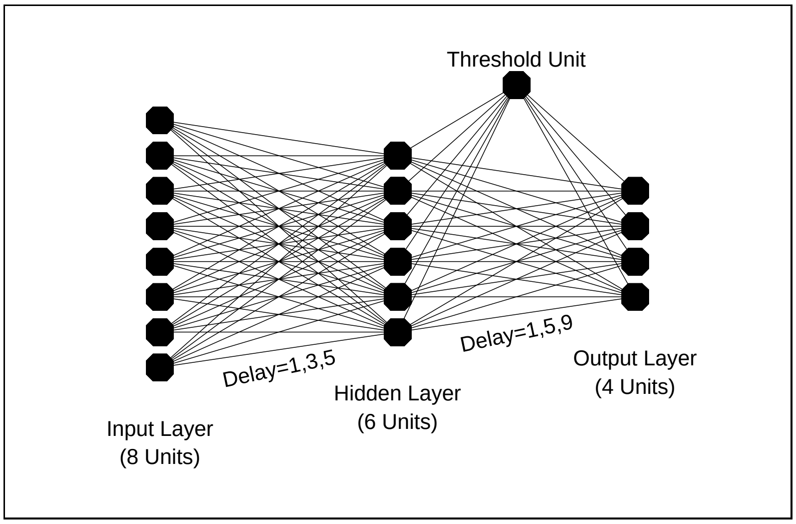

Figure 5.1: Structure of a Fully Interconnected 8-6-4 Network

For this project a network structure with a single hidden layer was used in all the neural network experiments. The interconnection topology between each layer is such that each node in one layer is connected to every node in the next layer, implementing a fully interconnected feed-forward network. Neural network architectures are often abbreviated in a notation that gives, for each layer of units, the number of units per layer. As an example a 8-6-4 network would consist of 8 input units connected to 6 hidden units connected to 4 output units. This network structure has been varied among the different experiments and their exact dimensions will be provided in Chapter 6.

The input layer is used to present each segment of an input template to the network. The temporal flow model presents the input vector for each segment (or time slice), one at a time to the input layer units. Therefore, the number of input units normally corresponds to the length of the input vectors.

The delay values for each connection have been kept static throughout all the experiments and these have been set at 3 delay links per connection. The delay links contain values of 1, 3 and 5 for the input to the hidden layer, and 1, 5 and 9 for the hidden to the output layer. This delay mechanism allows the hidden layer to scan across 4 input vector time slices to develop its representation. In a similar fashion the output layer scans 8 time slices of the hidden unit activations to develop its representation.

The output layer structure for this project consists of 10 units. One unit is allocated for each digit to be recognised. During the supervised training procedure, the single output unit that corresponds to the input template digit is set to give a positive response, while all the other output units are set to give a negative response. This negative or positive response is represented as a function of time and is also known as a *target function*. The target function that was chosen for the experiments is discussed in the next section.

One additional unit called the *threshold unit* whose activation is permanently set to one, is used to define the threshold values that determine if the output of each unit corresponds to a negative or positive activation. The threshold unit is connected to every unit in each of the hidden and output layers, and the threshold values are determined by the connection strengths. *Figure 5.1* gives a pictorial representation of this example. Note that each connection line contains three links, each containing the indicated delay values.

5.4 The GRADSIM Neural Network Simulator

The GRADSIM neural network simulator [14] was developed by Raymond L. Watrous to provide a connectionist simulator for use in speech recognition research using the temporal flow model. GRADSIM was developed from an early version of the "Rochestor Connectionist Simulator" [15], and was streamlined for speed providing faster *second order* learning methods for nonlinear network optimization. These methods include the Davidon, Fletcher and Powell (DFP) method and the Broyden, Fletcher, Goldfarb and Shanno (BFGS) method. GRADSIM is written entirely in the C language, and operates in the UNIX[1] environment. The latest version, GRADSIM V3.0 (May 1991) [16] was used for this project.

The operation of GRADSIM involves creating textual description files of the neural network structure (network descriptor), the list of training token filenames (experiment descriptor), and a parameters file to control various options. The network descriptor identifies the number of input units, each output unit and the threshold unit, followed by every link in the network which is given on a separate line. Each link is specified by a four tuple — the *from* unit, the *to* unit, the delay value, and the initial weight. The experiment descriptor contains details of the type of data stored in each token file (e.g. number of fields and data format), the target function for the output units, and a list of the training token filenames with their desired activation levels for each output unit.

5.4.1 Token File Format

The templates discussed in Chapter 4 are identical to the templates presented to the neural network. Each token file is constructed from the segments within a template. A text file is created, containing the input vector of coefficients for each segment, with one vector of coefficients output per line in a "double" floating point format. The number of coefficients per segment is known as the *field size*, and this size must correspond exactly to the number of input layer units.

It has been demonstrated that Neural networks train faster if the coefficients in the token file are distributed about zero, and range between -1 to $+1$. Normalization of this data is performed across the entire set of tokens by subtracting the average coefficient value and then scaling. Comparisons between the time to train the networks on the raw token files and the normalized token files showed that the normalized token files did indeed converge the network weights faster. This training token normalization has therefore been used in the neural network experiments.

[1] UNIX is a registered trademark of USL

5.4.2 Specification of the Target Function

An interesting feature of GRADSIM is the ability to customize the *target function* that is applied to the output units in both the training and the recall process. The target function can be thought of as a function of time (each time division consisting of a template segment), that specifies the expected value of each output unit for each time division. Target functions supported by GRADSIM include the linear, square, sigmoid, and Gaussian functions. Limited experimentation was given to testing each of these functions, and surprisingly the simple linear function appeared to give the best network recall results. The linear function was chosen as the output function for the experiments, so only this target function will be discussed. The linear target function, t, is expressed in terms of a normalized time index $v = n/S$, where S is the number of segments in the token and n is the segment index from $1..S$. A *positively* activated output unit is given by the following :

$$t(v) = 0.5(1 + v) \tag{5.3}$$

and for a *negatively* activated output unit :

$$t(v) = 0.5(1 - v) \tag{5.4}$$

It can be seen that this is a simple ramp function centred at 0.5. For a positively activated output unit this ramp function increases linearly from 0.5 at the start of the token, to 1.0 at the end of the token. In a similar fashion the negatively activated output function decreases from 0.5 to 0.0.

5.4.3 Testing the Network

Testing the neural network is done by calculating the MSE (mean squared error) from the difference between the actual output values and the required target function values. The MSE value is used to check the activation of each output unit in the testing process.

In the recall process to find out which digit is the spoken digit, each of the ten hypotheses that a particular output unit (corresponding to one of the digits) contains a positive activation is checked. For the hypotheses, each output unit in turn is assumed to contain a positive activation, while every other output unit is assumed to contain a negative activation. Forward passes of the network are performed through the segments of the input template, and the MSE of the actual activation of this output unit and the positively activated target function is calculated. The hypothesis with the smallest MSE is chosen to be the digit spoken.

A program was written to automatically create the descriptor files from the digit templates. Initially the training process perturbs all the network links with random weights between -1 and $+1$. Every training token is then read in and presented to the network for each pass of the network optimization process. The BFGS optimization method was chosen for the project experiments, and the parameters file contained the options to dump the entire network weights at periodic MSE points, so each saved network at different optimization levels can be tested.

Chapter 6

Speech Recognition Experiments and Results

This chapter is divided into two sections. The first section describes the speech recognition experiments undertaken for this project and the methodology used to evaluate each experiment. The nature of the experiments and the final implementation details are presented.

Keeping two separate sets of input data is an important issue in experiments such as speech recognition, as a speech recognition system is of no practical use if it cannot reliably recognise templates it has not been trained on. The available templates are divided into two sets; one set is used for creating the reference templates or in training the neural network, and another separate set is used for testing each system.

Comparison between the two system approaches demands a careful set of evaluation criteria because of their diverse nature. While the recognition performance is the best measure of how useful the system is, other factors such as the relative storage requirements and the system execution speed are also important.

One goal of this project was to investigate two categories of speech recognition; *speaker dependent* recognition and *speaker independent* recognition. In a speaker dependent system only templates from a single speaker are considered for recognition, a speaker independent system must consider templates from multiple speakers for recognition. As can be imagined, the speaker independent task is a more difficult one because of the wide speaker variation. Customisations of each system needed in attempting to cope with this latter task are also discussed in the first section.

In the second section of this chapter the results of the sixteen experiment variations are presented. Within each of the sixteen experiments the recognition results are presented for the three sets of utterances: the test set, the training set, and the alternative test set. The alternative test set is the test set from the other experiment category (either the single speaker or the multiple speaker set). The template and neural network results are discussed in separate sections, and then a brief comparison between the two is given.

6.1 The Speech Recognition Experiments

6.1.1 Splitting the Input Data into Two Separate Sets

The database of collected speech templates is partitioned into two sets so that a set of fresh templates is available for testing the systems; these are the *test set* and the *training set*. Of the 160 templates collected for the single speaker, half of them are kept aside for the speaker dependent test set which leaves 80 for the training set. The test set for the speaker independent case also takes out 80 templates of the 160 templates collected from the four different speakers. This partitioning gives two examples of each digit from each speaker in both of the sets. Note that although four speakers are used for the speaker independent case, one of these speakers is the same as the speaker used for the single speaker test set.

6.1.2 System Evaluation Criteria

Evaluation of each recognition system will be based on a number of criteria. Recognition accuracy is the most important measure of system performance. Four categories of experiment were used, template and neural network experiments for both the speaker dependent and speaker independent recognition tasks. Each of the above categories can be broken down further into four sub-categories corresponding to the types of input vector used in the template creation. Recognition percentages are quoted for each of these sub-categories. For the best method in each category it is also useful to present a "confusion" matrix, which tabulates the actual errors made. A confusion matrix is a good method for finding trends in the incorrectly recognised words, especially when there are a significant number of recognition errors.

The training set and the two test sets (speaker dependent and independent) are available to test each system. Obtaining recognition results for all three of these sets when evaluating each system may highlight weaknesses or strengths within each category. For instance if a speaker independent system achieves a high recognition rate on its test set, but achieves much poorer performance on the speaker dependent test set than its other sub-categories, this may indicate a weakness. It may also be found that one of the input vector types for the speaker dependent systems achieves consistently higher rates when applied to the speaker independent test set, indicating that this input vector type may contain less speaker dependent information.

The effectiveness of the two approaches (template and neural network) also needs to be evaluated using a set of measures. The relative execution speed and the storage requirements are the two main considerations in determining how applicable each system would be if adapted to a real environment. In this evaluation, both the system training and recognition steps will be considered as separate components.

6.1.3 The Experiment Categories

Table 6.1 tabulates the sixteen experiments that were performed for this project. The experiment names in the table will be referred to in this section to avoid long descriptions of each experiment. The "D", "I", "T" and "N" used in the names are abbreviations for speaker dependent, speaker independent, template matching, and neural network respectively. First the implementation details common to each category will be discussed, followed by sections describing each system architecture.

Table 6.2 shows the parameters chosen for each sub-category in the template construction process. The input vector length for the cepstral coefficients was set at 13 because of the

	Speaker Dependent		**Speaker Independent**	
Input Segment Type	*Template*	*Neural Net*	*Template*	*Neural Net*
Linear Filter Banks	DT.1	DN.1	IT.1	IN.1
Mel-Freq Filter Banks	DT.2	DN.2	IT.2	IN.2
LPC Coefficients	DT.3	DN.3	IT.3	IN.3
Cepstral Coefficients	DT.4	DN.4	IT.4	IN.4

Table 6.1: Names of the Experiment Variations

Input Segment Type	*Vector Length*	*Segment Size*	*Segment Duration (ms)*
Linear Filter Banks	15	512 samples	46.5ms
Mel-Freq Filter Banks	15	512 samples	46.5ms
LPC Coefficients	15	256 samples	23.3ms
Cepstral Coefficients	13	256 samples	23.3ms

Table 6.2: Vector Lengths and Segment Sizes for each Experiment Sub-Category

requirements to be a few coefficients less than the length of the LPC vector used in the calculation process. Preliminary experiments varying the segment size showed that for the FFT based input vector types (linear filter banks and mel-frequency filter banks), a larger segment size produced consistently higher recognition results. This is most certainly due to the FFT routines needing more samples within each window to compute an accurate vector. The LPC and cepstral segment sizes were kept at 256 samples as no significant recognition increase was found. Note that these segments for the templates are taken from the 11kHz sampling after filtering, and not the raw 22kHz samples. Another important note is that although there are two segment lengths, the segment windows are computed at intervals of 128 samples (11.6ms) for every sub-category.

Template Matching Experiments

For the speaker dependent experiments DT.1 to DT.4, a slope constraint of 1.0 and a window size of 20 was used for the DTW procedure in the creation of the reference templates. One reference template was chosen from the training set for each digit to give a total of 10 reference templates. The reference set was chosen using the method described in *Section 4.4* to choose the single best digit template. For the recognition testing, the DTW parameters were relaxed to ensure a greater chance of a match with any given reference template. A slope constraint of 0.5 was used with a window size of 40 for the testing.

For the speaker independent experiments IT.1 to IT.4 the same slope constraint of 1.0 and a window size of 20 was used for the reference set creation. However the method used to choose the reference templates was modified. An "ad hoc" template selection procedure was designed that chooses two reference templates for one digit, to give a total of 20 reference templates. The selection procedure obtains these two reference templates by taking the best matching digit from the female digit templates and the best digit from the male digit templates. A more sophisticated clustering technique combined with template averaging would almost certainly produce better results, but this simple method required very little extra work as the same procedure in *Section 4.4* only needed to be adapted to handle the two gender partitions. Also, it seems plausible that a clustering procedure would group the male and female templates into separate groups anyway. More preliminary tests indicated that relaxed DTW parameters caused the recognition rate to suffer, probably due to confusion arising from the doubled number of reference set templates. Subsequently, a slope constraint of 1.0 was used with a window size of 30 for the recognition testing.

Neural Network Experiments

Two neural network structures were used in these experiments. The first, and smaller one, was used for the speaker dependent experiments, DN.1 to DN.4. A 15-12-10 neural network was constructed, trained, and then tested for these experiments. To accommodate for the smaller length cepstral vectors, an additional 13-12-10 network was constructed with two less input units. One hidden layer with 12 hidden units was considered sufficient to encode the single speaker problem given the temporal properties represented by the time delay links. The time delay link structure was the same as mentioned in *Section 5.3* with each connection containing three delay links.

Testing each system uses the method outlined in *Section 5.4.3* that tests each of the ten hypotheses. The recognition recall of the networks at each of the saved MSE (mean squared error) points was then checked. Next the MSE range with the highest recognition rate was found. For some of the experiments, GRADSIM was re-run to save the network at a greater number of MSE points within this smaller range. The resulting network link file with the highest test set recognition rate is taken as the "best" optimized network, and this same network is used for checking the training and alternative test sets.

The second neural network is used for the speaker independent experiments. The only difference is that the number of units in the hidden layer is doubled to 24. The intuition behind choosing the increase of a factor of two, was the idea that if needed, the network could use these additional units for differentiating between the male and female input.

6.2 Results

6.2.1 Template Matching Experiment Results

	Speaker Dependent			Speaker Independent		
Input Type	*Test Set*	*Train Set*	*Multiple Set*	*Test Set*	*Train Set*	*Single Set*
Linear Banks	96.3%	98.8%	67.5%	80.0%	81.3%	90.0%
Mel-Freq Banks	91.3%	97.5%	75.0%	70.0%	75.0%	87.5%
LPC Coeffs	91.3%	92.5%	45.0%	63.7%	63.7%	78.8%
Cepstral Coeffs	100.0%	100.0%	61.3%	86.3%	85.0%	96.3%

Table 6.3: Recognition Percentages for the Template Matching Experiments

Table 6.3 tabulates the recognition results for the eight template matching experiments as tested on the three test sets. The "multiple set" and "single set" refer to the alternative multiple speaker and single speaker test sets.

Cepstral coefficient templates consistently achieved the highest recognition rates on both the test set and the training set. Surprisingly, the linear filter banks obtained significantly higher recognition rates than the mel-frequency filter banks, especially for the speaker independent experiments where up to a 10% difference was found. This could be an indication that the log spaced filter banks were spaced too far apart at the high end of the frequency range, or that more mel-spaced filter banks were needed. The LPC coefficient templates produced the lowest recognition rates, possibly because of the simple Euclidean measure that was used to obtain the distance between two LPC vectors. Other distance measures can be used such as the "minimum prediction residual" by Itakura [17] that makes use of a *log likelihood ratio* in the comparisons. This method may be more suitable as an LPC distance measure because of the complex feature space spanned by LPC.

The training and test set performances are very similar indicating that for a template system the test set error rate will usually be similar to the training set error rate. When the speaker dependent reference templates were tested on the multiple speaker test set, not surprisingly, lower recognition rates were found. However, test templates from three more speakers are added to this set, so this shows that the speaker dependent reference templates must contain significant information that is also common to the multiple speakers to obtain recognition rates of 60-75%. The single speaker test set when applied to the speaker independent reference templates, generated significantly higher results. This is almost certainly because the reference template selection process has selected more of the templates that pertain to the same speaker that was used for the single speaker test set.

6.2.2 Neural Network Experiment Results

In the neural network training process, as mentioned in *Section 6.1.3* the network is saved at various output unit MSE (mean squared error) values. In *Figure 6.1* and *6.2* plots are shown of the MSE at each save point during the training process. More save points are shown in some of the lines because the network was re-run over a smaller range of MSE values to accurately find the highest peak in each line.

From *Figure 6.1* for the speaker dependent neural networks it can be seen that the cepstral coefficient, linear filter bank, and the LPC coefficient inputs eventually all achieve 100%

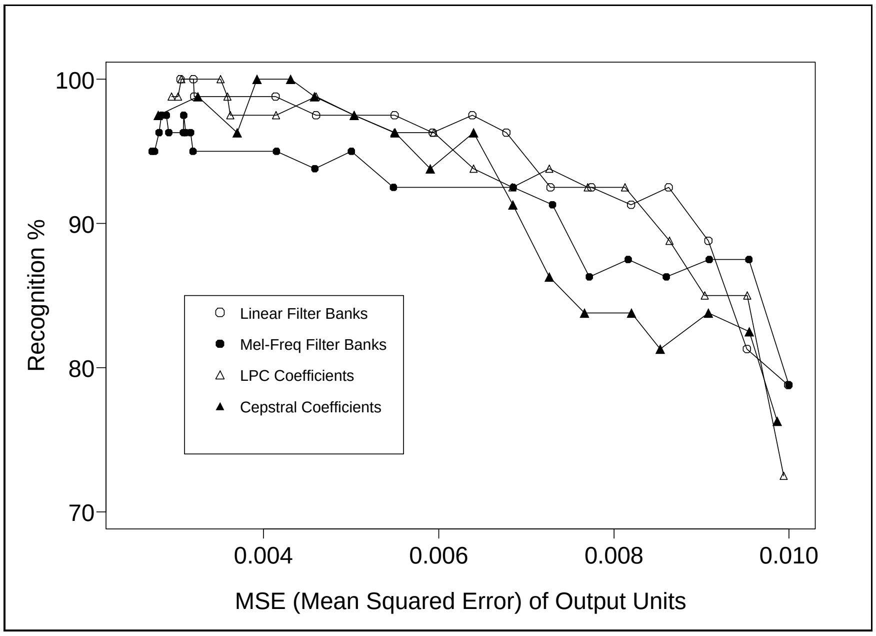

Figure 6.1: Speaker Dependent Neural Network Performance vs MSE during Training

recognition rates. The mel-frequency input does not appear to be a good representation for a neural network, as the recognition performance never matches that of the other input types.

Figure 6.2 shows the speaker independent neural networks during training. Here the linear filter bank and cepstral coefficient representations are clearly the preferred form of input when catering for multiple speakers, both attaining over 90% recognition rates. Mel-frequency filter banks and LPC coefficients perform poorly compared to the other two input types, possibly because of similar reasons to the template matching experiments.

A property of neural networks using back propagation training is that eventually a stage in the training process is reached where the network performance on a fresh test set of examples deteriorates. This is particularly evident in *Figure 6.2* where the performance of the network drops off at about the MSE=0.004 point. This is known as the *generalisation* problem and occurs at a point where the network weight structure starts remembering each exact training template, rather than its general features.

	Speaker Dependent			**Speaker Independent**		
Input Type	*Test Set*	*Train Set*	*Multiple Set*	*Test Set*	*Train Set*	*Single Set*
Linear Banks	100.0%	100.0%	41.3%	92.5%	100.0%	92.5%
Mel-Freq Banks	97.5%	100.0%	50.0%	83.8%	100.0%	86.3%
LPC Coeffs	100.0%	100.0%	37.5%	73.8%	100.0%	83.8%
Cepstral Coeffs	100.0%	100.0%	42.5%	91.3%	100.0%	93.8%

Table 6.4: Recognition Percentages for the Neural Network Experiments

Table 6.4 tabulates the recognition results for the eight neural network experiments as tested on the three test sets. An interesting difference between the neural network architecture and the template matching method is that the neural network is able to easily encode information into its structure to correctly differentiate between all the training set templates. Testing the

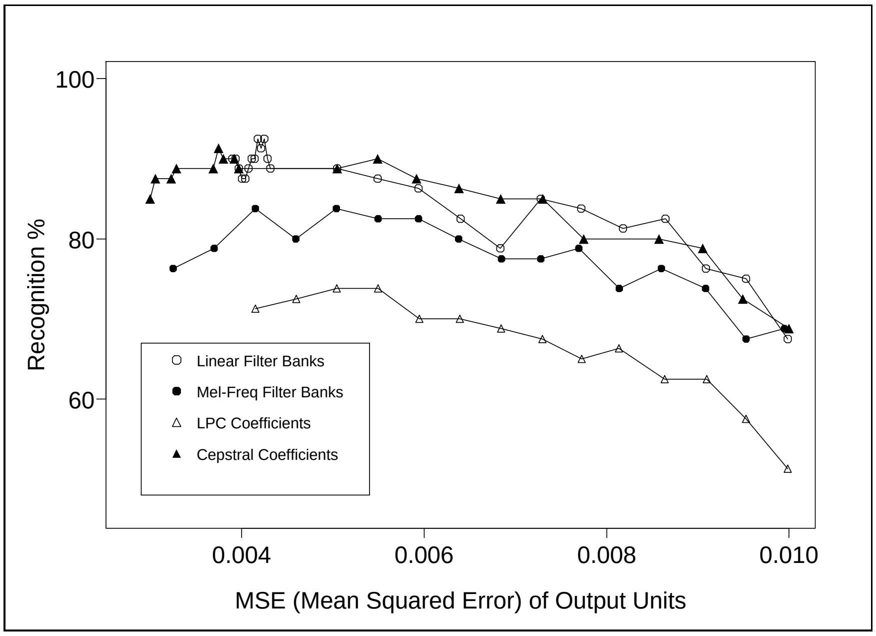

Figure 6.2: Speaker Independent Neural Network Performance vs MSE during Training

speaker dependent neural networks on the multiple speaker set highlights another interesting feature of the neural network architecture. Even though the neural networks attain significantly higher performance on the test set than the template matching experiments, they are not as capable when applied to the multiple speaker set. The reason for this is that the neural networks have only had a chance to encode information about one speaker, while the dynamic time warping procedure is able to adapt its reference templates to the multiple speaker set more successfully.

The speaker independent neural network experiments achieve significantly higher results on the test set than the same template matching experiments. This is not surprising since the network structure has encoded information about every speaker's digits, in contrast to the template matching system that only contains reference templates for one quarter of the training set templates. A smarter reference template averaging process for the template matching system would prevent losing some of this extra information.

6.2.3 Recognition Errors for the Speaker Independent Experiments

Table 6.5 gives a confusion matrix for the best template matching experiment from the speaker independent experiments, which uses the cepstral templates. The matrix shows a number of recognition errors that occurred in groups of two. This highlights a deficiency in the reference template selection procedure because the three occurrences of these double errors were errors in digits spoken by the *same* speaker. These errors arose because the reference templates chosen pertain to the other speaker of the same gender, giving more evidence that the particular templates chosen as the reference templates were not general enough. Greatest confusion was caused by the digits "zero" and "nine", followed by "four" and "six" in this template experiment.

In the same fashion *Table 6.6* gives a confusion matrix for the best neural network experiment, which used the linear filter bank input. No single category has two errors like *Table 6.5*, but

Input	**Recognised Digit**									
Digit	*zero*	*one*	*two*	*three*	*four*	*five*	*six*	*seven*	*eight*	*nine*
zero	5			1				1		1
one		8								
two			8							
three				7					1	
four		2			6					
five						8				
six							6	2		
seven								8		
eight									8	
nine		2				1				5

Table 6.5: Confusion Matrix for Cepstral Template Matching Experiment : IT.4

the digit "eight" was incorrectly recognised twice. The neural network has also distributed the errors more evenly than the template experiment, although only half the number of errors were made. With the shorter length digits, "two", "eight" and "six" comprising two thirds of the errors, this could indicate that an optimised network with adjusted delay values may give a lower error rate.

Input	**Recognised Digit**									
Digit	*zero*	*one*	*two*	*three*	*four*	*five*	*six*	*seven*	*eight*	*nine*
zero	8									
one		7								1
two	1		7							
three				8						
four					8					
five						7				1
six							7	1		
seven								8		
eight		1		1					6	
nine										8

Table 6.6: Confusion Matrix for Linear Filter Bank Neural Network Experiment : IN.1

6.2.4 Comparison between the Two Systems

The hardware platform used for both the systems consisted of a Silicon Graphics IRIS 4D, containing four processors and 64M of main memory. To put the timing comparisons into perspective, roughly 30MIPS of power was available for each experiment. The execution times and storage requirements of the template matching systems are given first, followed by the neural network systems, and then a comparison between them is presented.

The template matching system takes about 3.5 minutes to select the 10 reference templates for the speaker dependent experiments, and about 2 minutes to select the 20 reference templates for the speaker independent experiments. Less time is taken to select the reference templates in the speaker independent experiments because the available templates are divided into two gender sets resulting in fewer DTW template comparisons. The storage requirements are constant at 1.4M of memory as all the available templates are kept in memory for the DTW comparisons. Testing the recognition rate of the test set requires the same 1.4M of memory, and takes 50 seconds for the speaker dependent experiments and 80 seconds for the speaker independent experiments.

For the neural network experiments, the time taken to optimize the speaker dependent networks varied from 5-10 hours, while the speaker independent networks needed 18-36 hours. The network that optimized fastest in both categories used the cepstral coefficient inputs. The BFGS algorithm used for the weight adjustment requires square matrices of network links, and since there are about 900 links in the speaker dependent networks, 8M of memory is required. Likewise, there are about 1800 links in the speaker independent networks, and these require 28M of memory. Testing only requires 400K, and takes 8 seconds for the speaker dependent networks, and 15 seconds for the speaker independent networks.

The time needed and the memory required is far greater for the neural network training process than for selecting the reference templates, as the neural network learns to categorise the various input patterns very slowly. On average about 900 iterations of the network were needed before the best test set performance was obtained. The neural network memory requirements can actually be trimmed to less than 500K if a different gradient optimization algorithm supplied with GRADSIM such as the "conjugate gradient" method is used instead. These *memoryless* algorithms only require vectors of links and also take significantly longer to train, but some compromise must be made if 28M of memory is not available. Also, the memory required for the template matching system can be pruned as the DTW distance between two templates can be file based, so that only two templates are in memory at a time for each comparison. Only about 400K is required when just the 10 or 20 reference templates are in memory for checking against the test set.

Performance of the neural network in checking the test set templates is much faster than the template matching routine, and this is potentially more important in a real time system where the time needed for training or setup can be performed "off-line". The neural network achieves a 400% increase in performance over the template system when attempting to recognise a new template, and this is because a forward-pass of the network is faster than computing the DTW distance between an input and *every* reference template.

6.2.5 Comparison with other Recognition Systems

Rabiner and Schafer [4] quote references for isolated word recognition systems that achieve digit recognition on the order of 98-100% for a single speaker, and about 95% for a speaker independent system. The systems implemented for this project certainly attain the speaker dependent rate, but fall slightly short of the quoted speaker independent rates.

A speaker independent telephone speech recognition system developed by Hirosi Iizuka [18] reports a 95.8% recognition rate. This system uses template matching and the vocabulary consists of the Japanese digits 0-9 and the letters A-F. Reference templates are generated using a smart reference pattern updating procedure that allows erasure, averaging, and addition of templates. 128 reference templates were generated in separate systems for male and females consisting of 340 male and 230 female speakers. Even though Japanese digits were used, the vocabulary size is similar, and this system shows that a finely tuned speaker independent template matching system is capable of a high recognition rate.

In a recent paper by J.R. Kirkland and Dr K.L. Garden (University of Canterbury) [19], a very simple neural network was trained for one speaker using 16 examples of each digit (8 for training and 8 for testing). Recognition accuracies of 94% were reported for this system, and although the neural network system discussed in this report achieves a significantly higher rate, the temporal delay neural network structure and the neural network input format that was used is more complex.

Chapter 7

Conclusion

This project has achieved all the original goals I set, and the results from the constructed systems were very successful. Speaker dependent isolated word recognition is a relatively easy task given a limited vocabulary, in contrast to the complex speaker independent task that presents problems due to the wider speaker variation.

The template matching approach works well as long as reference templates are chosen that are good representatives of the words to be recognised. A worthwhile improvement to the reference template selection for the speaker independent experiments would be to average similar templates, and to use a clustering procedure to chose the best representatives.

The neural network experiments produced higher recognition rates than I had initially expected, and I suspect this was due to the temporal model that was used to isolate temporal features from the input vectors. It appears that background noise and meaningless variations in a word's frequency components can be ignored by a neural network structure, while important distinguishing features of the training templates are retained. Also, the neural network had not been told any specific information about these features, and yet through the use of a standard network optimization technique it was automatically able to perform correct classifications.

From the experiments the cepstral and linear filter bank information gave the best results. Cepstral processing was well suited to the template matching experiments, producing the highest results in both categories. An interesting feature of the neural network results was that both the linear filter banks and the cepstral input produced equally good results. From this it appears that a neural network can easily adapt according to the type of input — whether raw frequency components (linear filter banks) or the result of a complex transformation (cepstral processing).

Similar memory requirements are needed for both of the systems, as long as the alternative memoryless neural network training algorithms are used, making both the systems applicable to smaller machines (less than 1M of memory). If desired the computationally expensive network optimization step could be performed on a powerful machine, and the resulting network weights could then be transferred to the smaller machine for recognising the new input templates. Neural networks have great potential in speech recognition, as results equal or better to conventional approaches can be obtained. They also seem to be less prone to segmentation errors that arise from the automatic segmentation process — perhaps this is due to the propagation delay structure. Neural networks are able to store useful information about all the training templates, as opposed to the template matching approach which usually holds only a subset of the training templates. Also, recognising new input templates using a trained network is significantly faster than the template matching DTW algorithm.

7.1 Possible Extensions

Given more time, I would like to fine tune the neural network structure by adjusting the number of hidden units and investigating the addition of more hidden layers. Possibly a smaller neural network structure with fewer links can be developed that still produces similar results. Different delay values could also be tried. It would also be interesting to try a separate neural network for each of the two genders for the speaker independent task. To improve the template matching systems a much smarter reference template selection process could be designed.

Some experimentation could be tried using recurrent links and adjusting the delay values of the network in an attempt to improve recognition. I also believe that more speaker templates should be collected for the systems, especially for the neural network training set — this would almost certainly improve the generalisation of the network, and make the network less susceptible to any bad training examples. The classification errors made by the neural network could be investigated further by examining their actual target response, which could lead to a customization of the target function.

Finally, I think the greatest challenge would be adapting one or both of the systems discussed to a real-time environment. A machine of moderate power would be required that was capable of sampling the spoken speech, and was fast enough to compute the various speech signal transformations quickly.

Acknowledgements

Many thanks to my Honours supervisor Steve Cassidy for his support and guidance throughout the year. Thanks also to Guszti Bártfai for useful comments and ideas relating to neural networks, and the idea of the zero crossing with threshold rate.

A special mention goes to Ray Watrous from Siemens Corporate Research who provided invaluable support on the use of his neural network package, GRADSIM, both through the use of email responses and the very prompt despatch of two of his own papers.

Thanks also to Tony Robinson from Cambridge University who kindly provided the short LPC routines.

Thanks again to each of the recording subjects that were forced to speak into a Macintosh microphone for an indeterminate period of time.

Finally, I would never have made it through the year without the help of my family (special thanks to my mother) and all the other fellow graduate students.

Bibliography

[1] W. A. Ainsworth. *Speech Recognition by Machine*, volume 12 of *IEEE computing series*. Peter Peregrinus Ltd, 1988.

[2] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T.Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988. Example book in C, and Diskette also available.

[3] Gary R. Dougherty William D. Stanley and Ray Dougherty. *Digital Signal Processing (2nd Ed)*. Reston Publishing Company, Inc., Reston, Virginia 22090, 1984.

[4] L.R. Rabiner and R.W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall International, 1978.

[5] Steven B. Davis and Paul Mermelstein. *Comparison of Parametric Representations for Monosyllabic Word Recognition in Continously Spoken Sentences*, chapter 3, pages 65–74. Readings in Speech Recognition. Morgan Kaufmann Publishers, Inc., 2929 Campus Drive, San Mateo, California, 1990.

[6] P. M. Grant. Speech recognition techniques. *Electronics and Communication Engineering Journal*, 3(1):37–48, February 1991.

[7] Lawrence R. Rabiner and Bernard Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall International, 1975.

[8] Jared J. Wolf. Speech signal processing and feature extraction. *Spoken Language Generation and Understanding*, pages 103–128, 1980.

[9] G. Bártfai. Adatátviteli célú beszédszünet detekció (Speech silence detection for data communications purposes). Technical report, Technical University of Budapest, Hungary, 1987. Doctoral thesis (in Hungarian).

[10] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, February 1978.

[11] Robert Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, 1990.

[12] Alex Waibel and Kai-Fu Lee. *Readings in Speech Recognition*. Morgan Kaufmann Publishers, Inc., 2929 Campus Drive, San Mateo, California, 1990.

[13] Raymond L. Watrous. Phoneme discrimination using connectionist networks. Technical report, University of Pennsylvania, August 1989.

[14] Raymond L. Watrous. Gradsim : A connectionist network simulator using gradient optimization techniques. Technical Report MS-CIS-88-16, University of Pennsylvania, March 1988.

[15] Toby Mintz Nigel H.Goddard, Kenton J. Lynne and Liudvikas Bukys. Rochestor connectionist simulator. Technical Report 233, University of Rochestor, October 1989.

[16] Raymond L. Watrous. Gradsim 3.0 users guide. Technical report, Siemens Corporate Research, May 1991.

[17] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975.

[18] Hirosi Iizuka. Speaker independent telephone speech recognition. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2:842–845, 1985.

[19] J.R. Kirkland and Dr. K.L. Garden. Neural magic and speech recognition. *Proceedings of the 28th National Electronics Connference, NELCON 91*, 28(ISBN-0-473-01292-8):42–47, August 1991.

[20] David B. Roe Stephen E. Levinson. A perspective on speech recognition. *IEEE Communications Magazine*, pages 28–34, January 1990.

[21] C. H. Chen. *Digital Waveform Processing and Recognition.* CRC Press, 1982.

[22] K. C. Pan, F. K. Soong, L. R. Rabiner, and A. F. Bergh. An efficient vector-quantization preprocessor for speaker independent isolated word recognition. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2:874–877, 1985.

[23] Satoru Fujii Shuji Morii, Katsuyuki Niyada and Masakatsu Hoshimi. Large vocabulary speaker-independent japanese speech recognition system. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2:866–869, 1985.

[24] Mark A. Yoder and Leah H. Jamieson. Simulation of a highly parallel system for word recognition. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3:1449–1452, 1985.