A SPEECH RECOGNITION SYSTEM USING A NEURAL NETWORK MODEL FOR VOCAL SHAPING

by

Christopher D. Love

A Thesis

presented to the University of Manitoba

in partial fulfillment of the

requirements for the degree of

Master of Science

in the

Department of Electrical and Computer Engineering

Winnipeg, Manitoba

March 1991

A SPEECH RECOGNITION SYSTEM USING A NEURAL NETWORK MODEL FOR VOCAL SHAPING

BY

CHRISTOPHER D. LOVE

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

ABSTRACT

Computerized vocal shaping systems are of considerable value and interest. This study is motivated by the need to improve: (i) recognition of previous vocal shaping systems in order to reduce disagreement between computer and human evaluator, (ii) multi-level classification using five levels that correspond to human assessment, (iii) real-time recognition through the use of neural networks, and (iv) the human interface to achieve efficient interaction through the Macintosh graphical interface. A system has been developed to recognize a limited set of isolated utterances on a finite scale in a quiet environment. A speech signal is first sampled and analyzed to obtain linear predictive coding (LPC) filter coefficients which are then presented as templates to a backpropagation neural network. The LPC coefficients provide a sufficient spectral feature for high accuracy isolated speech recognition. A standard backpropagation neural network is used to improve recognition accuracy, to reduce classification time, and to improve multi-level grading results from the past vocal shaping system. The backpropagation neural network uses an adaptive learning rule, which is based on the total internal network error measure and has been shown to reduce the network training time in obtaining a global solution. The experimental system includes a 20 MHz, 68030 based Macintosh IIsi computer with a MC68882 floating point coprocessor. The development software is C language (an object-oriented version). The new human interface developed for vocal shaping is an improvement from the previous model due to the configuration flexibility and ease of operation. The multi-level grading scheme and classification accuracy were evaluated using a vowel and consonant set which are based on the relative locations of the first two formants. The vowel set provided good recognition classification results (61.8%) while the consonant data sets resulted in slightly lower classification results (48%). The multi-level classification scheme provided good separation between utterance versions for the vowel set.

ACKNOWLEDGEMENTS

The author would like to acknowledge the many people who who contributed to this research. In particular, credit is given to my advisor Dr. W. Kinsner who helped guide me through the thesis process and helped me to understand the true meaning of research. Also, I would like to thank my fellow colleagues, Ken Ferens, Adi Indrayanto, and Armein Langi, who, without their support and encouragement, would have made this task so much more difficult. I also wish to thank Mr. J. Schnabl of Manitoba Telephone System for providing the noise cancelling microphone. Partial support for this work came from the University of Manitoba and NSERC is also acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

Figure **Page**

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

A/D	Analog to digital converter
ADPCM	Adaptive differential pulse code modulation
α	Representation of the momentum
ARMA	Auto regressive moving average
BAM	Bi-directional associative memory
BP	Backpropagation
CPU	Central processing unit
δ	Representation of the network error signal
DPCM	Differential pulse code modulation
DFT	Discrete Fourier transform
DSP	Digital signal processor
FFT	Fast Fourier transform
LMS	Least mean square
LPC	Linear predictive coding
η	Representation of the learning rate
PARCOR	Partial correlation
PCM	Pulse code modulation
RAM	Random access memory
SPB	Sound speech blocks

CHAPTER I
INTRODUCTION

Native speech is by far one of the easiest things for humans to perform. We seem to perform this task almost effortlessly and can deal with such problems as: (i) background noise, (ii) coarticulation, (iii) multiple speakers, (iv) speed and variability of speech, (v) context sensitivity, (vi) ambiguity, and (vii) temporal and phonetic variations. On the other hand, computer-based recognizers require near ideal environments in which many of these problems have been reduced to achieve good results.

1.1 Purpose

The goal of this research is to study, design, and implement an isolated, limited vocabulary, automatic, speech recognition system using a backpropagation (BP) neural network with linear predictive coding (LPC) coefficients. This system is intended to be used as a vocal shaping tool for autistic individuals.

The objectives of this work are to improve a previous vocal shaping system, specifically related to the problems of multi-level classification, system interface design and functionality, and recognition time. The improvement entails: (i) using a BP neural network with LPC coefficients, (ii) a modified multi-level utterance classification scheme for recognition, and (iii) a custom designed human interface employing Macintosh graphics techniques. Multi-level means that up to five non-linear levels of quality, ranging from unsatisfactory to excellent, are used to classify an utterance. The levels are non-linear since they are assigned at the time of recording based on the therapists judgement. This type of scheme is necessary for vocal shaping.

The earliest model of the vocal shaping system, developed on the Apple IIe, was targeted as a language translation tool and was later modified to perform the necessary functions of vocal shaping [KiPR86]. Other researchers independently developed systems (Indiana speech training aid project (ISTRA)) to perform similar goals [KMRW87] and, one even employed a BP neural network with LPC coefficients [DeBo90]. Difficulties with past systems were that the utterance classification was not multi-level, the grading results did not agree with their human counterpart, the recognition was not real-time, and the human interface was not intuitive enough. Thus, this work is concerned with addressing and solving these problems.

Contributions of this work include: (i) a neural network approach using BP to perform real-time isolated speech recognition for vocal shaping, (ii) an improved multi-level grading scheme, and (iii) a custom designed, highly configurable human interface.

1.2 Motivation

Computer speech recognition has already found many uses in society. One application is in computer-aided vocal shaping [Cair90], [Desr90]. This process requires accurate recognition and assessment over a multi-level scale and precise use of specific training procedures. "Administration of this process by computers may increase the precision of assessment and progress over therapists used alone" [Desr90]. The computer provides the therapists with a reliable, consistent, unbiased, and multi-level assessment of the individual's performance. Multi-level assessment is necessary since the individual may not be able to achieve the ideal target from the outset. This goal may only be achieved through the "administration of reinforcement for closer approximations of the response and extinction of previous approximations...until a specified target response is acquired by the student" [Desr90]. The multi-level assessment in the past was at the discretion of the computer based on the inherent rules it was provided. In this model, the therapist is provided the ability to decide on what is good or bad, in their opinion, and is further able to assign this credit to the recorded utterances. This important feature bridges the gap between the inconsistencies found in past versions of this type of recognizer.

For the intended recipients of this system, recognition of isolated, limited vocabulary, utterances is sufficient. In order to reduce the necessary intervention by a therapist to operate the speech recognition software, this system has been made *user friendly.* This means that the autistic individual can use it with reduced therapist supervision. It also corresponds to the amount of program automation and the way that ideas are presented by the speech recognition software. The Macintosh graphical interface provides an environment which facilitates these tasks. Another fundamental motive to do research in this area is because the end result will be applied to help other people, and in the end, this is what engineering is concerned with achieving.

1.3 Thesis Organization

An understanding of existing systems and the methods that they employ to currently perform the desired tasks are first investigated. Next, an evaluation of what new technologies will be used and how they will improve performance over the existing technologies is described. The next step in the thesis development is to assemble, verify, and test the system; describe the results; and provide conclusions based on the system's performance. Finally, conclusions on the overall course of the study are presented.

Chapter I provides the thesis goals, human motivation for vocal shaping, and technical reasons for designing and implementing an improved speech recognition system. Chapter II provides background on speech processing. A discussion of the human speech production system is followed by speech analysis techniques. LPC is focussed on amongst the other speech processing techniques. Next, a description of two features and two methods used in speech recognition is provided. The investigated features are the fast Fourier transform (FFT) and LPC. The two recognition methods described for use in isolated speech recognition are template matching and neural networks. Chapter III provides a detailed review of the BP neural network. A description of the classification technique, or multi-level grading scheme, used for recognition follows. Four metrics are used to determine the total internal network error. Chapter IV describes the system from a global viewpoint. The major components are described and their relationships are established. Also, two different methods of data acquisition are contrasted. The first method uses a NEC DSP in an IBM computer, and the second method uses a sound chip in a Macintosh computer. Current operation uses the first method involving the NEC DSP and IBM computer to acquire the data while using the Macintosh computer as the host. Chapter V describes the speech recognition system software. Software programs include: (i) the human interface, (ii) data acquisition routines, (iii) communication routines, (iv) a training method, (v) a testing method, and (vi) a recognition method. Chapter VI describes the design, execution, and the experiment results. The experiments are intended to evaluate the multi-level classification ability and classification accuracy of the BP neural network based on vowel sounds and consonant combinations. Conclusions and recommendations are contained in Chapter VII.

CHAPTER II
BACKGROUND ON SPEECH PROCESSING

2.1 Introduction

Human speech is the most widely used form of communication. It is therefore intuitive to create a way for machines to understand humans in their native language, not a language that *humans* must learn in order to understand their machines. Although this would be the ideal environment, human speech is much more difficult to capsulate into a single process or device than it is for us to understand the computers which we have created. "Humans are able to understand speech so easily that we often fail to appreciate the difficulties that this task poses for machines. As used by humans, speech is universally applicable, omni-directional, absolutely speaker-independent, highly noise immune, adaptively acquired, and *apparently effortless*" [LeRo90]. In order to successfully achieve speech recognition at any level, precise steps and procedures must be followed.

From the primitive knowledge base of speech recognition three fundamental principles have evolved. The first is concerned with intelligence. The intelligence contained in a speech waveform is encoded in the temporal variations of the short amplitude spectrum. The second principle is literacy. By this, speech can be completely capsulated by a finite domain of fundamental speech units. Finally, speech is a cognitive process. Hence, it can not be separated from the grammatical, semantic, or pragmatic structure of the language. Based on these three fundamental principles, a discussion of the properties and characteristics of speech processing and recognition is presented. This discussion will result in the selection of a sufficient spectral feature and isolated speech recognizer to be used in the isolated speech recognition system design.

2.2 Human Speech Production

As the first fundamental property of speech is intelligence, it is logical that speech begins as some abstract form of neural signals in our brain. The signals routed to the vocal apparatus contain information of how the vocal apparatus should be shaped and moved in order to produce the desired sounds. The vocal apparatus, shown in Fig. 2.1, includes the lungs, larynx, nasal tract, vocal chords, jaw, lips, tongue, and velum.

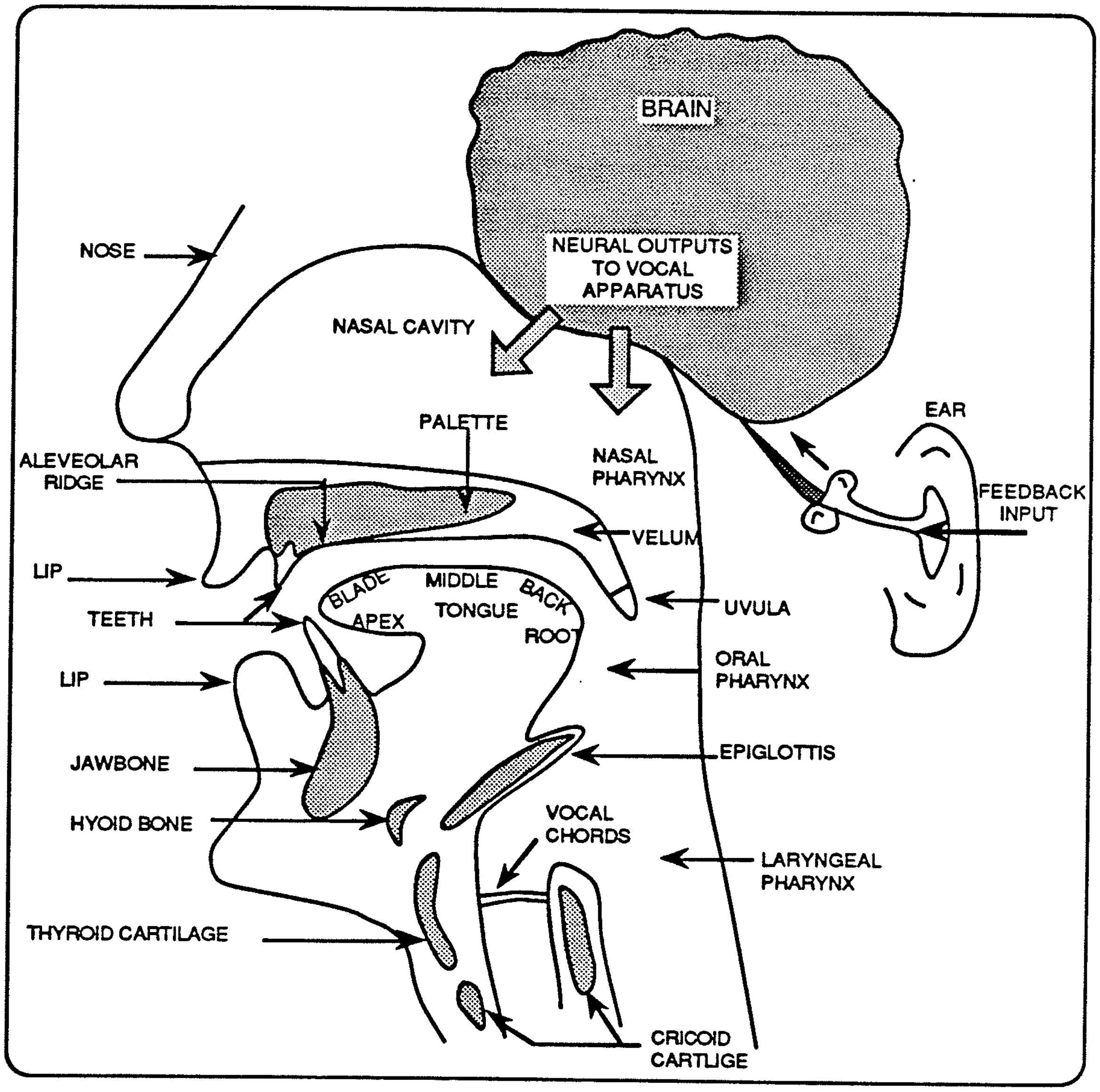


Fig. 2.1 Human speech apparatus (after [Pars86]).

The production of the desired sounds and their arrangement leads into the next fundamental property - speech literacy. This involves the ability to produce *meaningful* sounds given a language. This in itself is a very complex process. However, to simplify things, the speech production process is broken into two fundamental functions for English. The first function deals with excitation and the second with modulation.

Excitation is achieved through phonation, whispering, frication, compression, and vibration. If the vocal chords oscillate then the speech is considered to be phonated. The opening, closing, and air pressure across the vocal chords, and their shape, are all important parameters in determining the *pitch*. Pitch is the repetition rate of these pulses and is controlled mainly by the tension in the vocal chords, and also by feedback from the ears. As the vocal chords become more tense, the pitch will increase. Conversely, as the vocal chords become relaxed, the pitch will drop. When the vocal chords are drawn together, except for a small triangular opening in the vocal tract, air is allowed to rush through this opening generating turbulence. The wideband noise generated by this configuration of the vocal apparatus constitutes the excitation signal. This results in *whispering*, *consonants*, and *syllable boundaries*. Frication is caused when the vocal tract is restricted at any other point than the triangular section. The spectrum is mainly reflected by the point of vocal tract restriction and the sounds produced are called *fricatives* or *sibilants*. If the vocal tract is completely cut off while pressure continues to build, a small explosive burst will result when the vocal tract is finally opened. When the silence is followed by a release which is abrupt and breaks off quickly, such as [p] or [t], the sound is known as a *plosive* or *stop*. If the release drops off slowly, such as [tch], then the sound is known as an *affricate*. When air is forced through a closure other than the vocal tract, resonance may occur about the tongue, uvula, or between the lips, such as *whistling*. Sounds of this nature are known as *vibrations*.

The second function following excitation is modulation. This process describes how the adjustable vocal apparatus are modified to change from the current sound to the desired one. The ear provides feedback to the brain of what is being modulated. The brain sends neural signals to adjust the vocal parameters to correct the modulation of the excitation signal (refer to Fig. 2.1). Modulation is sometimes referred to as *articulation.* This leads into a very complex area of speech production understanding which linguists are concerned with and is known as *articulatory phonetics* – the knowledge of how all the vocal parameters are arranged to produce any sound. Physiologically, modulation is reduced to simply moving the vocal apparatus to change the quality of the excited sound. Acoustically, the main means of modulation can be thought of as a filtering process. The excited waveform contains many harmonics, due mainly to the vocal tract which is tubelike, and thus has many natural frequencies as shown in Fig. 2.2. These natural frequencies are called *formants*, and are the single most important modulating technique. Many sounds can be synthesized from knowing the values of the first three formants [Pars86].

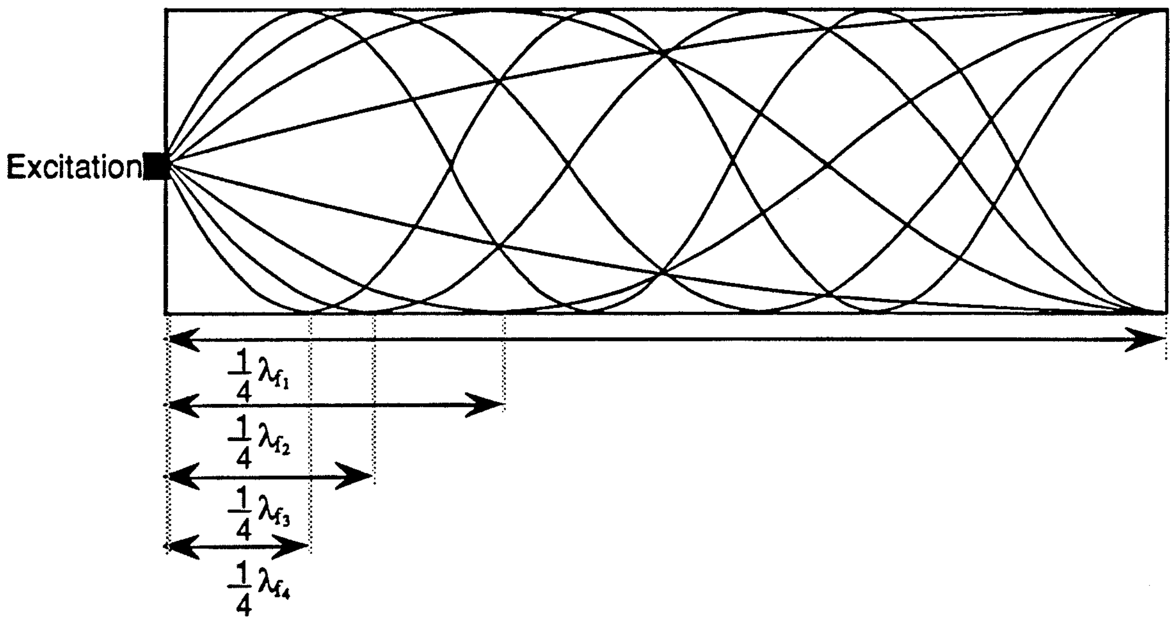

Excitation
$\frac{1}{4}\lambda_{f_1}$
$\frac{1}{4}\lambda_{f_2}$
$\frac{1}{4}\lambda_{f_3}$
$\frac{1}{4}\lambda_{f_4}$

Fig. 2.2 Vocal tract natural frequencies (after [Swan87]).

The desired sounds, once excited and correctly modulated, are carried through the air in the form of an acoustic pressure wave (APW). The speech information contained in the APW is encoded in the air molecule fluctuations. The APW is received and collected by the outer ear. The collection of sound waves vibrates the cochlea which stimulates the auditory nerve generating the neural signals that are sent to the brain and decoded into language.

A description of the vocal apparatus and a discussion of both vocal generation and perception is contained in [Pars86]. Now that the fundamental speech generation process has been provided, a study of the properties are presented to describe what characterizes the speech waveform and to determine which properties are most relevant to speech recognition. Speech waveform characteristics includes amplitude, resonant length, pitch, formant frequencies, bandwidth, silence, and stationarity.

The first noticeable characteristic of speech is its discrete nature. By this, "we know, but don't always acknowledge everyday, is that the utterances we produce consist of separate or discrete units of speech and that these units are concatenated to form words and sentences" [FeLo89]. Further, about half of all speech contains silence [RaSc78]. What is the cause of speech being discrete? The reason this occurs is, first, because humans have a fixed rate of articulation and, second, because words are separated by periods of silence due to the nature of the language. "For speech, a crude estimate of the information rate can be obtained by noting that physical limitations on the rate of motion of the articulators require that humans produce speech at an average rate of about 10 phonemes per second" [RaSc78]. This physical limitation is evident from observing the waveform provided in Fig. 2.3 which shows the silent intervals between utterances. The silent intervals are distinguished from the utterances by noting they have near zero amplitudes and are not boxed in by a shaded region. Each phoneme, or word segment, is boxed in by a shaded

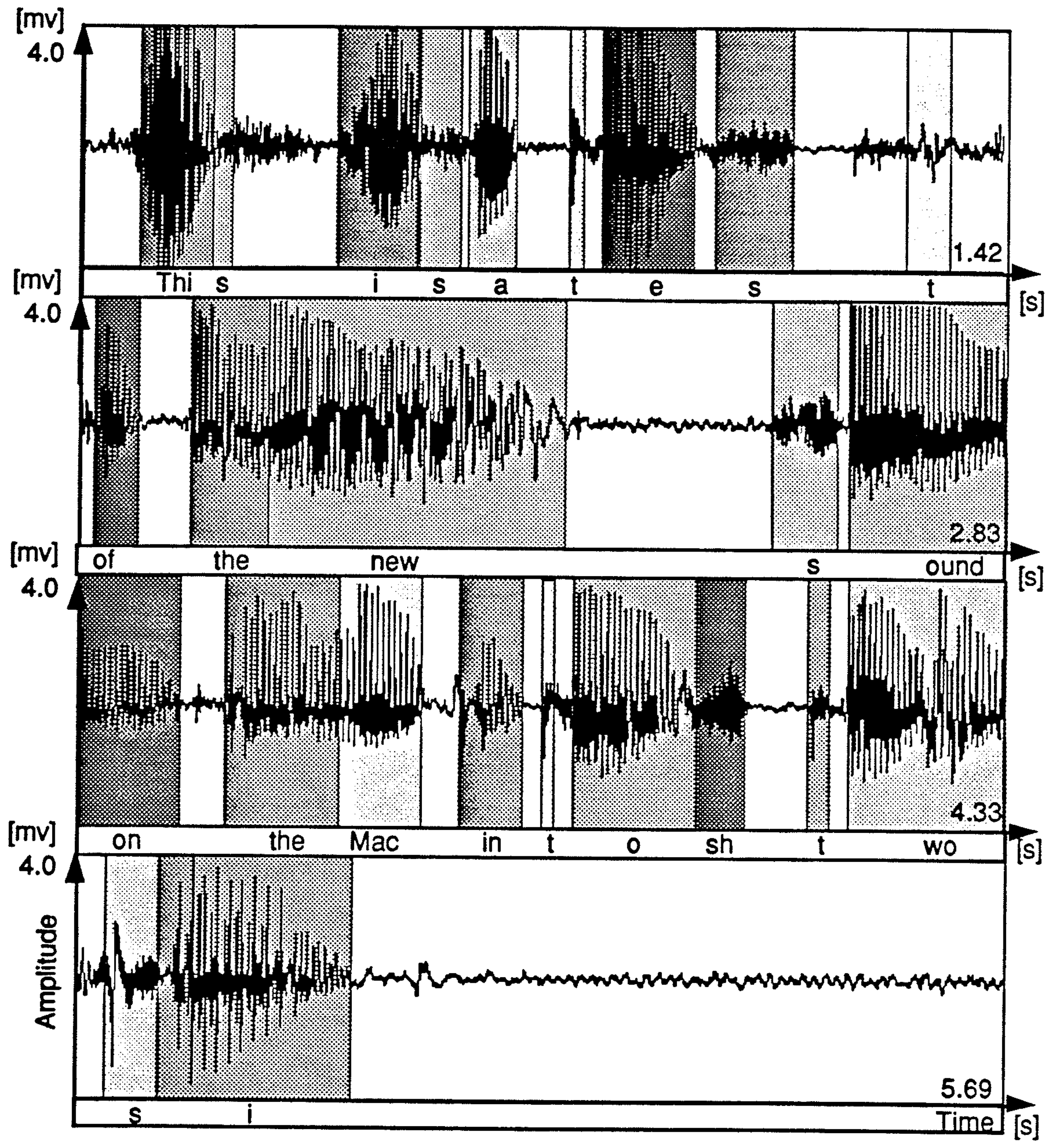


Fig. 2.3 Speech waveform, "This is a test of the new sound on the Macintosh IIsi."

region with an attached label describing the utterance located below the waveform segment. The waveform given in Fig. 2.3 was acquired through a Macintosh electret microphone and processed using MacRecorder's SoundEdit 2.0. In the American English language, 47 phonemes have been identified [Pars86]. These phonemes are shown in Table 2.1.

Table 2.1 Phonetic Alphabet (from [Pars86]).

IPA symbol	Arpabet		Examples	IPA symbol	Arpabet		Examples
i	i	IY	heed	v	v	V	verve
ɪ	I	IH	hid	θ	T	TH	thick
e	e	EY	hayed	ð	D	DH	those
ɛ	E	EH	head	s	s	S	cease
æ	@	AE	had	z	z	Z	pizzaz
ɑ	a	AA	hod	ʃ	S	SH	mesh
ɔ	c	AO	hawed	ʒ	Z	ZH	measure
o	o	OW	hoed	h	h	HH	heat
ʊ	U	UH	hood	m	m	M	mom
u	u	UW	who'd	n	n	N	noon
ɝ	R	ER	heard	ŋ	G	NX	ringing
ə	x	AX	*a*head	l	l	L	lulu
ʌ	A	AH	bud	l̩	L	EL	batt*le*†
aɪ	Y	AY	hide	m̩	M	EM	bott*om*†
aʊ	W	AW	how'd	n̩	N	EN	butt*on*†
ɔɪ	O	OY	boy	ɾ	F	DX	ba*tt*er‡
ɨ	X	IX	ros*e*s	ʔ	Q	Q	§
p	p	P	pop	w	w	W	wow
b	b	B	bob	j	y	Y	yoyo
t	t	T	tug	r	r	R	roar
d	d	D	dug	ʧ	C	CH	church
k	k	K	kick	ʤ	J	JH	judge
g	g	G	gig	ʍ	H	WH	where
f	f	F	fife				

Some sounds from other European languages (IPA notation): Other marks (IPA):

y	F rue, G Bühne	x	G ich, S México	: indicates preceding vowel is long
ʏ	G Hütte	ɛ̃	F vin	ˈ precedes an accented syllable
ø	F peu, G Söhne	ɑ̃	F vent	† Vocalic l, m, n ‡ Flapped t § Glottal stop
œ	F boeuf, G Götter	ɔ̃	F vont	
ɯ	R *ы*	œ̃	F un	

Another important speech characteristic is frequency and bandwidth. A phoneme is sufficiently characterized by the first three fundamental frequencies, or formants. For vowels, the formant frequencies are listed in Table 2.2. Also, vowels and consonants differ fundamentally in terms of their frequency spectrum. Vowels oscillate at quasi-periodic rates while consonants are contained in a broad band of frequencies. A *sonogram* or spectrograph of the sentence, "This is a test of the new sound on the Macintosh IIsi" is given in Fig. 2.4. The sonogram provides a useful graphical description of the spectral features of speech. The time-varying spectral characteristics are graphically displayed in the sound spectrograph. The vertical axis represents the frequency, and the horizontal axis, time. The darkness, or density of dots, is proportional to the signal energy. Dark banded regions show up as resonant frequencies, or formants, of the vocal tract while voiced

Table 2.2. Formant Frequencies of Ten Vowels (from [RaSc78]).

FORMANT FREQUENCIES FOR THE VOWELS					
Typewritten Symbol for Vowel	IPA Symbol	Typical Word	F_1	F_2	F_3
IY	i	(beet)	270	2290	3010
I	ɪ	(bit)	390	1990	2550
E	ɛ	(bet)	530	1840	2480
AE	æ	(bat)	660	1720	2410
UH	ʌ	(but)	520	1190	2390
A	ɑ	(hot)	730	1090	2440
OW	ɔ	(bought)	570	840	2410
U	ʊ	(foot)	440	1020	2240
OO	u	(boot)	300	870	2240
ER	ɜ	(bird)	490	1350	1690

regions are characterized by a striated appearance on the spectrograph. Unvoiced regions appear more solid [RaSc78]. The frequency content of the sentence given in Fig. 2.4 is contained in the toll quality band consisting of frequencies between 300 Hz to 3300 Hz.

The next important property of speech is the degree of stationarity. Speech changes significantly in amplitude and frequency over long periods (greater than 30 ms) of time, and is thus non-stationary, while over short intervals (20 to 30 ms) speech is quasi-periodic, or locally stationary [RaSc78]. An example showing these properties is given in Fig. 2.3. In this figure, the vowels have a very periodic nature about them, but for complete words though, this periodicity is lost. The local stationarity is attributed to the earlier result which dealt with the construction of the language and the fixed rate of change of the human vocal apparatus.

The next step is to apply these properties and speech characteristics to analyze the APW. This next section deals with speech processing techniques. The two fundamental classes in speech processing are waveform and parametric. From this description, a method of analysis is selected. Following the analysis method descriptions and technique selection, a discussion of speech features is given.

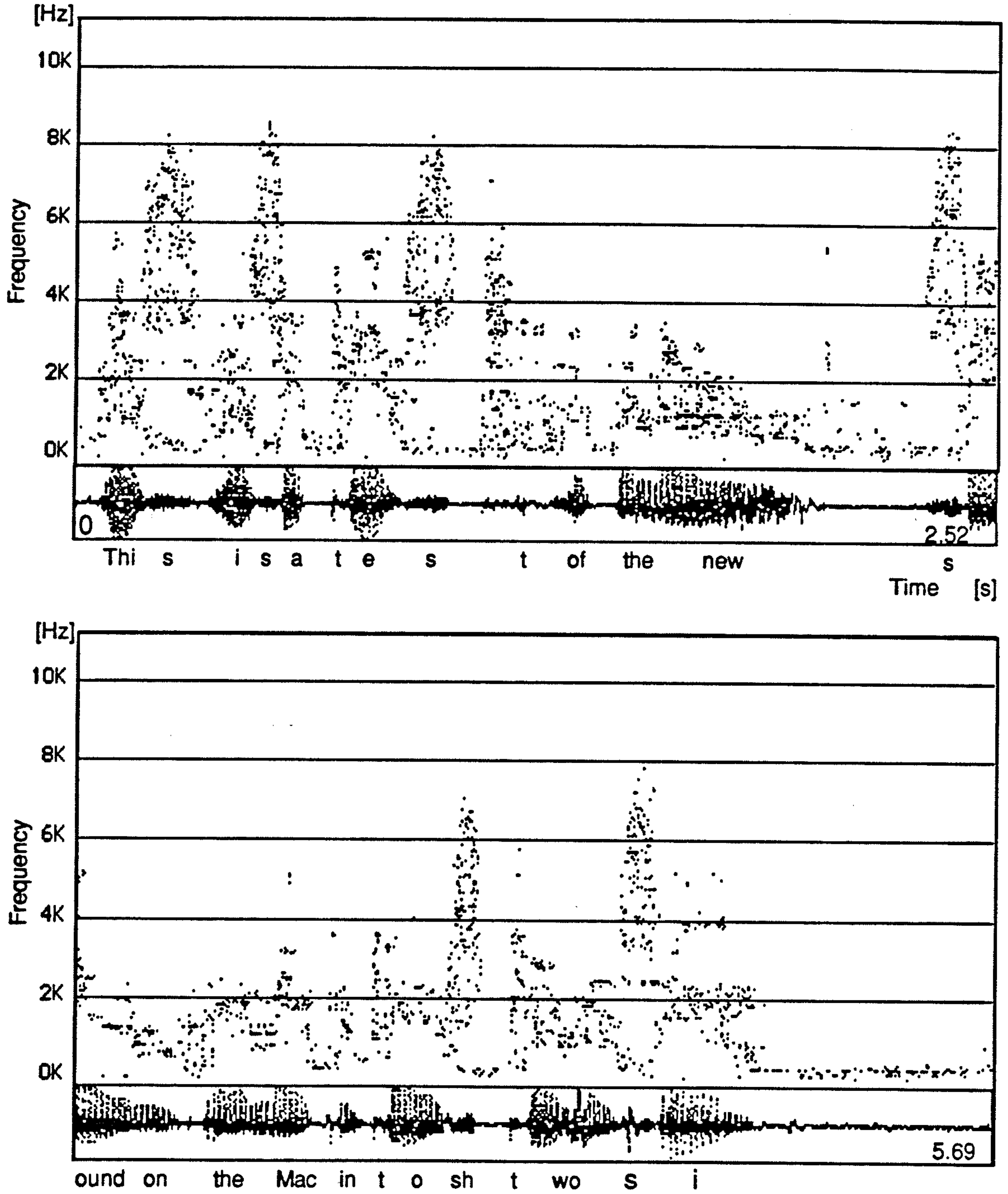


Fig. 2.4 Frequency Spectrum of the sentence, "This is a test of the new sound on the Macintosh IIsi."

2.3 Speech Processing Techniques

Before recognition of isolated speech by a machine can be accomplished, the speech waveform must first be pre-processed. This involves filtering the analog waveform in the toll quality band (300 Hz to 3300 Hz), sampling the speech waveform, and encoding the samples using some analytical method. In speech processing, two fundamental coding classes have evolved. The first class is known as *waveform coding* while the second class is known as *parametric coding*. Figure 2.5 provides an overview of the two classes with respect to quality, bit rate, and coding technique.

Waveform techniques track and record the actual amplitude fluctuations of the speech waveform using a finite digital scale (usually using 8 bits or 255 levels) while sampling the waveform at a fixed frequency (for toll quality, this would be 8 kHz). Waveform methods require more storage since in order to track the speech waveform, it turns out that a sampling rate of 64 kbits/s (8 bits multiplied by 8 kHz) is required to accurately capture speech in the toll quality band. Waveform techniques require more storage but they also provide superior sound quality when compared to parametric techniques. This increase in quality is attributed to the waveform analysis approach as opposed to the parametric approach, which models the observed waveform through filters which simulate the vocal tract. Some examples of waveform coding techniques include: (i) pulse code modulation (PCM), (ii) differential pulse code modulation (DPCM), and (iii) adaptive differential pulse code modulation (ADPCM) [FeLo89]. PCM does not employ any speech compression technique; DPCM stores only the signed difference of consecutive samples, while ADPCM adaptively changes its quantization step size to improve the accuracy of obtaining the signed difference of consecutive samples. DPCM and ADPCM only require half the storage of PCM. DPCMs two limitations which degrade waveform tracking performance are slope overload and quantization error (or granular noise) [FeLo89].

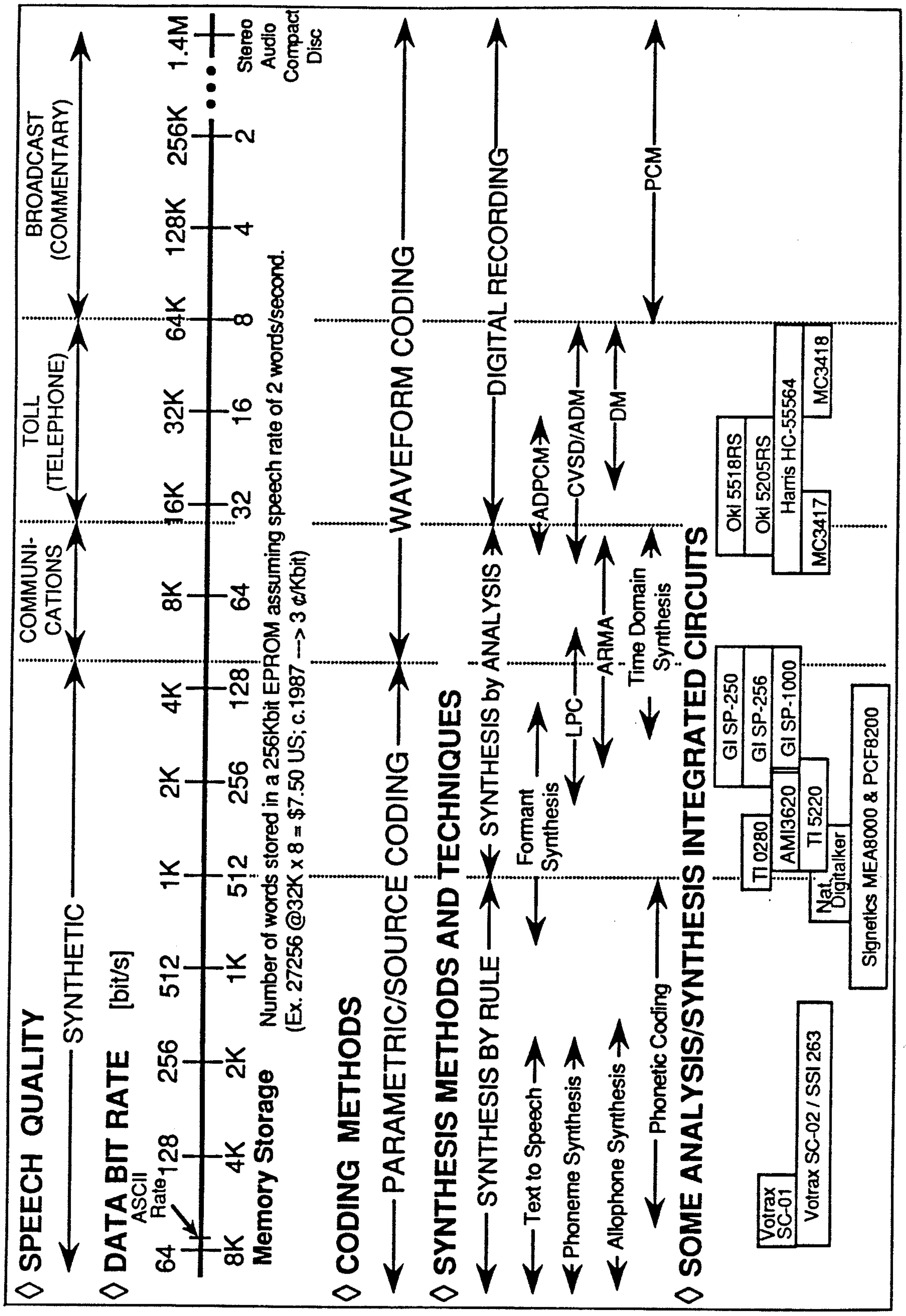


Fig. 2.5 Speech encoding techniques (after [KlKi87]).

By using an adaptive quantizer, ADPCM effectively minimizes both limitations. Figure 2.6 (a-c) show an example of the resulting waveforms using these three models. The quality of ADPCM is superior to DPCM and has a SNR of 10 dB to 12 dB higher than PCM for the same bit rate [FeLo89].

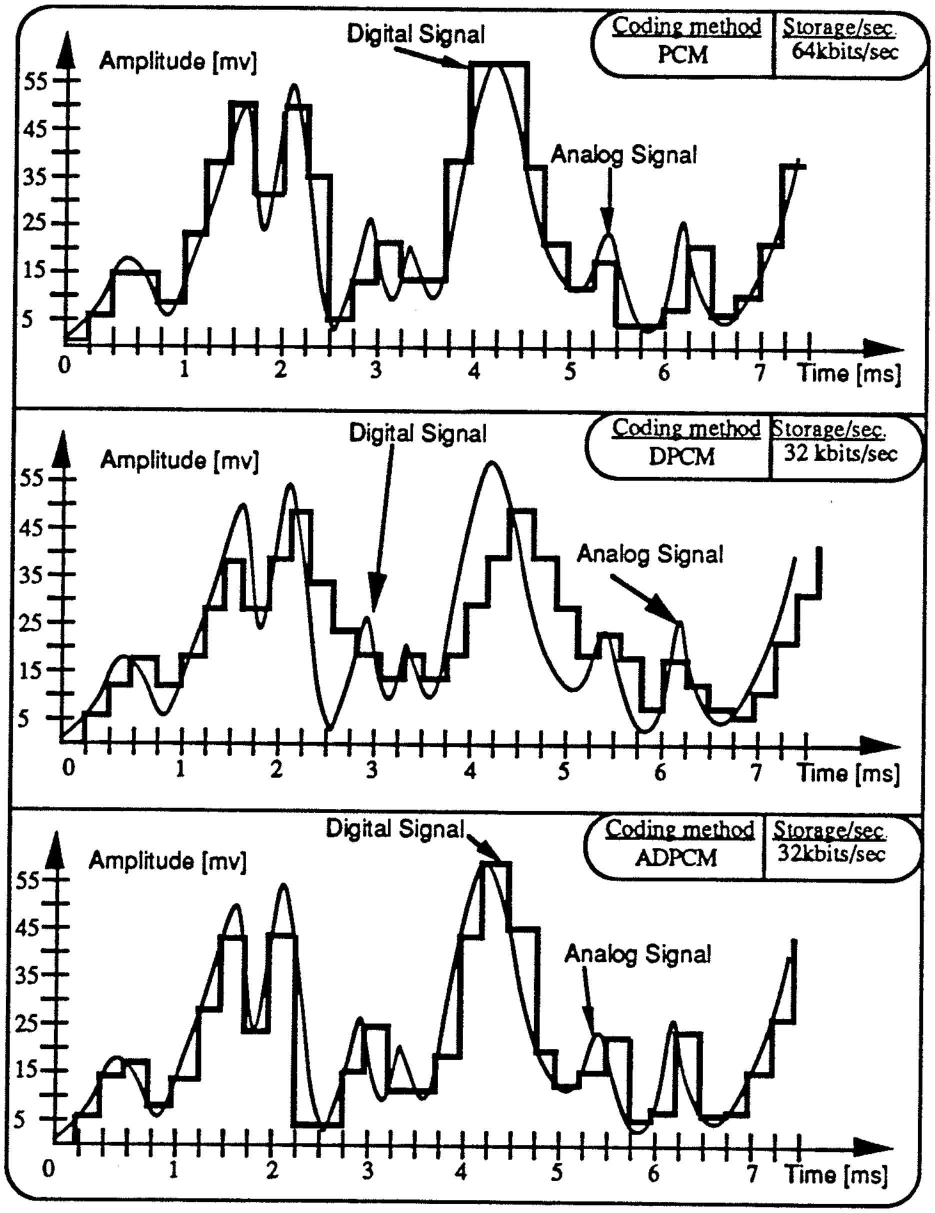


Fig. 2.6 Analog and associated digital representation of: (a) PCM, (b) DPCM and, (c) ADPCM.

The second form of encoding does not track and record the actual fluctuations of the analog signal, but succeeds by obtaining spectrally related parameters, amongst another few vocal parameters. Parametric coding techniques, such as LPC, "utilize an electronic model of the human vocal tract which is driven by signals from frequency and noise generators to *mimic* the natural resonances of the vocal tract" [Klim87]. LPC parameters include filter coefficients, amplitude, pitch, and voiced/unvoiced parameters to model short stationary periods (20 to 30 ms) of speech. Parametric coding techniques, like LPC, are inferior to waveform coding, but they possess the advantage of speech storage reduction (refer to Fig. 2.5). In speech recognition systems, interest is in achieving correct recognition - not in maintaining high speech quality. As a result, waveform techniques are not employed. LPC is used because the speech data is contained in a very compact form which contains the speech essence. A diagram of an analysis method used to obtain all of the LPC parameters is shown in Fig. 2.7. The six pitch period estimates (PPEx) generated from the Gold-Rabiner pitch detection unit (dashed box) are used to quickly select the correct pitch based on majority logic. Upon peak detection using the signal peak processor, three peak estimates, M1 to M3, and three valleys, M4 to M6, are generated. These three symmetric estimates are based on amplitude. Certain threshold logic is also included to improve upon the voiced/unvoiced parameter decision. The gain is determined in such a manner as to, "match the short-time energy of the excitation source to that of the residual error sequence" [Swan87]. After pre-processing, the partial correlation (PARCOR) coefficients are obtained from the LPC filter. PARCOR coefficients are good since they maintain filter stability when quantized, and through spectral sensitivity have been observed as having high sensitivity for magnitudes close to one. For a detailed description of LPC, refer to [Swan87]. Although there is a definite trade off between the quality of LPC as compared to waveform techniques, the advantage remains in the reduced storage and spectral representation.

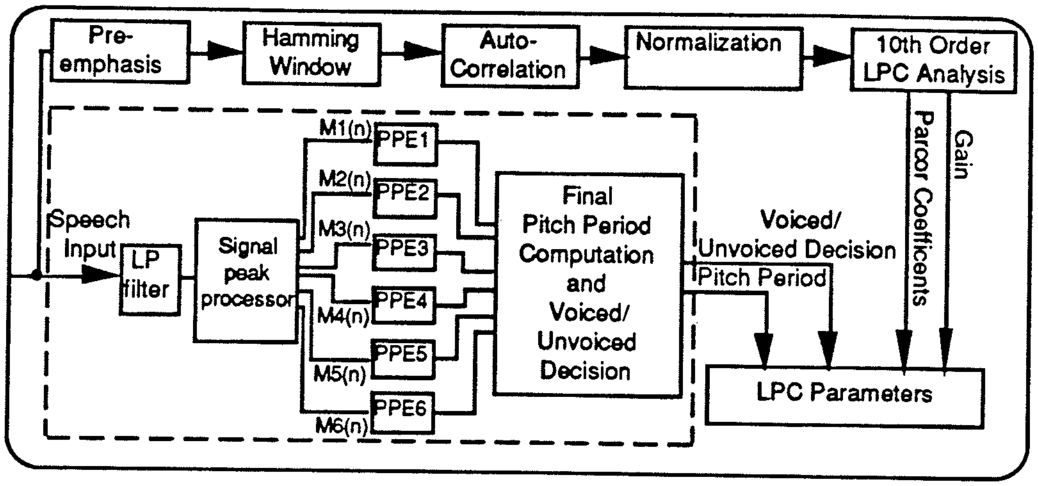


Fig. 2.7 Parametric coding analysis for LPC (after [Swan87]).

Real-time recognition requires spectral information and LPC provides this through the LPC coefficients. The manner of obtaining LPC data and its role in recognition will become evident in the following section. The next step in understanding speech processing and recognition is to learn what constitutes a good speech recognition feature and, how that feature may be employed by a recognizer to achieve correct recognition. This analysis is based on the physiological characteristics of speech production and analytical techniques.

"The design of recognition systems is centered around two problems: feature selection and evaluation and, the decision rule. Recognition system operation consists of two phases: training (learning) and recognition" [Pars86]. If this is what is required for recognition, then how should the system be designed to perform both of these functions, and what steps are involved in arriving at the solution? In isolated speech recognition an isolated utterance is the recognition objective. The first step towards the solution is to evaluate features that will be good for isolated speech recognition. Following this selection, a recognizer that is good with both isolated recognition and the chosen feature is investigated. Instead of a binary grading scheme, a multi-level scheme is used which provides a degree of membership from the unknown utterance to the trained utterances. Commercial recognition systems can not be used in this thesis because they perform binary recognition (match / no-match), their error distance measure is fixed and proprietary, their architecture is generally inflexible, and the human interface does not meet the requirements of this thesis. The two feature extraction techniques evaluated are the FFT and LPC. The two types of recognizer methods discussed are template matching and neural networks.

2.4 Feature Technique Selection

The careful evaluation and selection of features for isolated speech recognition will increase correct classifications. "A feature is some measurable characteristic of the input which has been found to be useful for recognition. Feature extraction is the process of jettisoning as much irrelevant information as possible and representing the relevant data in a compact and meaningful form" [Pars86]. It is understood that successful speech recognition must contain some spectral characteristics of the speech waveform, "As should be obvious, the first principle of speech recognition, spectral estimation, is essential. At this time, Fourier and linear predictive techniques are most common" [LeRo90]. Hence, the two techniques focussed on to extract spectral information are LPC and the FFT.

2.4.1 The Fast Fourier Transform

The FFT is a subclass of a larger family of transforms known as discrete Fourier transforms (DFT). The FFT has become popular in signal processing starting in 1965 due to computational efficiency and its ability to provide the spectrum of a waveform cheaply. Since that time, the FFT has become extremely popular and many applications in signal processing have evolved from its use. The resulting FFT coefficients may be used as the spectral feature in obtaining the speech templates used in the recognition procedure. A discussion concerning this goal is now presented.

The FFT extracts the frequency spectrum of a finite length waveform. Performing the *forward transform* of the discrete amplitude samples from the speech waveform yield the frequency components. The method used to calculate the frequency components is shown in Eq. 2.1, where *X(k)* are the frequency components, *x[n]* are the discrete time samples, *N* is the number of samples used in the transform, and the exponential term represents the mapping rule from the time domain to the frequency domain. In Fig. 2.8 (a), the discrete

$$X(k) = \sum_{n=0}^{N-1} x[n]\, e^{\frac{-j2\pi nk}{N}} \qquad (2.1)$$

samples of a trapezoid waveform are shown. Using the forward transform given in Eq. 2.1 over a discrete interval, the spectrum of the trapezoidal waveform is determined as shown in Fig. 2.8 (b).

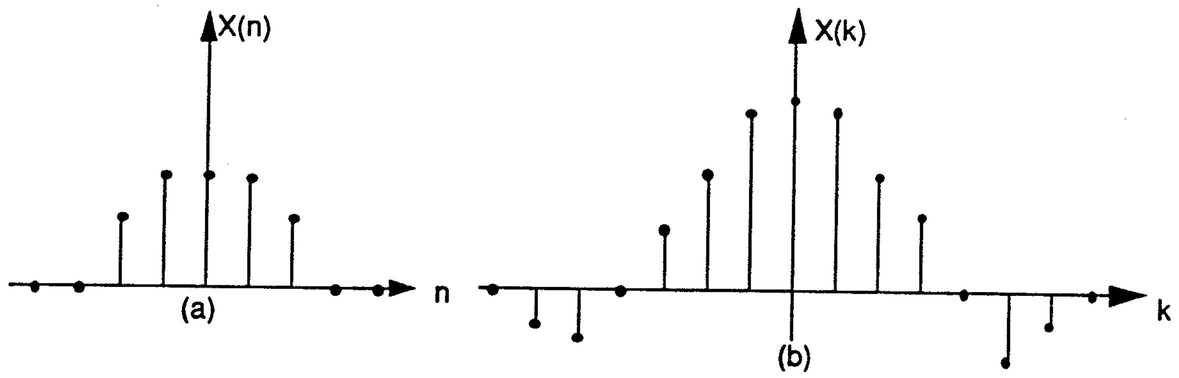


Fig. 2.8 Representation of a trapezoidal waveform in the (a) Time domain and (b) Frequency domain (after [Pars86]).

The spectrum of a speech waveform is analyzed similarly. First, the analog waveform is filtered and sampled. The samples are normalized, separated into windows of a discrete length (256, 512, or 1024 points), and passed through a FFT which determine the spectral coefficients of the waveform segment. The set of spectral coefficients obtained from each speech waveform window are the recognition feature. One FFT algorithm that is commonly used is based on the radix 2 butterfly. An example providing the derivation and feature extraction implementation of the radix 2 algorithm follow. The butterfly, shown in Fig. 2.9, is the basic structure used in the radix 2 algorithm. The two frequency components, X_0, and X_1, are calculated using Eqs. 2.2 and 2.3. By using *decimation in*

$$X_0 = x_1 W^0 + x_0 \tag{2.2}$$

$$X_1 = x_0 - x_1 W^0 \tag{2.3}$$

time, which amounts to interleaving the time domain elements in the algorithm, the number of steps in the decomposition is reduced from $\log_2(N\text{-}1)$ to $\log_2(N/2)$, where $N = 2^a$ $(a \varepsilon I)$ is the number of samples in the transform and is greater than the duration of the signal.

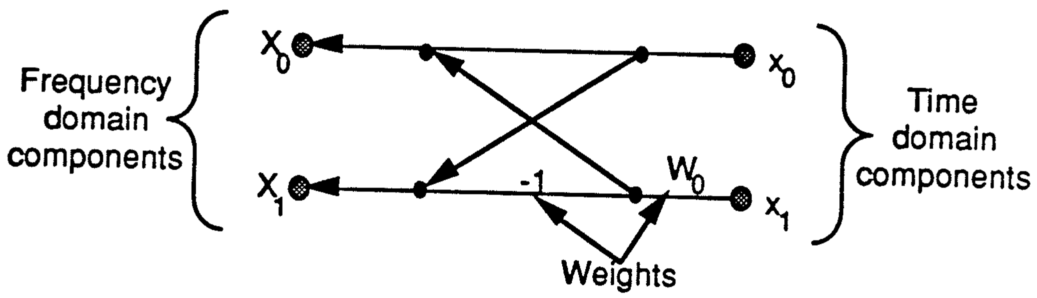


Fig. 2.9 The radix 2 butterfly.

The total number of calculations involves complex multiplications plus additions/ subtractions. The number of complex multiplications, *Mc*, is determined using Eq. 2.4 where *N/2* represents the number of multiplications per stage and log_2 *(N/2)* represents the number of stages. The number of additions/subtractions, A_C, is determined using Eq. 2.5 where *N* represents the number of additions/subtractions, log_2 *(N/2)* represents

$$M_c = \frac{N}{2} \log_2 \frac{N}{2} \tag{2.4}$$

$$A_c = N(\log_2 \frac{N}{2} + \log_2 N) = N \log_2 N \tag{2.5}$$

the number of stages, and $log_2\ N$ represents the last stage (order 2 stage). The computational savings appears in the reduced number of complex multiplications. The entire flow diagram of a two stage FFT (for N = 4, N/2 stages) is shown in Fig. 2.10. The time decimated samples are shown on the right with the ordered frequency components on the left. The conventional abbreviation for the forward transform mapping rule is *W*. The symbol *W* represents the mapping from the time domain to the frequency domain and is given by Eq. 2.6. The FFT procedure begins by generating the weight matrix representing the system given in Fig. 2.10 as described by Eq. 2.7.

$$W = e^{\frac{-j 2 \Pi n}{N}} \tag{2.6}$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.7}$$

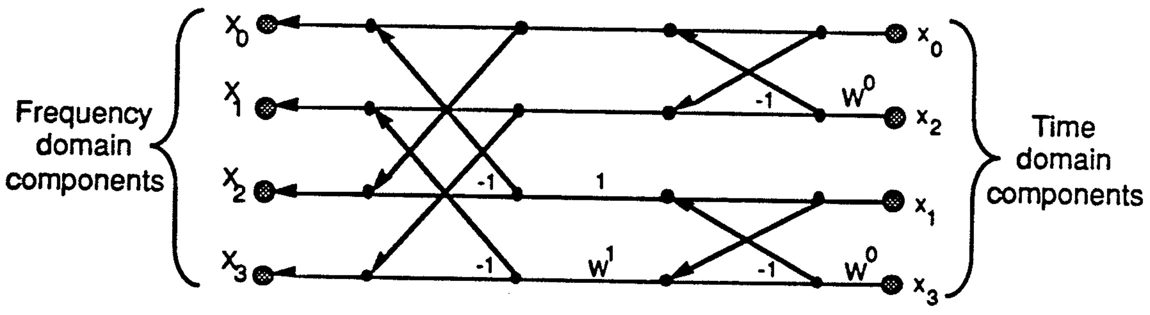


Fig. 2.10 Flow diagram describing the radix 2 FFT for N = 4.

Next, the matrix is interleaved into two smaller matrices consisting of even and odd indexed powers. Using standard matrix multiplication, the appropriate weight elements are extracted, simplified, and assembled into the two smaller matrices given by Eq. 2.8, even powers, and Eq. 2.9, odd powers. This results in similarly ordered frequency and time

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & W^2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ W^1 & W^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \quad (2.8)$$

$$\begin{bmatrix} X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & W^4 \\ 1 & W^6 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \end{bmatrix} - \begin{bmatrix} W^2 & W^6 \\ W^3 & W^9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \quad (2.9)$$

components. Since computers function best on discrete data, the FFT is very suitable for computer speech recognition. Also, the spectrum of the FFT is a very useful feature for speech recognition. "The FFT, a transform of the signal, reflects its clustering properties better than a parametric code" [Koho88]. Clustering occurs when features are able to separate classes well enough to define non-overlapping regions which permits the recognizer to *cluster* or group the data into categories *without supervision.* "Clustering has not been seen until recently since recognizers were not able to achieve recognition high enough to make this feasible" [Pars86]. The FFTs disadvantages include: (i) lengthy computation time, possibly too long for *real-time* speech recognition unless a very fast DSP is used (such as the TMS320C30), (ii) system complexity for sample processing and frequency component generation, and (iii) the data representation may not be compact enough for real-time speech recognition by a neural network. An alternative to this feature is LPC.

2.4.2 Linear Predictive Coding

A discussion of LPC has been presented as a speech analysis technique in Sec. 2.2.2. An extension of that discussion is now provided. Concern here is focussed on how LPC works and why it is a good spectral feature.

The objective of LPC is to determine a set of predictor parameters directly from the speech signal in such a way as to minimize the difference between the actual speech and the predicted speech signal given a linear discrete-time system. These parameters include pitch interval settings, voicing, predictor coefficients, and amplitude. Every 20 to 30 ms a set of these parameters is obtained. The predictor coefficients are the feature used as input to the BP neural network for both training and testing, "the goodness of the model means that the linear prediction is an appropriate way to encode speech and that the predictor parameters are a valuable source of information for recognition" [Pars86].

An estimate, $\tilde{y}(n)$, of the predicted output is estimated by taking a linear combination of the past outputs, *y(n)*, and past inputs, *x(n)*, given by Eq. 2.10. The terms *a(•)* and *b(•)* represent the predictor coefficients. There are many systems that can at least be approximated by the difference equation given by Eq. 2.11. Thus, if the predictor

$$\tilde{y}(n) = \sum_{j=0}^{q} b(j)\, x(n-j) - \sum_{i=1}^{p} a(i)\, y(n-i) \tag{2.10}$$

$$\sum_{i=0}^{p} a(i)\, y(n-i) = \sum_{j=0}^{q} b(j)\, x(n-j) \tag{2.11}$$

provides a good representation of the system, the system function can be obtained directly from the predictor coefficients based on the Z–transform. The system transfer function, *H(z)*, is defined by Eq. 2.12, and after performing the Z–transform becomes Eq. 2.13. Hence, the predictor coefficients provide the locations of all the poles and zeros of the transfer function. The general equation given by Eq. 2.10 is called the *mixed pole model*

$$H(z) \triangleq Y(z)/X(z) \tag{2.12}$$

$$H(z) = \frac{\sum_{j=0}^{q} b(j)\, z^{-j}}{\sum_{i=0}^{p} a(i)\, z^{-i}} \tag{2.13}$$

and is associated with the statistical model called the *Autoregressive moving average* (ARMA). There are two important variations of this model however, the *all-pole model* and the *all-zero model.* The all-pole model (auto regressive (AR) model) enforces that the numerator *N(z)* remain constant. If this is the case, the past input predictor coefficients, *b(j)*, are set to zero for *j* greater than zero. Thus, the predictor is based entirely on the past output predictor coefficients *a(i)*. In a similar fashion, the all zero model (the moving average (MA) model), has denominator, *D(z)*, set to unity. This removes the effect of the past predictor outputs, *a(i)*, from the predictor equation and leaves only the past predictor inputs, *b(j)*, as a means to calculate the predictor coefficients.

The all-pole model is commonly used because: (i) it is easier and faster to compute, (ii) future values may not be known, (iii) the vocal tract is well represented by the all pole model, and (iv) zeros can be approximated (expanded) by poles, and if they converge quickly, only the first few terms are necessary [Pars86]. Finally, in speech applications, the zeros of the ARMA difference equation represent nasals and some fricatives which can be ignored for the most part with little loss of accuracy [Swan87]. In return, a simplified equation that represents a good model of the vocal tract is achieved.

Derivation of the simplified linear prediction model is now provided. Beginning with a zero-mean signal, and using the all-pole model of the ARMA difference equation, an estimate, $\tilde{y}(n)$, is calculated using Eq. 2.14. The associated error signal is given by Eq. 2.15, where *a(0)* equals one. In order to obtain the predictor, $\tilde{y}(n)$, the coefficients, *a(•)*, are determined in a manner which minimizes the error in a mean squared sense. This is done most efficiently by taking advantage of the orthogonality principle which states that

$$\tilde{y}(n) = -\sum_{i=1}^{p} a(i)\, y(n-i) \tag{2.14}$$

$$e(n) = y(n) - \tilde{y}(n) = \sum_{i=0}^{p} a(i)\, y(n-i) \tag{2.15}$$

if *i* is orthogonal to *j*, the vector product is zero. In the predictor case, this leads to the requirement given by Eq. 2.16.

$$< y(n-j)\, e(n) > = 0, \text{ for } j = 1,2,\ldots p \tag{2.16}$$

Substituting *e(n)* back into Eq. 2.16 and simplifying yields the following result given by Eq. 2.17. Finally, by interchanging the operations of averaging and summing, and representing <•> summed over *n*, provides the final form given by Eq. 2.18.

$$< y(n-j) \sum_{i=0}^{p} a(i)\, y(n-i) > = 0 \tag{2.17}$$

$$\sum_{i=0}^{p} a(i) \sum_{n} y(n-i)\, y(n-j) = 0 \text{ , for } j = 1,\ldots,p \tag{2.18}$$

The predictor coefficients, *a(i)*, are found be solving this set of equations using the Levinson - Durbin iterative technique of solving a system of linear equations. A frame of speech of finite length, say 25 ms, is input to the LPC system and the resulting floating point predictor coefficients are then used as the spectral feature in speech recognition.

2.5 Recognition Method Selection

The second phase of operation in a recognition system following feature extraction and training is recognition. The recognition methods under consideration are template matching and neural networks. For template matching, recognition involves collecting a series of templates and comparing an unknown template to each known template. For neural networks, this involves finding a mapping from the templates to the desired output vector using a single set of weights. It is the objective of this section to determine a sufficient recognizer. Four metrics are available to measure the difference in distance between the unknown template and the target template, whether it is to minimize the cost function or mean squared error. The distance metrics include: (i) Hamming, (ii) Euclidean, (iii) Minkowski, and (iv) Chebychev. Also, the Itakura-Saito distance measure may be considered and is popular when using LPC as the spectral feature in recognition [Pars86].

2.5.1 Template Matching

Better than linear compression-expansion is the dynamic time warping (DTW) technique which can achieve optimum non-linear time alignment [LeRo90]. This technique involves a template containing the waveform feature which is time normalized and warped onto a known set of utterance templates prepared in a similar fashion. A template consists of a sequence of vectors arranged in a linear fashion with each vector consisting of a feature of the waveform. The matching is performed by using a metric which compares the unknown template to each of the known templates. The template that is closest in a metric sense to the unknown template is considered the correct match. Although DTW is a good technique when applied to recognition, it requires the accurate location of the utterance end points for improved isolated utterance recognition. When considering isolated speech recognition, the template contains the entire utterance. Locating the end points assists in reducing the cost function result by providing an improved initial time-alignment between the unknown template and the known template. This initial time alignment will improve the search procedure by placing the unknown template closer in time to that of the known template. This will result in a more accurate time normalized distance. If the end points are not located, the template may be misclassified. "This means that time-dependent features may fail to match because the unknown and the reference word are out of time registration. In such a case, the correct pattern may seem as far different from the unknown as any incorrect pattern" [Pars86]. A resulting search path and window between the word *pattern* is shown in Fig. 2.11. The smallest distance, D(G,H), results from the template that has the fewest timing variations to the unknown template. Three possibilities exist, (i-1,j), (i-1,j-1), and (i,j-1) when determining the minimum energy extending from an arbitrary point (i,j) as shown in Fig. 2.11.

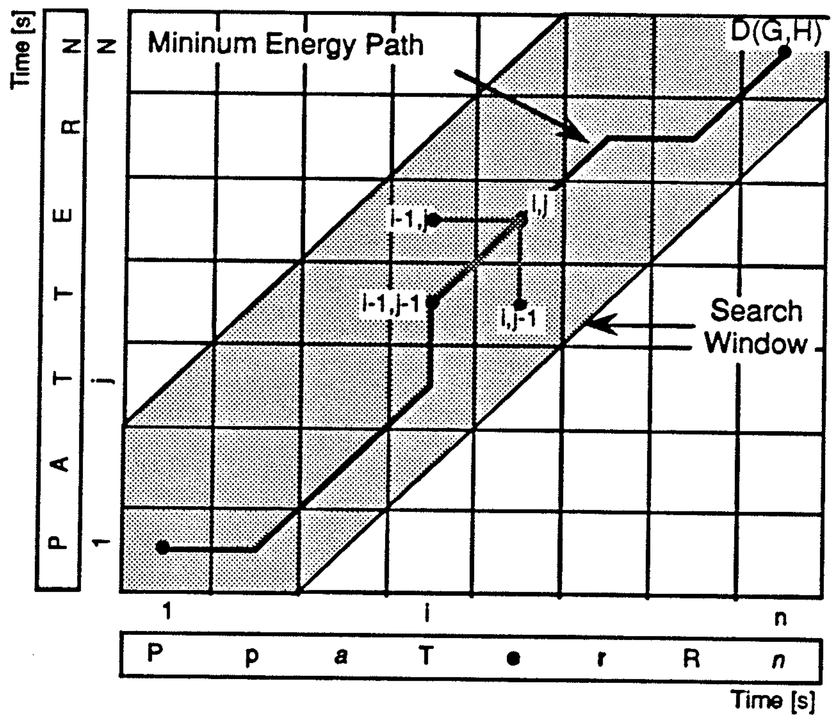


Fig. 2.11 DTW template matching (after [Holm88]).

Since an exact match is unlikely, the smallest distance is chosen. A match can only be made once all library templates have been compared. Therefore, a constant time is required for an unknown template to be identified which is proportional to the library size. DTW is not used because the classification time is dependent on the library size, multi-level classification based on a pattern is not possible using the conventional approach and, for highly accurate recognition, the location of the utterance boundaries are a necessary component. Endpoint location involves additional processing, adding to the classification time. Further to this, "In many cases the accuracy of time alignment depends on the accuracy of identifying the end points" [Pars86]. Finally, DTW may even be used by neural networks. In this approach, following time alignment, the resulting template would be used as input into the neural network. This would assist in reducing timing variations of the input pattern which may lead to higher network classification results. This approach would increase the classification time, thus it will not be attempted. A complete description of DTW is contained in [SaCh78], [Pars86], and [Pete88].

2.5.2 Neural Networks

The artificial neural network approach, which is also referred to as *parallel distributed processing*, provides an entirely different approach to managing the templates in the recognition process. The goal of a neural network is to determine a mapping of the input templates to their respective output templates using a single weight matrix. The mapping has a constant recognition time which may be small enough to permit real-time recognition. A neural network contains many units called neurons which perform a single function. This function is to sum weighted inputs and perform a non-linear mapping of the sum to obtain an output value which is propagated to connected neighboring neurons. The network architecture, activation function, and learning paradigm vary between models. The purpose of this section is to determine a sufficient model to use for isolated speech recognition.

In a neural network, there are usually two steps involved in recognition. The first step involves (supervised) *training* which is used to adjust the network weights to obtain the correct mapping, and this is achieved through a variety of techniques. The last step is generalization. Generalization allows a network to be able to correctly classify patterns that it has not seen before [Rein90]. Neural nets are robust with noisy data which typically accompany speech signals and thus, quickly provide a good solution to the abstract problem of isolated speech recognition.

Recent interest has shifted to *recurrent* neural networks. "Since speech is a complex time-sequential process, recurrent networks have been viewed as a natural choice for modelling the temporal structure of speech in both production and perception" [Watr90]. Others have also developed systems that use the recurrent network approach with limited success on isolated speech, [Watr90], and [HaWa90]. In one approach, the network architecture is unchanged from the typical concurrent approach except for the recurrent links found in the hidden layer and/or output layer. A recurrent link is a connection which is established from the output of a neuron back into the input. In this manner, a form of dependency on past inputs is accomplished. This dependency is shown using Fig. 2.12.

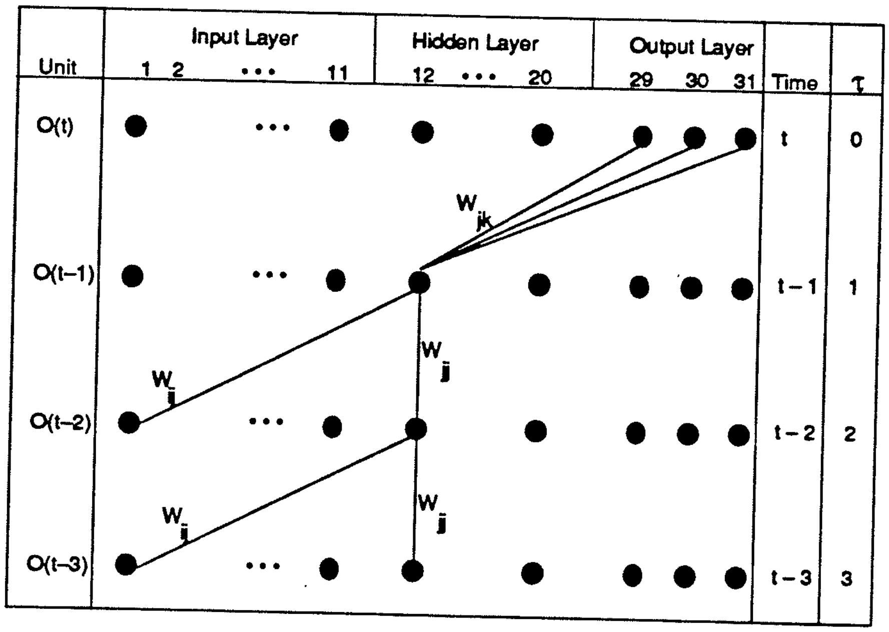


Fig. 2.12 Dependence of the network outputs with recurrent links at earlier time frames. The columns correspond to the 3 layer BP architecture. The rows locate output O(t) of each unit as a function of time (after [Watr90]).

A pattern, using this technique, consists of a group of pattern segments. Thus, in order to present a single utterance *pattern*, a set of time related input pattern segments are presented in an ordered sequence to the network. Results of 96% to 98% recognition were achieved on specific vowels and consonant combinations [Watr90]. The results of concurrent networks achieve near this accuracy and the computational overhead is much less. Therefore, in light of the fact that one of the objectives is to achieve real-time recognition, this approach is avoided.

A limitation of supervised BP is that a priori training data is required. For very complex tasks, a lot of data is required in order to achieve sufficient weight accuracy, which is reflected as correct recognition. Given a network with three layers having N weights, N^2 patterns are generally required for statistically reliable weights. This overhead requires long training periods to correctly adjust the network weights. Also, it is not always convenient or possible to acquire a large enough data set to accurately determine the mapping. Another deterring factor in a software developed BP neural network is the possibility of achieving only a local solution which is not optimum. In the following chapter, these problems are addressed as well as a detailed discussion of the classical BP technique which is used in this thesis.

2.6 Summary

Speech is generated by the vocal apparatus based on abstract neural signals originating in the brain. It is transmitted through air as an APW, and has characteristic features including amplitude, pitch, frequency, and resonant length. Speech is inherently redundant (up to 50% of the time), locally stationary over short periods, non-stationary over long periods and, can be sub-divided into 47 fundamental units called phonemes. Speech encoding is possible using either waveform or parametric analysis of the APW. Waveform methods offer higher quality and bit rates, while parametric encoders generally have lower quality and bit rates. For real-time recognition, an encoder which employs LPC is selected because achievement of real-time recognition necessarily requires that the data contain the essence of speech (frequency information), and that this essence be in the most compact format possible. LPC is chosen over the FFT due to its abilities to provide the essential speech information in a compact format, more so than the FFT, its real-time processing capabilities, and ease of implementation over the FFT. The type of recognizer selected is a concurrent BP neural network. The neural network is used in place of DTW because: (i) it provides multi-level grading and real-time recognition, (ii) it has a constant recognition time which is independent of library size and, (iii) it does not require additional pre-processing. The concurrent BP neural network is chosen over a recurrent one since the recognition results of [Watr90] were not significantly better from those found using concurrent networks with similar architectures and vocabularies [KMRW87], [DeBo90], and [Burr87]. A further description of the essential components is discussed beginning with a detailed description of the concurrent BP neural network.

CHAPTER III
REVIEW OF BACKPROPAGATION

3.1 Introduction

This chapter presents a review of the backpropagation (BP) neural network which performs isolated utterance recognition. Various distance metrics, used by the neural network to determine the total internal network error, are described. A description of the grading scheme is included which explains how the results of the neural network are interpreted to determine the identity and quality of the response utterance. This is the type of feedback that is necessary in the vocal shaping process. The BP neural network used in this thesis employs an adaptive learning rule and batch update technique to reduce the network training time. Other parameter variations and architectural configurations are also available to provide training flexibility.

3.2 Error Backpropagation

BP is a generalization of the least squares procedure and works with multi-layer networks. It is only one of many techniques used to determine the synaptic efficacies and threshold potentials for multi-layer supervised networks. BP is superior to networks which do not contain hidden units since data sets in which class separability is complex will provide greater difficulty in producing the correct mapping for these other network models.

3.2.1 Background

The BP learning procedure is described by Hinton, Rumelhart, and Williams (1986). Variations of this model were also independently discovered by Werbos (1974), LaCun and Parker (1985). Isolated speech recognition using BP is contained in [Burr88] and [DeBo90]. BP uses a supervised gradient descent technique to adjust the network weights.

Since speech is a very dynamic signal, even for the same speaker, and since it is very complex to represent, it would be very difficult to determine just how to *hand-adjust* the internal weights to represent such utterances if an *automatic* gradient descent learning technique is not employed.

Layered networks, including BP, classify patterns by partitioning the multi-dimensional input space into *hyperplanes*. In the three layer BP architecture, the first hyperplane, located between the inputs and hidden neurons performs *AND* operations (convex regions), while the second hyperplane, located between the hidden and output neurons performs *OR* operations of the convex regions. The combination of *AND* and *OR* operations permits BP to isolate regions or perform clustering [Burr88]. Although it does provide the correct mapping rule, BP is not a realistic parallel to what we perceive to be occurring in humans. This is because error signals, which are back propagated in the algorithm to reduce the weight space error, are not seen in nature since neural signals do not travel in the reverse direction down an axon. Also, the architecture of BP does not agree with other physiological constructions. Specifically, brains have feedback and are more fully interconnected than BP networks. Conventional BP networks are only connected between layers, not interconnected within a layer, and do not permit feedback during the forward pass. One of the plaguing problems of BP is the lengthy training periods involved in achieving sufficient accuracy of the weight space. Much research has gone into reducing the time required to achieve sufficient weight accuracy [VMRZ88], and [McRu89]. Typically, this is achieved using some form of adaptive rule to vary the learning rate and/or momentum. Also to speed convergence, adding noise to either the patterns and/or weights is used to avoid local minima or worse yet, entrapment within a local minima [McRu89]. The lengthy training time is due largely to *ravines* and localized solution *plateaus*. A ravine, representing the energy surface of a problem in three dimensional space is given in Fig. 3.1. Point A is given as the network starting point. Using gradient descent, the network will move to point B, which is across the ravine. Ideally, movement down the centre of the ravine to point C is desired. "Under some conditions, a large proportion of the training effort will be spent oscillating from side to side of the ravine, and not in the desired direction along the bottom of the ravine" [ChFa88].

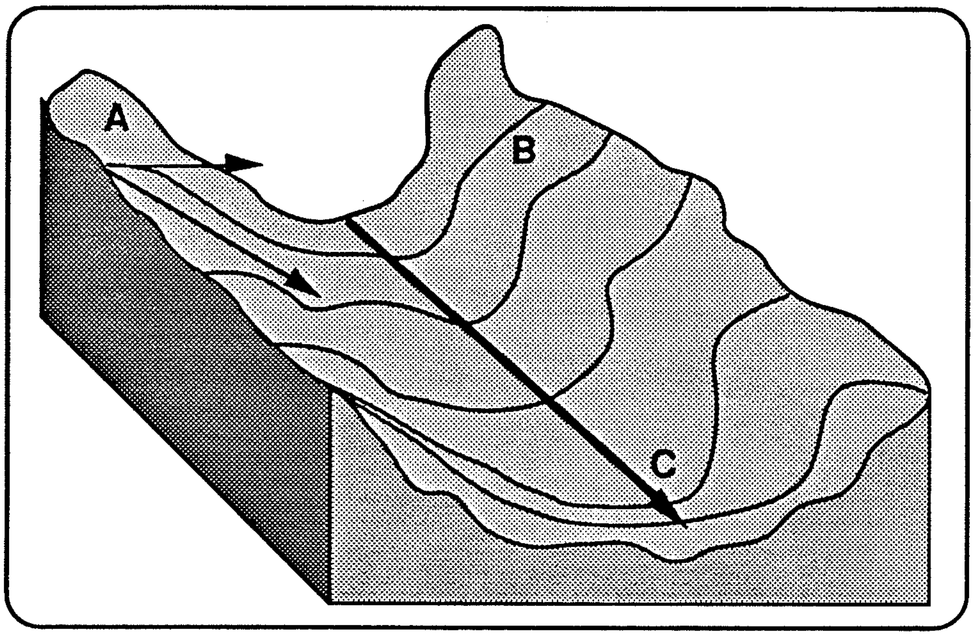


Fig. 3.1 Ravine in three dimensional energy space (after [ChFa88]).

Small learning rates reduce this problem, but also slow movement. Including a momentum factor dampens the oscillations and assists in re-directing the movement towards the ideal point C. "In many tasks in speech recognition, the solution (global minimum) forms a plateau in weight space. As the network nears the plateau, the gradient becomes small, resulting in a small tail off" [ChFa88] as shown in Fig. 3.2. At the sigmoid (logistic activation function) extremes, a neuron is either completely on or completely off. When this occurs, the gradient is small resulting in small movements on the energy surface. To reduce the network training time, an adaptive learning rule is used.

When the total internal error falls by at least 1% to that of the previous total internal error value, then the learning rate is increased by 1% of its previous value. On the other hand, if an increase in the learning parameter yields an increase in the total internal error by more than 1%, the learning rate is decreased by 40% to 50% of its current value for the

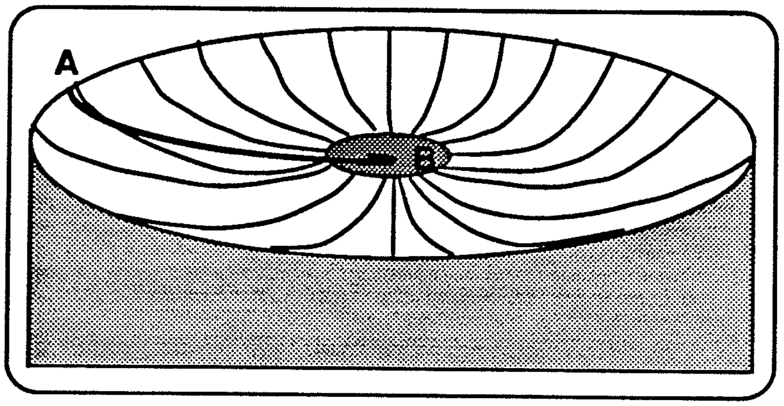


Fig 3.2 Energy surface topology for global solution in 3 dimensions (after [ChFa88]).

forthcoming iteration. The logic behind this rule is that if the network error is consistently being reduced then an increase in the learning can be safely applied to accelerate the training process. If on the other hand, the error begins to oscillate, which may be due to ravine jumping, curvature in the ravine, or narrowing of the ravine [Card90], the sensitivity of learning is decreased quickly in order to correctly exit from this problem zone. The oscillations in error trigger the threshold leading to a drop in the learning rate. This procedure has been applied with success by [LoKi90] and [VMRZ88].

3.2.2 Training

In training, the predefined set of patterns are presented to the input neurons. These neurons propagate their signals to the hidden layer through small random weighted connections, W_{ij}, as shown in Fig. 3.3. Each hidden neuron sums the weighted inputs

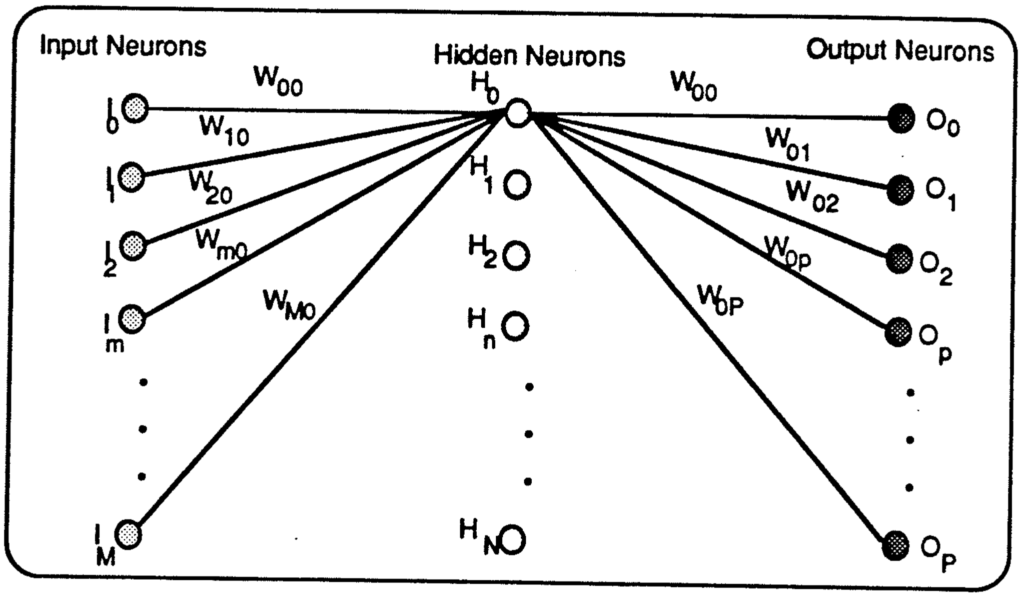


Fig. 3.3 Three layer back propagation architecture.

and may add in a bias weight if such a connection has been made. A bias is an extra neuron which is always on. When calculating the total weighted input to a neuron, the bias weight may be added in before the sum is applied to the logistic activation function. The bias weight shifts the sigmoid characteristic a distance *Th*. The activation function is used in obtaining the hidden neurons output. These descriptions are illustrated in Fig. 3.4. The activation function can be any function that has a continuous first derivative.

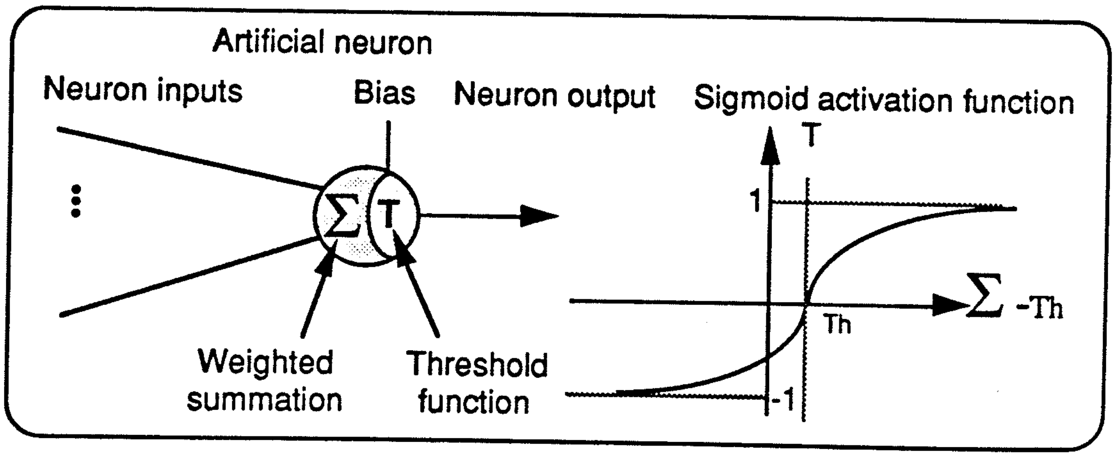


Fig. 3.4 Representation of an artificial neuron and its logistic activation function.

It should be non-linear since a linear activation function gains nothing from using hidden units [McRu89]. The non-linear activation function introduces higher order statistics which provide important feature information of the pattern set to the network and bias weights. This procedure is repeated again between the hidden layer and the output layer. The *output vector* is subtracted from the ideal or *target vector* in obtaining the *error signal.* The negative gradient of the error signal is calculated and backpropagated through the network to modify the weighted connections which reduces the total internal network error. The mathematics of this procedure are given in Eqs. 3.1 to 3.8. Equation 3.1 describes the forward propagation of a pattern which is presented to the input layer and is used to calculate a hidden neurons new activation level, where H_j represents the hidden neuron,

$$H_j = \frac{1}{1 + e^{-(Hb_j + \sum_{m=1}^{x} I_m V(n)_{mj})}} \tag{3.1}$$

Hb_j represents a hidden bias weight, I_m represents the input neuron, and $V(n)_{mj}$ represents a weight located between the input and hidden layer for the current pattern n. Equation 3.2 describes the forward propagation of the hidden neuron vector to the output. This is used to calculate the output vectors new activation levels,

$$O_k = \frac{1}{1 + e^{-(Ob_k + \sum_{m=1}^{y} H_m W(n)_{mk})}} \tag{3.2}$$

where O_k represents the output neuron, Ob_k represent an output bias weight, and $W(n)_{mk}$ represent a weight located between the hidden and output layer for the current pattern n. Equation 3.3 describes the calculation of the pattern error between the target and

$$E_p = \sqrt{\frac{1}{2} \sum_{m=1}^{z} (O_m - Tr_{m,p})^2} \tag{3.3}$$

the actual output vector (this is an application of the Euclidean metric), where E_p represents the error for a single pattern, and $Tr_{m,p}$ represents the corresponding training output value to the output neuron for the current pattern p. Equation 3.4 describes the

$$\delta_k = O_k \ (1 - O_k)\ (Tr_{p,k} - O_k) \tag{3.4}$$

error gradient signal, δ_k, which is backpropagated into the network starting from the output vector. Equations 3.5 to 3.7 describe the adjustment of the weights between the output units and the hidden units. Equation 3.5 describes the weight change, $\Delta W(n)_{jk}$,

$$\Delta W(n)_{jk} = \eta\ \delta_k\ H_j + \alpha\ \Delta W(n-1)_{jk} \tag{3.5}$$

$$W(n)_{jk} = W(n)_{jk} + \Delta W(n)_{jk} \tag{3.6}$$

$$\text{OutputBias}(n) = \text{OutputBias}(n) + \Delta W(n)_{jk} \tag{3.7}$$

which is calculated using the current error signal added to a fraction of the previous weight adjustment, $\Delta W(n-1)_{jk}$, where η is the learning rate. The fraction is set arbitrarily using the momentum factor, α, (α<1). The current weight adjustment is then added to the current related weight, $W(n)_{jk}$, given by Eq. 3.6. A similar procedure is followed for the biases given by Eq. 3.7. Equation 3.8 describes the calculation of the error signal, δ_j^θ, at

$$\delta_j^\theta = H_j\ (1 - H_j)\ (\sum_{m=1}^{k} \delta_m\ W(n)_{jm}) \tag{3.8}$$

the hidden layer. Equations 3.9 to 3.11 describe the propagation of the error signal and adjustment to the input and hidden layer weights. These calculations are similar to those performed for Eqs. 3.5 to 3.7 except that the inputs, I, are used in place of the hidden units. This process is repeated until the total error falls below a set threshold.

$$\Delta V(n)_{ij} = \eta\, \delta_j^{\theta}\, I_i + \alpha\, \Delta V(n-1)_{ij} \tag{3.9}$$

$$V(n)_{ij} = V(n)_{ij} + \Delta V(n)_{ij} \tag{3.10}$$

$$\text{HiddenBias}(n)_j = \text{HiddenBias}(n)_j + \Delta V(n)_{ij} \tag{3.11}$$

The total error, T_E, is defined as the summation of the pattern error of Eq. 3.3 taken over all patterns and is given by Eq. 3.12. The total error can be calculated on a single pattern if desired. The form of presentation of the data set to the network may cause different

$$T_E \triangleq \sum_{m=1}^{P} E_p \tag{3.12}$$

training time solutions. The two forms of pattern set presentations are *sequential* and *random*. Here, random presentation means the pattern set is presented to the network in a randomly permuted order, while sequential presentation involves showing the pattern set in the same order during training. Other settings include initialization using small network weights having random or constant values, biases on selected network layers, an adaptive learning rule, and various metrics to measure the total internal error. The training given in the above set of equations is referred to as *single* update, while the more typical form of training is referred to as *batch* update.

3.2.3 Batch Update

Two variations in updating are possible using BP. The first is single update which repeatedly follows the eight equations provided in Sec. 3.2.2 until the current total internal error value reaches the prescribed stopping criteria. Batch update on the other hand, performs the weight adjustments differently, implying a different application of Eqs. 3.4 to 3.12. In the batch procedure, the weight error derivatives are accumulated over an entire training set, or batch. This implies that the set of weights and biases, both previous values and current are not changed until the end of the pattern presentation set. This is unlike the single update where all weights and biases are changed for each and every pattern. In the equations for calculating the weight update values, the previous iteration weight update values, $\Delta W(n-1)$ and $\Delta V(n-1)$, are required. Ideally, a copy of the entire weight update structure must be stored in memory for every pattern set to provide the correct previous weight changes. This is necessary since the weight updates do not occur until the end of all pattern presentations. Further to this though, once the accumulated weight changes have been acquired, some form of calculation is necessary to determine the *current* set of *old* weight changes, $\Delta W(n-1)$ and $\Delta V(n-1)$, to be used in the forthcoming series of batch weight updates. If the previous weight changes were stored for each pattern, which would tend to be somewhat inefficient and memory intensive, and since the whole idea behind batch update is to decrease the convergence period of the network, a simplified approach is taken. Here, the weight error derivatives are accumulated and the previous batch weight change term is added into the current batch weight change calculation to generate the new set of *previous weight changes* [McRu89]. The *previous weight changes* are applied to every pattern, and they are used in the forthcoming batch update. Hence, each weight that is updated uses the same *previous weight changes*, $\Delta Wbatch(n-1)$ and $\Delta Vbatch(n-1)$, for every pattern during that update period. The approximation may not be as accurate as using single update, but the advantage is realized when many more trials are performed over the same period of time. This seems appropriate because when the network begins training, the weight structure is not at all representative of the pattern set. The global weight update procedure dampens the effect of spurious weight shifts between patterns since one batch update is calculated instead of many pattern updates. As the network converges, the weight changes become smaller since the weight adjustments become smaller. During the final stages of training, the effect of the batch weight adjustment represented in the second term of Eqs. 3.5 and 3.9 influence the network less and the first term of Eqs. 3.5 and 3.9 becomes more dominant. Equation 3.13 describes how the batch weight updates are accumulated between the hidden and output layers. *ΔWbatch(n)*$_{jk}$ represents the accumulated weight change and *ΔWbatch(n–1)*$_{jk}$ represents the weight change value from the previous batch update. As in the single update process, Eq. 3.8 is used to calculate the error signal at the hidden layer. The hidden-input batch weight changes are accumulated over all patterns using Eq. 3.14 where *ΔVbatch(n)*$_{ij}$, represents the accumulated weight change and *ΔVbatch(n–1)*$_{ij}$ represents the weight change from the

$$\Delta \mathrm{Wbatch}(n)_{jk} = \alpha\, \Delta \mathrm{Wbatch}(n-1)_{jk} + \sum_{p} (\eta\, \delta_k\, H_j) \qquad (3.13)$$

$$\Delta \mathrm{Vbatch}(n)_{ij} = \alpha\, \Delta \mathrm{Vbatch}(n-1)_{ij} + \sum_{p} (\eta\, \delta_j^{\theta}\, I_j) \qquad (3.14)$$

previous batch update. This procedure is performed for the entire training set. Once completed, the forthcoming weight changes, *ΔWbatch(n–1)* and *ΔVbatch(n–1)*, the new weights, and the new biases, are calculated using Eqs. 3.15 to 3.17 and Eqs. 3.18 to 3.20, respectively. The accumulated batch weight updates are added to the existing weight

$$\Delta W(n-1)_{ij} = \Delta \mathrm{Wbatch}(n)_{ij} \qquad (3.15)$$

$$W(n+1)_{jk} = W(n)_{ij} + \Delta \mathrm{Wbatch}(n)_{ij} \qquad (3.16)$$

$$\mathrm{OutputBias}(n+1)_j = \mathrm{OutputBias}(n)_j + \Delta \mathrm{Wbatch}(n)_{ij} \qquad (3.17)$$

$$\Delta V(n-1)_{ij} = \Delta \mathrm{Vbatch}(n)_{ij} \qquad (3.18)$$

$$V(n+1)_{jk} = V(n)_{ij} + \Delta \mathrm{Vbatch}(n)_{ij} \qquad (3.19)$$

$$\mathrm{HiddenBias}(n+1)_j = \mathrm{HiddenBias}(n)_j + \Delta \mathrm{Vbatch}(n)_{ij} \qquad (3.20)$$

values, *W(n+1)*and *V(n+1)* to obtain the new network weight values. The batch weight update values are also added to the biases, OutputBias and HiddenBias, in providing the new bias values.

3.2.4 Error Measures

Flexibility in grading is also accounted for in the BP network. This scheme involves four types of distance metrics which are used to determine the total internal error signal (refer to Eq. 3.3). The metrics include: (i) Hamming, (ii) Euclidean, (iii) Minkowski, and (iv) Chebychev. The Hamming metric is given by Eq. 3.21, where D_H represents the

$$D_H \triangleq \sum_{m=1}^{z} \left| Tr_{p,m} - O_m \right| \tag{3.21}$$

Hamming distance, $Tr_{p,m}$ represents the training pattern output of pattern p, and output neuron m, and O_m represents the output neuron value. The Euclidean metric is given by Eq. 3.22, where D_E represents the Euclidean distance.

$$D_E \triangleq \sqrt{\frac{1}{2} \sum_{m=1}^{z} (Tr_{p,m} - O_m)^2} \tag{3.22}$$

The Minkowski metric is given by Eq. 3.23, where D_M represents the Minkowski distance and B represents the power. The Chebychev metric is given by Eq. 3.24, where D_C represents the Chebychev distance. In most applications, including this thesis, the Euclidean metric is used. The flexibility in the other metric selections is available for experimentation. Also, the Euclidean metric represents the vocal energy more closely than do the other metrics [Cair90].

$$D_M \triangleq \left[\frac{1}{B} \sum_{m=1}^{z} (Tr_{p,m} - O_m)^B \right]^{\frac{1}{B}} \tag{3.23}$$

$$D_C \triangleq \max_{P} \left[\sum_{m=1}^{z} (Tr_{p,m} - O_m) \right] \tag{3.24}$$

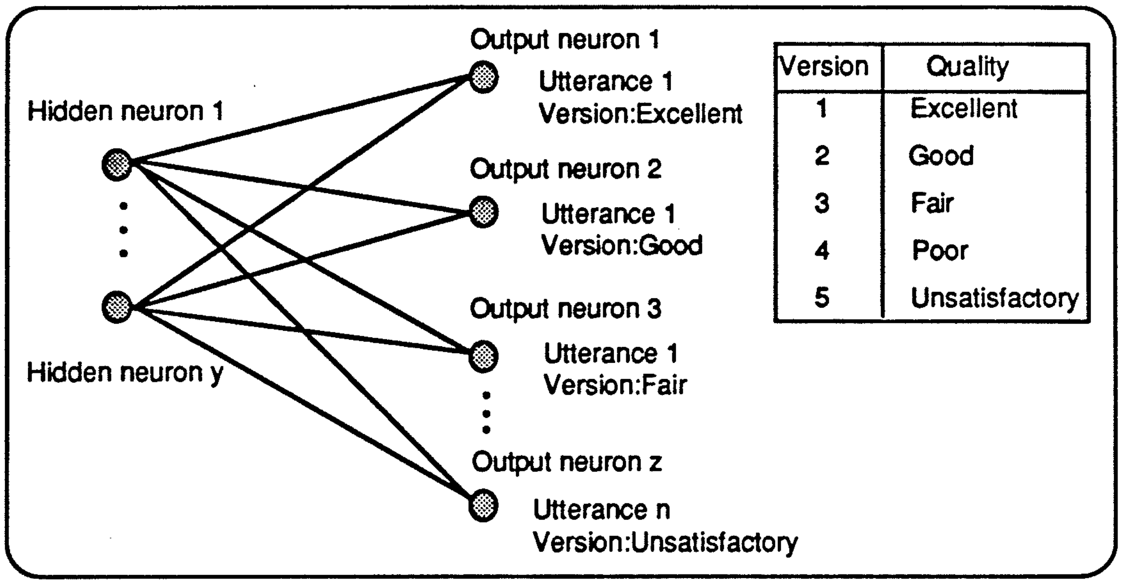
3.3 Recognition

Speech recognition is the key objective of this thesis. Recognition in this thesis means the identification of an unknown pattern from a known set of patterns based on a spectral feature. The spectral feature, obtained from the LPC coefficients, is embedded in the network weight structure. The known patterns are the pre-recorded utterances. The unknown pattern is the utterance response, emitted and recorded, from the autistic individual after hearing the target utterance synthesized from the host computer.

Every recognition system must contain the following functions: (i) data acquisition, (ii) templates generation and storage, and (iii) identification. This system contains all these functions, but the contribution enters in how each step is implemented. The data collection involves obtaining the LPC coefficients from the sampled speech data. The *inherent* information contained in the LPC data is passed into the neural network weight structure through the BP training process. In this step though, the template features are acquired by the network weight space. In conventional methods, representation of each template is contained in independent memory locations. The neural network approach requires initial overhead in the form of training prior to use, but it also introduces the reality of real-time recognition during use. This is especially true in the case of larger libraries, since any unknown pattern only need pass forward through the network for the identity to be determined. In the conventional template comparison approach, each pattern has to be compared separately and the solution is not known until the entire set is evaluated. Using a large library, this may prevent real-time recognition.

3.3.1 Recognition and Multi-level Grading

Recognition, as used in this neural network, is commonly called *classification*. By this, the identification involves the selection of only one of the output layer neurons. After forward propagation of the unknown pattern has occurred through the trained network, the identity is determined by selecting the most positive output neuron. Embedded in the classification scheme of the neural network is a method of achieving multi-level grading. Associated with recognition of an utterance is the *identity* and the ***quality***. Each of the trained utterances may have up to four associated allophones. Each allophone represents a quality level of the target utterance and is assigned a single output neuron during training. This scheme is shown in Fig. 3.5. By using this scheme, the ability of multi-level classification is realized.



Version	Quality
1	Excellent
2	Good
3	Fair
4	Poor
5	Unsatisfactory

Fig. 3.5 Classification scheme used in recognition.

In speech shaping [Cair90], this approach is necessary because the target utterance may not be achievable from the outset. In the past models, 255 levels were used to grade an utterance, but these many levels were not meaningful to human assessors. Thus, only five levels are used here, and these non-linear levels are assigned to the desired output pattern by the human assessor at the time of recording. This approach reduces the problem of interpreting the quality assessment of results and may also reduce the disagreement between computer-human assessment.

3.4 Summary

This chapter describes both a review of the classical BP technique and a specific description of the particular implementation of the technique used in this work. This description includes a detailed explanation of the operation and use of BP as applied to isolated speech recognition. A section describing the implementation of the adaptive learning rule and batch network updating is included. Also, a brief description of the various metrics used in calculation of the total internal error is provided. A discussion of the multi-level grading as it relates to recognition of the utterances is presented. Multi-level grading is used since it is an essential component in the process of vocal shaping.

CHAPTER IV
SYSTEM DESCRIPTION

4.1 Introduction

This chapter is concerned with describing the isolated speech recognition system from a conceptual viewpoint. The reason for giving a conceptual description is to provide an overall understanding of what the system contains in it, how each component is related to the others, and to provide a perspective for the following chapter. The system is separated into two major segments. The first segment describes the computer hardware and software that is required. The other segment describes the system resources as well as the custom designed speech recognition software. Finally, two approaches to the recording and playback routines are contrasted. The one used in the thesis involves the use of an IBM AT computer together with a Macintosh IIsi computer.

4.2 The Speech Recognition System

The isolated speech recognition system includes four main components: (i) data acquisition, (ii) feature extraction, (iii) the neural network and recognition logic, and (iv) the human interface. Each component is written in software, although the data acquisition also includes some special hardware. A block diagram of the system is given in Fig. 4.1.

This system is designed with the goal of computer automated vocal shaping in mind [Desr90] and [Cair90]. The first approach involves the use of an IBM PC AT to acquire the data and obtain the LPC coefficients. Another approach, which is under development, involves the use of the Macintosh sound manager. In this approach, the system would be confined to the Macintosh. The software is compiled and compressed into one package and could be used by any Macintosh computer with a minimal RAM requirement of 2 Mb.

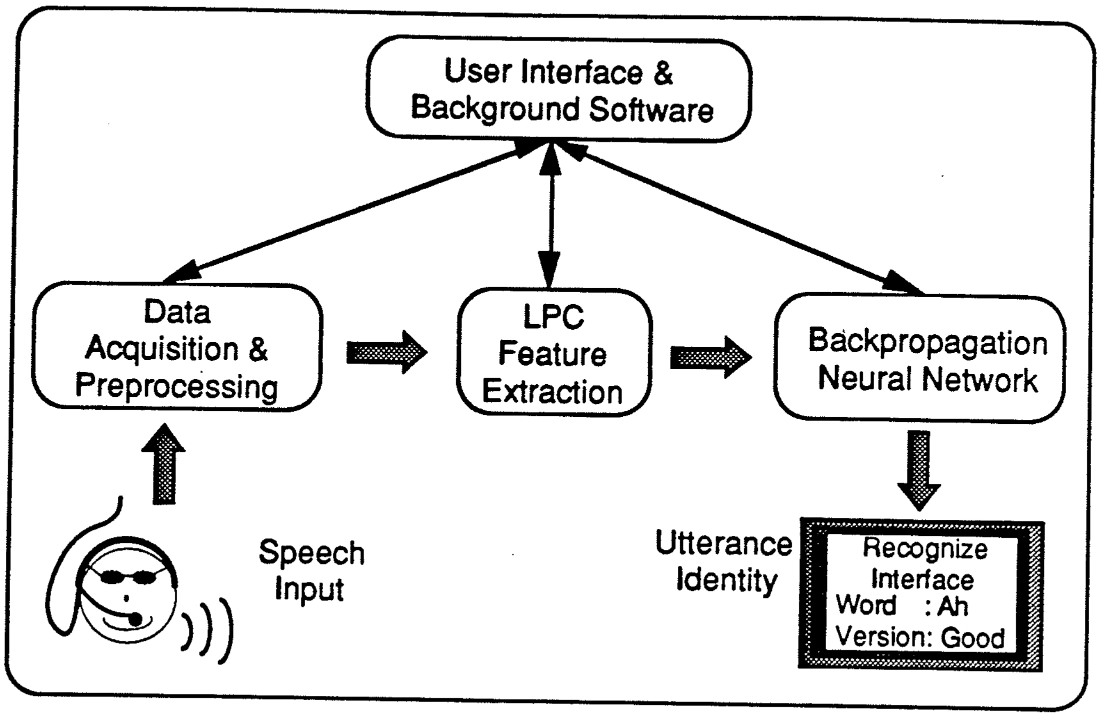

User Interface & Background Software
Data Acquisition & Preprocessing
LPC Feature Extraction
Backpropagation Neural Network
Speech Input
Utterance Identity
Recognize Interface
Word : Ah
Version: Good

Fig. 4.1 Speech recognition system block diagram.

The system currently operates with the assistance of an IBM computer and serial communication routines to obtain the necessary LPC coefficients. The system is developed on the Macintosh IIsi, which has an additional math coprocessor. The development software includes Think C object oriented programming language and the Macintosh resource editor, ResEdit 2.1.3b. Speech is captured using a Shure uni-directional, noise cancelling microphone connected into an AKAI tape deck. The AKAI tape deck filters the signal and passes it to the NEC DSP board which further filters the analog signal down to the toll quality band. The NEC samples the analog waveform at 8 kHz using an 8 bit A/D converter. These samples are passed through a LPC program which selects three frames from the data file and obtains 10 predictor coefficients per frame. These coefficients are sent serially to the Macintosh, catenated into a sequential pattern format, and trained by the BP neural network. The neural network later accepts an unknown pattern which is acquired and prepared in a similar fashion to the training templates. This pattern is identified by a recognition routine which interprets the neural networks output.

4.2.1 Resident System Tools

The system tools include all that hardware and firmware that is supplied by the computers. The ROM of the new Macintosh includes a very powerful sound manager that is capable of managing sounds easily. Also included in the software, are special serial drivers contained in the Macintosh toolbox and in Think Cs resource templates. This combination of toolbox drivers and software drivers permits very convenient serial communication between computers.

4.2.1.1 System Hardware

The speech processing system hardware consists of the Macintosh IIsi computer, the floating point card, the IBM PC AT portable computer, the NEC processor, the communication cables, and the Shure microphone. A minimum of 2 Mb of RAM for the Macintosh is required to run the software and a hard disk would be useful in maintaining the utterance library. The Macintosh uses a MC68030 processor running at 20 MHz and a MC68882 floating point coprocessor which is used to increase the number of floating point calculations done per second. Many such calculations are required during training of the neural network. The APW is acquired using an uni-directional, noise cancelling microphone which is connected into the right channel microphone input of an AKAI tape deck. The right channel tape deck output is connected, using 1/4" stereo jacks, into the analog input, J35, of the NEC EDSP-77230 DSP board. A *C* program configures the DSP board which acquires the analog speech data at the request of the certain serial communication messages existing between the two computers. The program extracts the desired speech segments, obtains the LPC coefficients, and transmits them serially to the Macintosh. Serial communication occurs through *com2* to the Macintosh serial (modem) input port. A block diagram describing this configuration is given in Fig. 4.2. In the alternate scheme that is currently under development, the data acquisition is performed

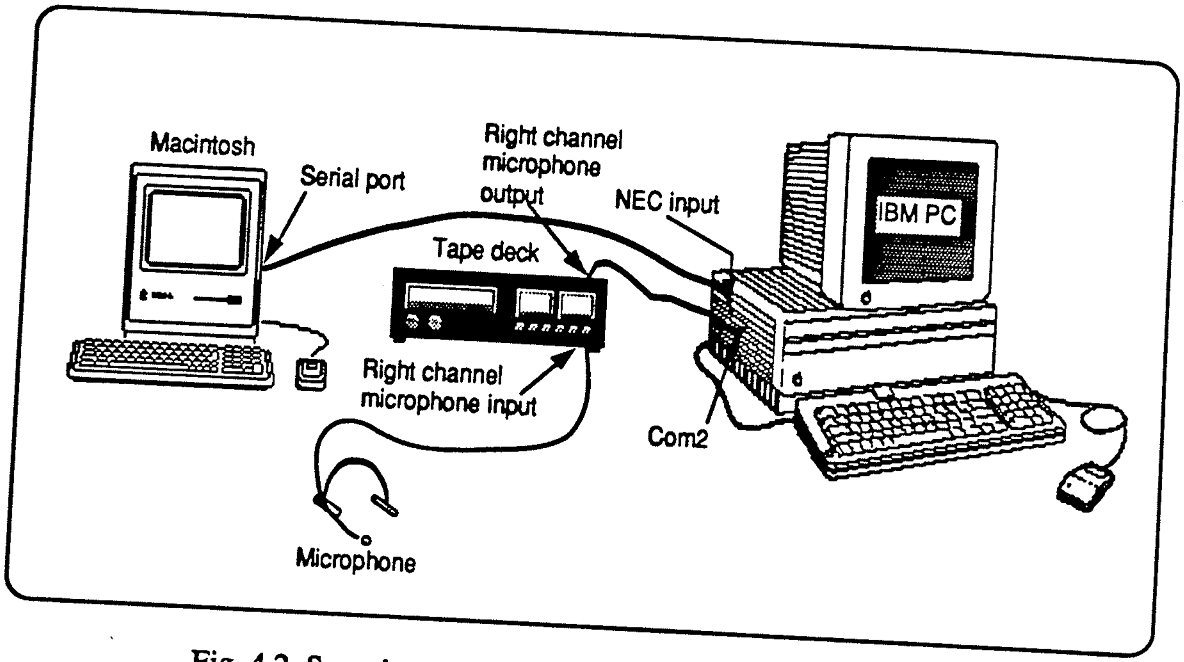

Macintosh
Serial port
Right channel microphone output
NEC input
IBM PC
Tape deck
Right channel microphone input
Com2
Microphone

Fig. 4.2 Speech recognition system hardware configuration.

using a Sony A/D and D/A converter, and the Apple sound chip (ASC) found in the Macintosh IIsi. The speech may be acquired by using the built-in *electret* omni-directional microphone provided with the computer or any other microphone that has an 1/8" input jack. An external audio 1/8" jack is also provided on the Macintosh to provide any audio playback. In the vocal shaping process, this would be a necessary requirement.

4.2.1.2 System Firmware

The second alternative to data acquisition, recording, and playback, involves the newly developed sound manager found in the new 512k Macintosh system ROM. The new sound manager provides sound processing abilities such as *RecordToFile* and *PlayFromFile*. Also, it hosts a large set of lower level commands called *Sound speech blocks* (SPB). Although the sound manager has a high level command that brings up a sound editing dialog box, it is not desired because this procedure stores the data and sound header to a sound resource where all the recorded data would be laden in the application resources. This feature is made available for recording and storing *ALERT* sounds to the application software or Mac OS resources. *ALERTS* are used for warning the recipient of a possible danger, or to notify them of a possible recourse from an incorrect system procedure they have initiated. Thus, this approach could not be used for the above reason and since multiple sets of library data would necessarily require separate file space. Hence, the choice to use the SPB commands, which provide extended flexibility in the analysis, recording, playback, saving, and loading process are chosen. Further, the sampling of the speech input is selectable at 5 kHz, 7 kHz, 11 kHz, 22 kHz, or 44 kHz. The analysis of the waveform is performed through Apple's proprietary technique known as *Macintosh audio compression expansion (MACE)*. The waveform is analyzed using 8 bit samples and a monaural channel. Also, the sound can be analyzed and compressed in real-time by 3:1 or 6:1 factor, or sampled as PCM. Storing of the sound data requires either a *resource* type header or standard file type header. The *resource* requires an ID and a predetermined header structure. For further details concerning this structure, refer to [Appl90]. The other format, known as *Audio interchange file format (AIFF)*, is a standard format used to store digital data to a file [AIFF89]. This format is used in this thesis and is supported by Apple Computer in their new sound manager. It requires a header structure which is described in [AIFF89] and [Appl90]. The header contains information about the sampling rate, size of data, file name, compression type, and file size. Once the data has been stored using the AIFF format, the file can be played directly from disk, loaded and then played, or played directly from RAM following recording. In addition to this, the computer is able to perform real-time compression and storage, and real-time expansion and playback directly from disk if necessary. Since a custom interface is used in this thesis, the low level SPB calls are used to facilitate the structure of the software. A standard recording framework is proposed which includes: (i) 22 kHz sampling rate, (ii) 8 bit samples, (iii) one second recordings, (iv) no compression and, (v) a monaural channel. The data necessarily requires that it be uncompressed initially because of the need to obtain the LPC filter coefficients. Once the values have been obtained, the data may be compressed and stored if disk space is an issue.

Other pieces of firmware located in the system ROM include the serial driver, which is used for serial communication between the IBM and the Macintosh computers, the dialog manager, menu manager, control manager, quick draw manager, and text editing manager. These managers provide low level functions that are used in the software. For a further description of these managers, refer to [Appl88].

4.2.2 New Tools

In order to develop the software, a language which possesses its own set of commands, is capable of accessing the system firmware managers, and can also communicate with the hardware via the serial communication channel is necessary. The software development is performed using Think C (object-oriented version) on the Macintosh computer. A set of objects, provided with the software, and various additional libraries are used in the software. Since great detail will be used to describe these in Chapter V, an overview is provided here.

4.2.2.1 The Human Interface

The speech recognition system software contains a very complex, yet easy-to-use, human interface. The purpose of the interface is to provide the recipient with feedback which will guide them easily through the software and provide the necessary information about the operations during a session. Its other objective is to provide all of this information, whether it be in the form of sound or text, in a concise form that will provide the individual with a clear understanding of what actions are to be taken, or have occurred. The speech recognition software conforms to standard Apple interface guidelines so that knowledge gained from using past software can be applied to the current speech recognition software. For example, when a dialog window appears, it is placed in the standard position on the screen and, the default button such as *DONE* or *OK* is highlighted in the prescribed fashion to indicate the default action. If the recipient decides to press the *RETURN* key, instead of clicking the button, the default action is still initiated. Similar types of parallels have been embedded in the remainder of the software.

4.2.2.2 Data Acquisition and Feature Extraction

The software must first acquire the raw analog speech data and extract the necessary component of the speech waveform before the recognition procedure is performed. This procedure involves sampling the input channel at 8 kHz and 8 bit samples. The raw speech data is stored temporarily in RAM while another procedure obtains the necessary LPC coefficients. Once the LPC coefficients are obtained, the software would play back the sound directly from RAM using the raw PCM samples or store them for future reference using the Oki ADPCM speech chip analysis technique.

Since much effort has gone into the development towards the single CPU employing the Macintosh and the resident sound manager, the development using the IBM-Oki ADPCM sound storage and playback routines have been replaced with development routines to support the single CPU approach. A beta version, employing the system resources, of this playback scheme is present in the current version of the speech recognition software. Using this alternative scheme, the storage would be done using the AIFF file format on the Macintosh computer, possibly using 3:1 or 6:1 data compression. Essentially, all recorded sounds would remain in separate files which the speech recognition software would access and playback and uncompress, if necessary, in real-time from disk file upon request of the speech recognition software.

4.2.2.3 Training and Recognition

Following the acquisition of the data, production of the LPC coefficients, serial transmission to the Macintosh, conversion to the proper data type and pattern format specifications, the software trains the network using the data and later uses it to recognize new utterances. The neural network architecture is determined through the use of dialog boxes and pull down menus. The configuration options are made clear during selection of a custom network architecture, and a fall back position to a default configuration is present if no configuration is entered. Also, if the decision to discard the changes halfway through the configuration procedure is made, an option to remove the new information is available which reverts to the default configuration. With the data set in place, the network is trained. The network training dialog box displays the important parameters indicating the network's progress. An *Exit* button in the dialog box is present to halt training while maintaining the current weight structure. Also, a menu command to re-initialize the weight structure with new random weights is available. Testing, following convergence of the network is performed by selecting the *TEST NETWORK* command. Network training or testing is not permitted prior to recording or loading of a data pattern set. An *ALERT* box indicating the problem and a plausible recourse is displayed. Since testing of the network is optional, immediate use of the recognition routines following training is permitted. Selecting *RECOGNIZE* under the *RECOGNITION* menu displays a dialog box for recognition. In this box, an utterance combination from the pre-recorded library is selected. Actions to playback the selected utterance prior to recording the test utterance are permitted. Clicking the *RECORD* button begins the recording of the test sound. This sound is prepared in the same way as the library utterances. The LPC coefficients are obtained, transmitted to the Macintosh, and presented to the input of the trained BP neural network. The output vector is thresholded using recognition software and the identity is displayed in the dialog box. The identity consists of the utterance class and the version of that class. This is the same combination used in selecting an utterance for playback. This procedure encompasses the fundamental steps that constitute vocal shaping and the operation of the speech recognition software.

4.3 Summary

This chapter describes the speech recognition system from a fundamental level. The system includes: (i) data acquisition, (ii) feature extraction, (iii) the neural network and recognition logic, and (iv) the human interface. Each section is described with respect to its purpose in the system and relationships with its neighbors. A detailed description of each of these components are presented in Chapter V. Two versions of data acquisition are contrasted which involve the NEC-IBM and the Apple Macintosh sound chip. Current operation of the software employs the IBM-NEC DSP for both acquisition and obtaining the LPC predictor coefficients.

CHAPTER V
SOFTWARE IMPLEMENTATION

5.1 Introduction

The software is the integral heart of the system. It performs data acquisition and management, training and testing of the neural network, and performs utterance recognition. A description of how the software accomplishes its many tasks is achieved through the use of structure charts. These charts provide an informative insight into how the various operations are actually accomplished without the source code details. If however, the source code is required, a complete and current listing is included in Appendix C.

5.2 Software Background

Before introducing the software that is used for speech recognition, a background is established to understand the operational structure of the object-oriented methodology. Object-oriented approaches are available using *C* or *Pascal* language on the Macintosh computer. The software is written in object-oriented *C*. A class library of predefined objects is provided with the software, and since some of these objects are used, a formal discussion of the ones used is necessary. A set of relationships exists between many of the objects. To begin, the software global functions are shown using the main menu given in Fig. 5.1. The structure of the *C* class objects is given in Fig. 5.2. The class hierarchy, given in Fig. 5.2, describes the relationships between the objects provided in the software package. All the objects originate from the root object called *CObject*. All class objects begin with a capital *C* but for clarity the *C* has been dropped. Following the class hierarchy, is the flow, or chain of command. The chain of command describes how data is treated and who receives the first opportunity at processing it.

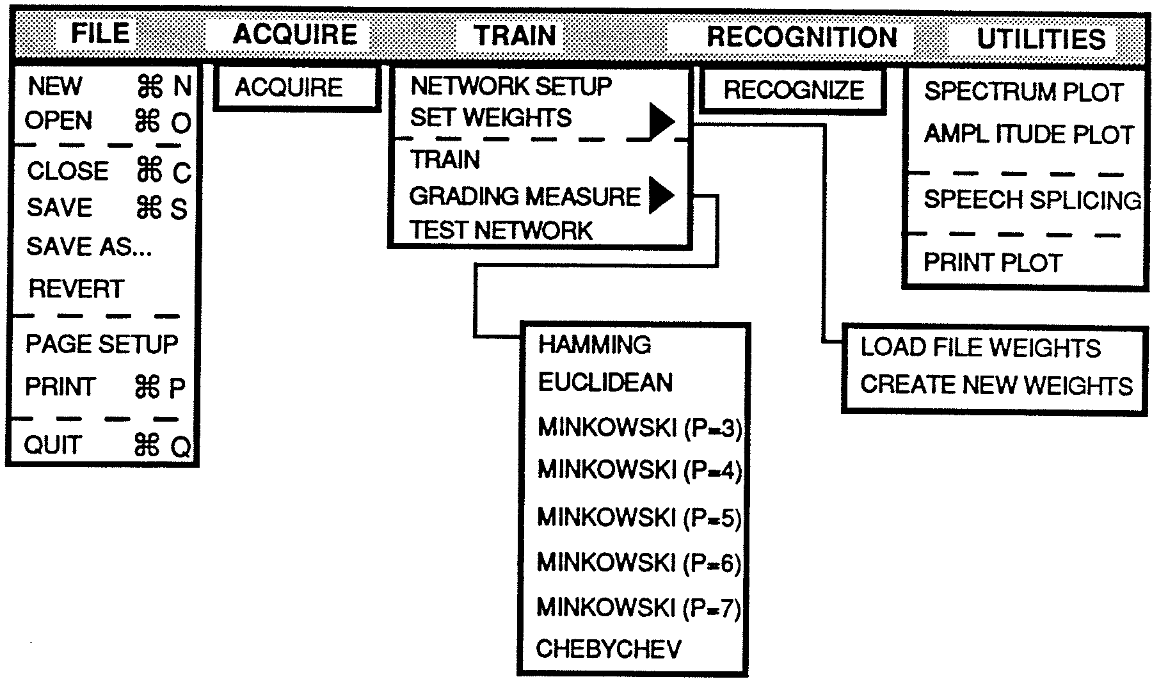


Fig. 5.1 Functional overview of the speech recognition system menus.

In many cases, descendants call upon the functions of their successors for processing knowledge. This form of behavior is known as inheritance. Inheritance allows a descendant object to be able to process methods that its predecessor objects are capable of processing. This form of linking is signified by the word *inherited* followed by a double colon and then the method that the descendant wishes to inherit. For example, if a descendent of *CDocument* called *CBackpropDoc*, wishes to inherit the *DoCommand* structure of *CDocument* then all that is necessary is to insert the following command, given by Eq. 5.1, into its *DoCommand* method. Then, whenever the *DoCommand* method of *CBackpropDoc* wishes to process a message that it does not explicitly contain, it will be able to use the inherited structure of its predecessor to assist it.

```
inherited::DoCommand()
```
(5.1)

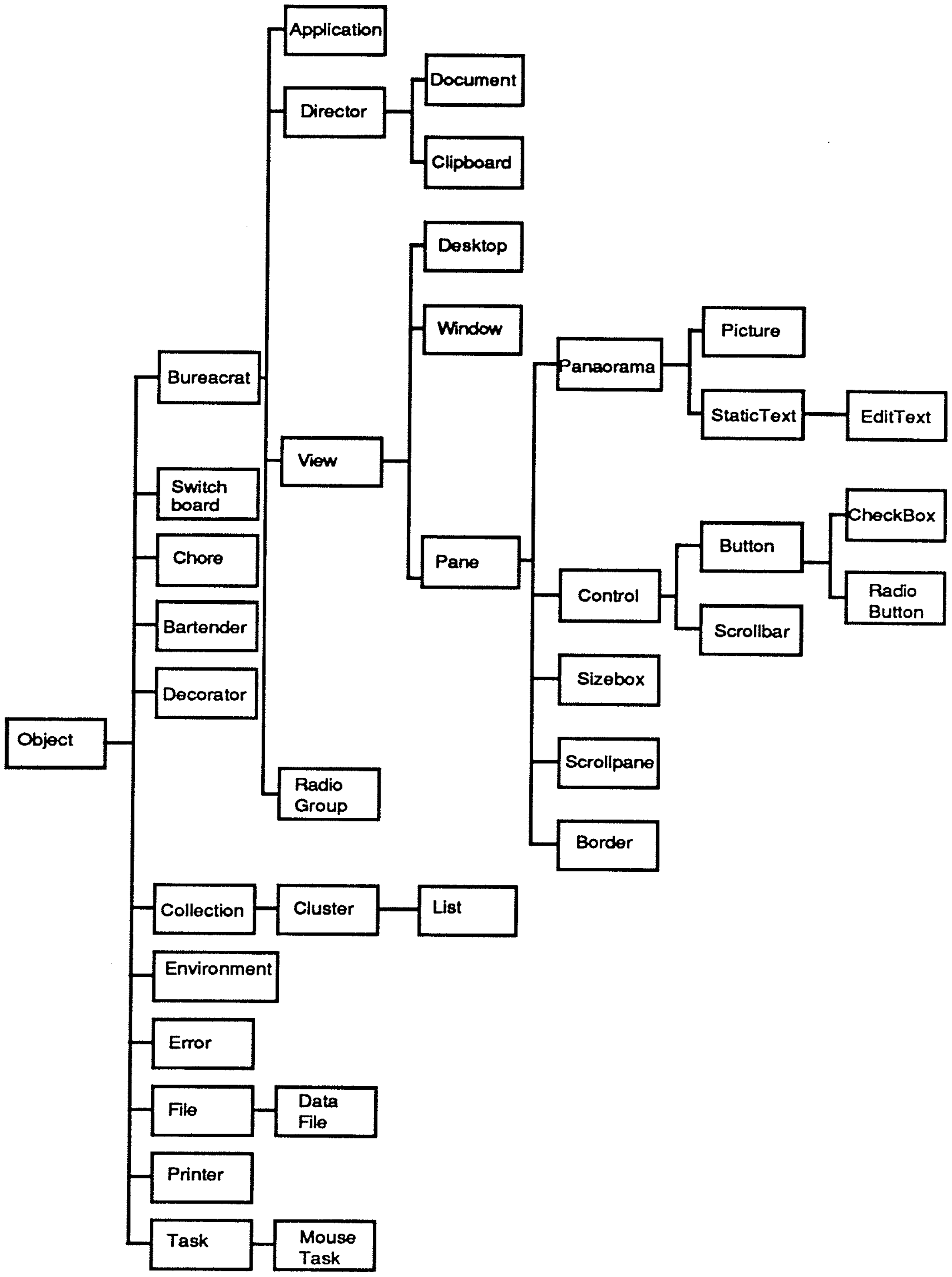


Fig. 5.2 Think C class hierarchy (after [Syma90]).

The flow of control begins at the object *CSwitchboard* picking up a message and decoding it. The four decoding groups are: (i) menu, (ii) window, (iii) key events, and (iv) system control. Based on these main areas, the switchboard routes the control to the highest object of that class. If this object can not process the message, it passes the message down the hierarchy to the next predecessor. These predecessors in turn attempt to process the message. If the message reaches the last link in the hierarchy, and is still not understood, a system error will occur if the last object can not correctly process the message. Typically, the command is carried by a global variable available to all objects known as the *gGopher*. The flow of control describing this structure is given in Fig. 5.3.

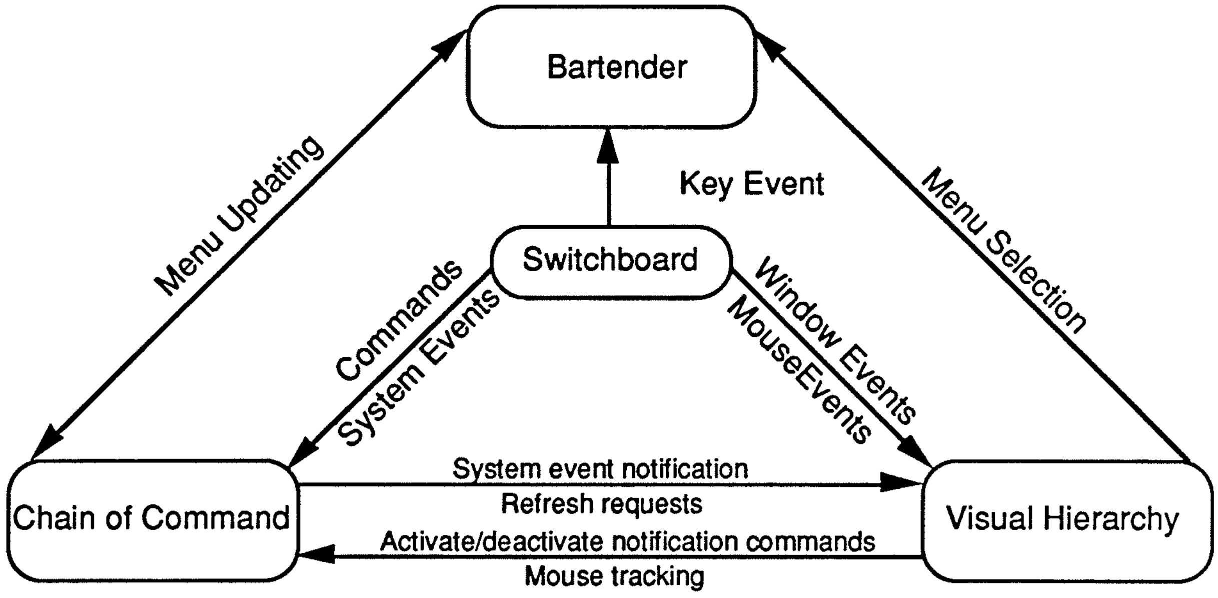


Fig. 5.3 Flow of control in object-oriented programming environment (after [Syma90]).

The difference between procedural and object-oriented programming is the way data is treated. In procedural programming, data and the functions are treated separately; first the data structures are written and then the routines to operate on them. In object oriented programming, action and the data are closely tied. In some cases though, it is better to use procedural strategies to perform a task. A diagram of the structure of an object is given in Fig. 5.4.

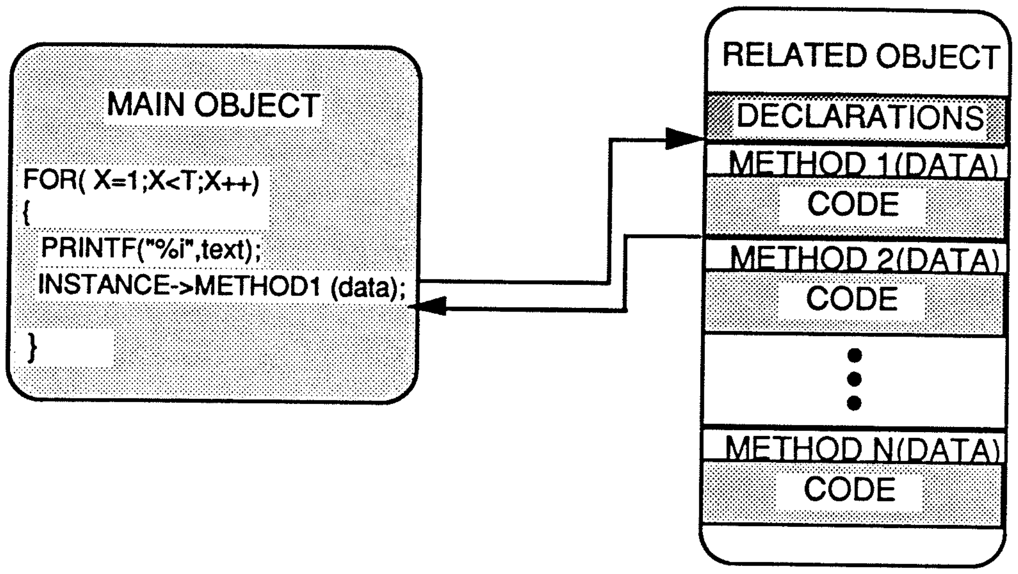


Fig. 5.4 Structure of an object sending a message.

The outline of the software is based on the class hierarchy given in Fig. 5.5. This structure depicts the specific methods used in the software. Three main objects are present in the software. The first object, *CBackpropPane,* is a descendant of *CPane.* This object possesses the inherent structure of *CPane* and is used to process any form of window intensive feature, such as a graph, that may be developed for the *UTILITIES* menu in the future. The second object is *CBackpropApp.* It is a descendant of *CApplication* and is used to initialize the necessary application parameters and to create and open the documents. The last of the three main objects is *CBackpropDoc.* It is a descendent of *CDocument* and possesses all of the necessary file handling abilities. One other descendant that is used periodically is *CDataFile,* also a descendant of *CDocument.* The *CDataFile* object is used for reading and writing tasks.

A working knowledge of object-oriented language would be a great asset although this is not necessary to understand the software. To facilitate the understanding of the software, structure charts are provided. Prior to describing the central software which is

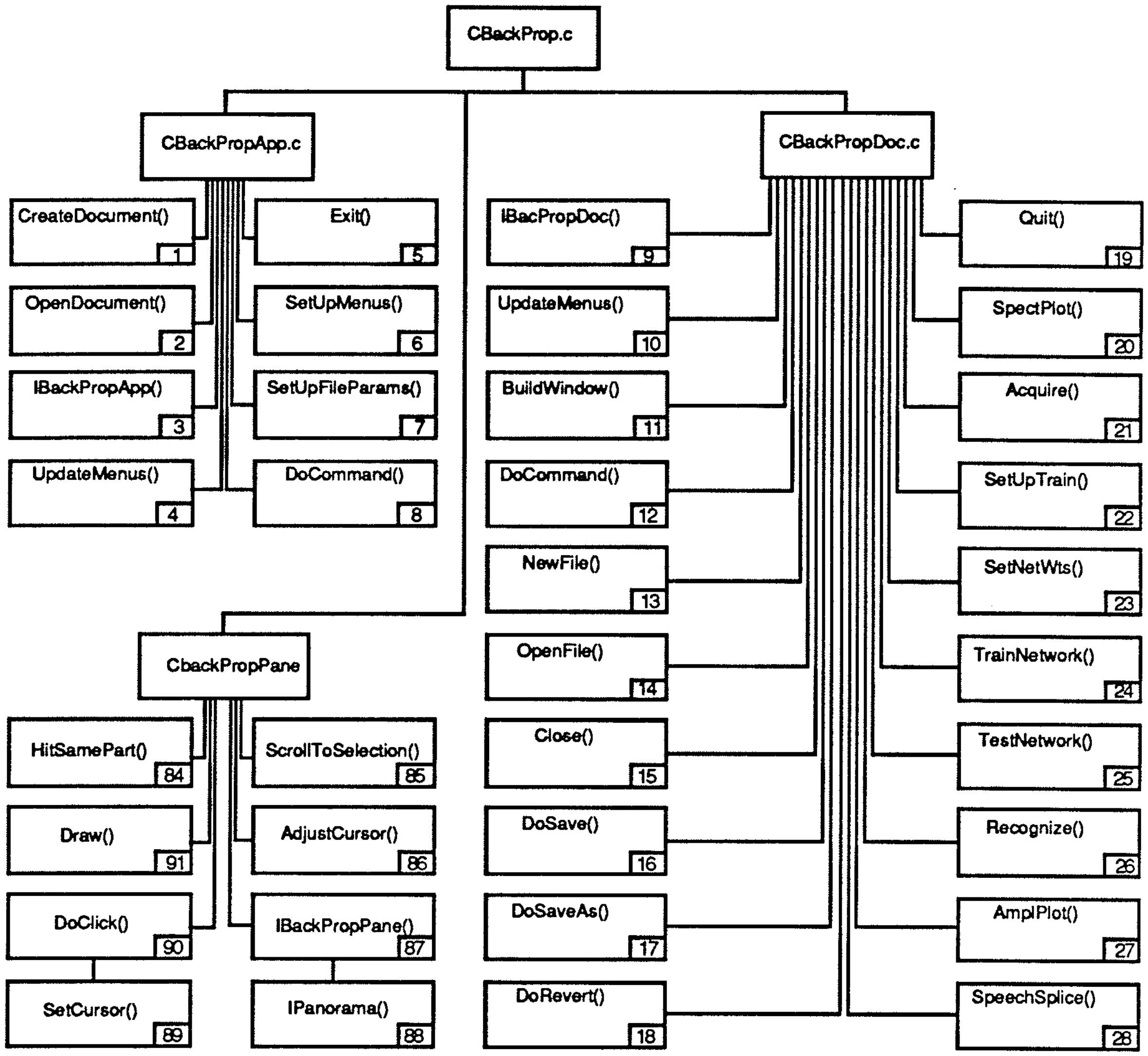


Fig. 5.5 Global software structure.

contained in the *CbackPropDoc* object, the application object is first described. To begin operation, initialization and allocation of memory is performed by *CBackpropApp* which inherits its structure from the predecessor class object, *CApplication*. Basic initialization steps include: (i) loading the menu resources (*SetUpMenus*), (ii) updating the menus in the window (*UpdateMenus*), (iii) creating a document (*CreateDocument*), (iv) opening the document (*OpenDocument*), and (v) initializing the necessary memory and system managers (*IBackpropApp* and *SetUpFileParameters*). A structure chart showing the key programs, relationships, and methods of interest is provided in Fig. 5.6.

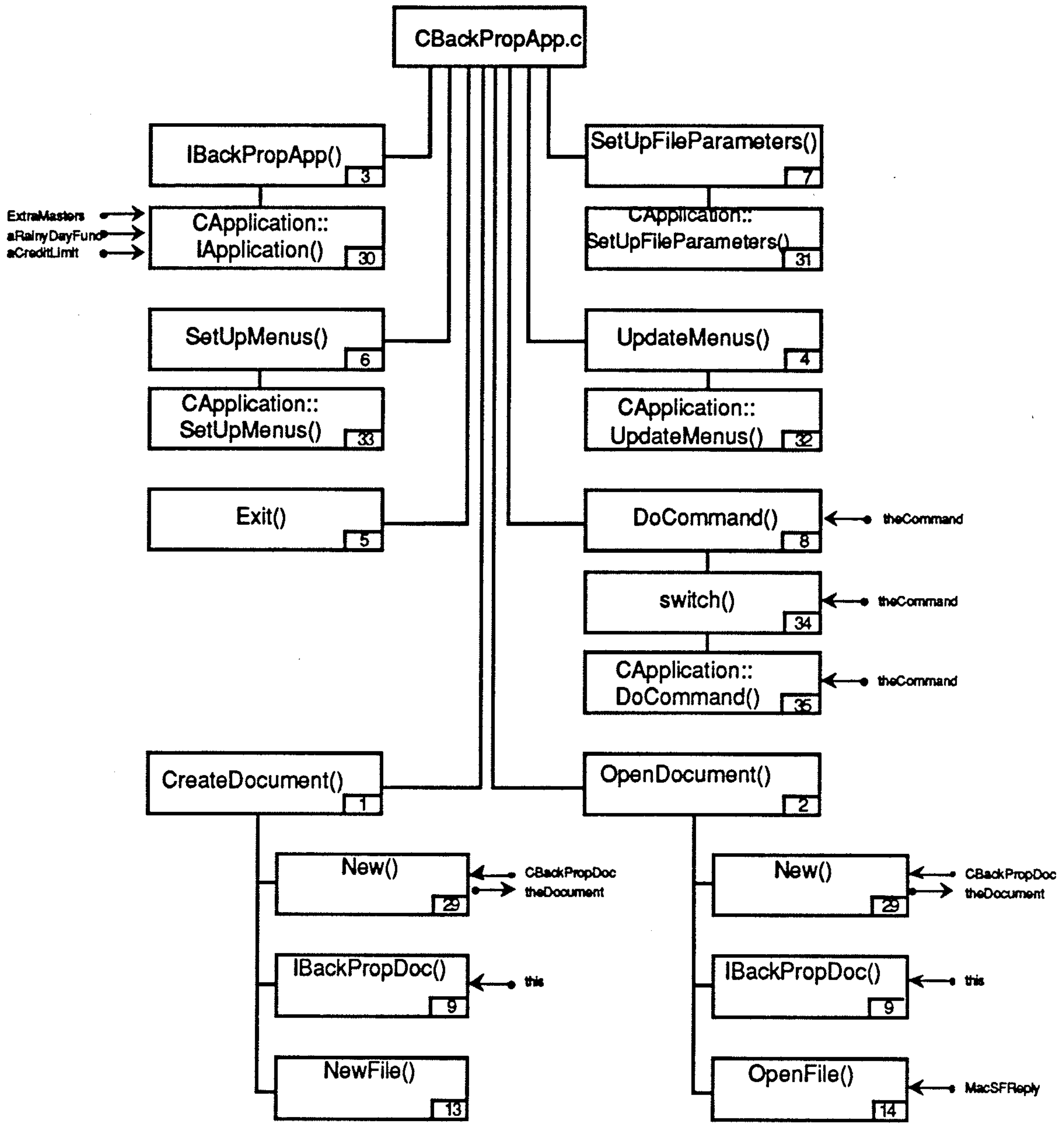


Fig. 5.6 Application object.

As in the example provided describing *inheritance*, *CBackpropDoc* does inherit the structural information from *CDocument* for commands such as *NEW, PAGESETUP, PRINT, QUIT, OPEN, CLOSE, SAVE, DOSAVEAS,* and *QUIT*. The structure of the command delegation is found in the *DoCommand* method given in Fig. 5.7. This structure chart describes the paths to all of the relevant methods used in the software. All of the remaining structure charts are from the document object. The first section to be discussed describes the interface to the software, which is referred to as the *human*

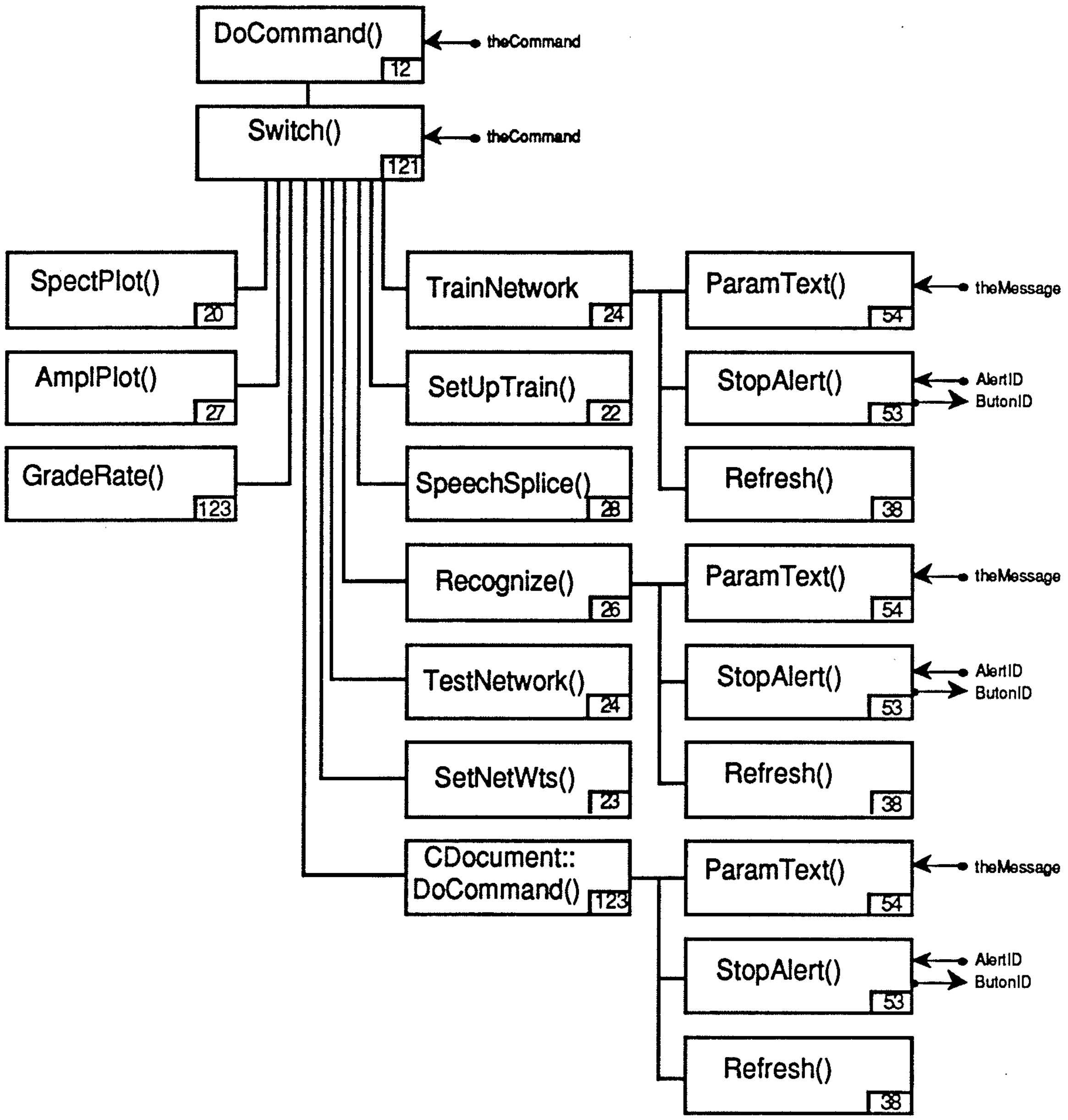


Fig. 5.7 *DoCommand* method.

interface. The other inherited commands are shown in the following set of structure charts. The only other command that is inherited, but not a menu command, is *BuildWindow*, which constructs a window for use by the application, and assigns *gGopher* to itself in order to receive commands from this window as described in Fig. 5.3.

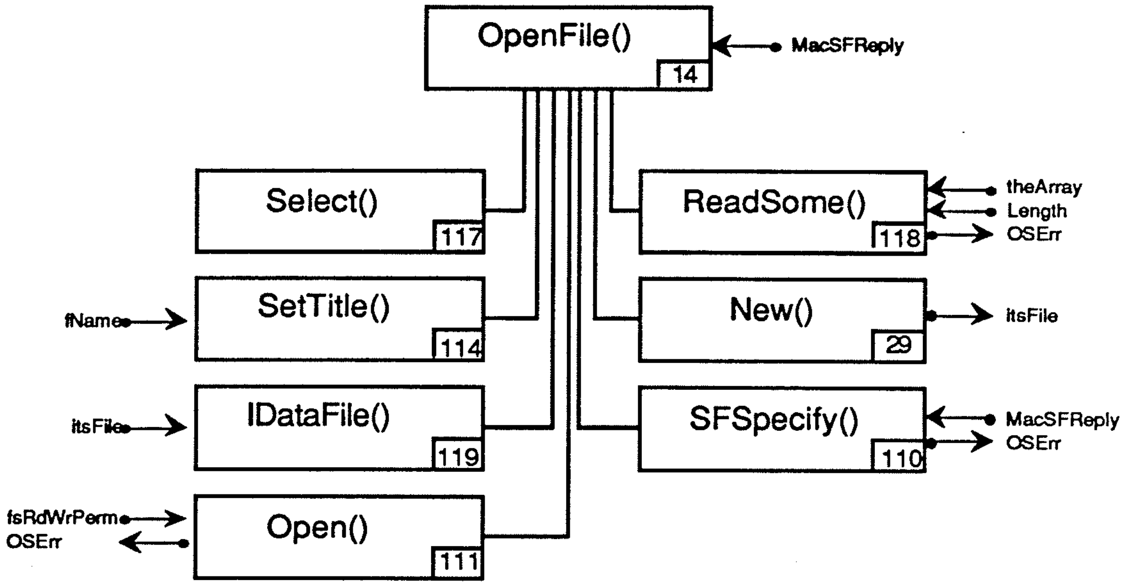


Fig. 5.8 *OPEN* command.

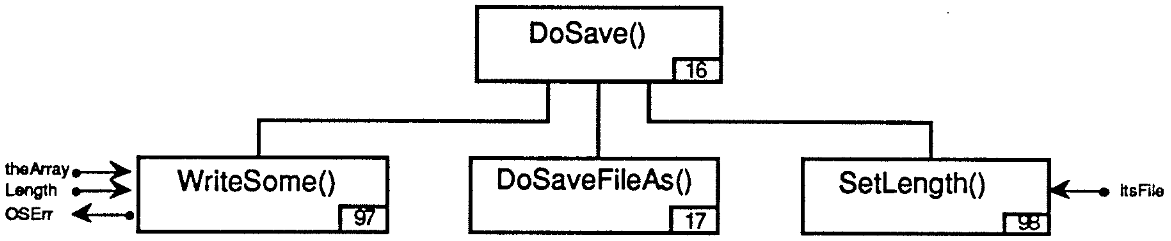


Fig. 5.9 *SAVE* command.

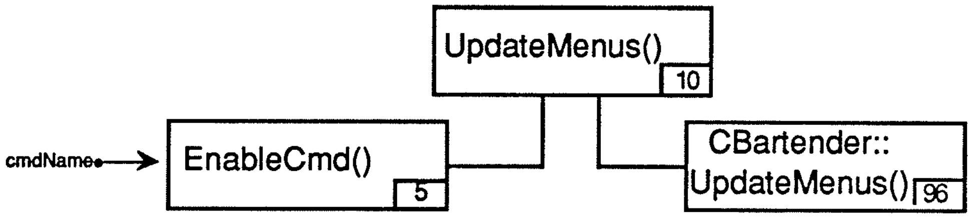


Fig. 5.10 Updating the menus.

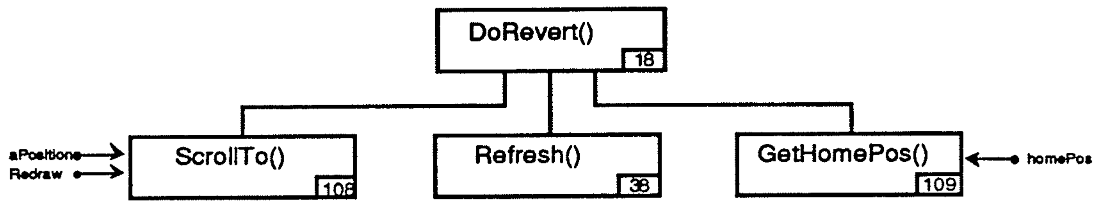


Fig. 5.11 *REVERT TO SAVED* command.

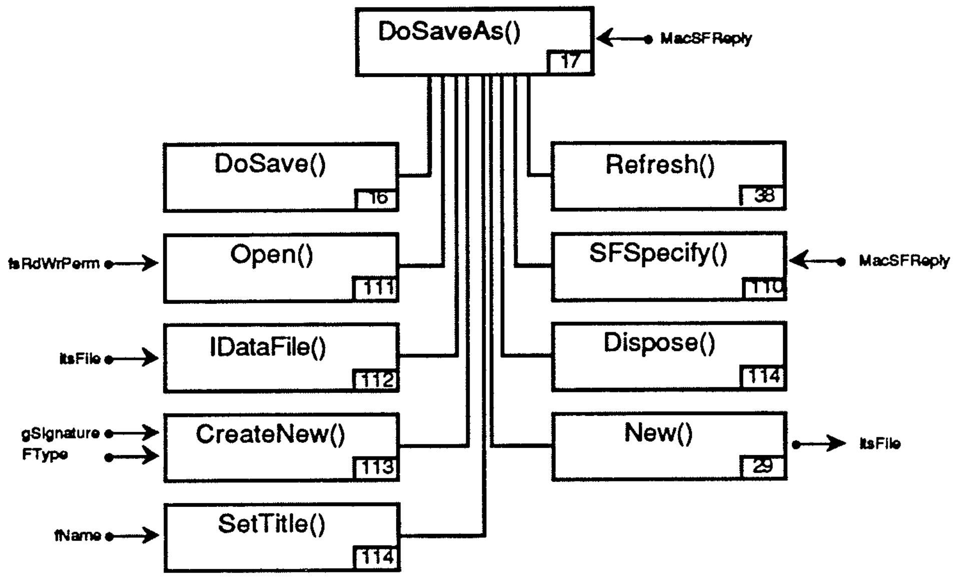


Fig. 5.12 *DOSAVEAS* command.

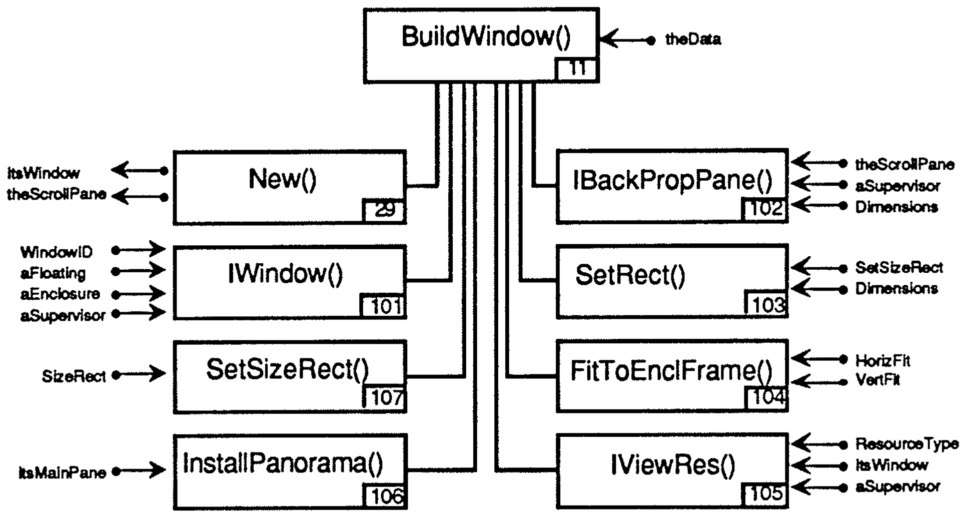


Fig. 5.13 Build window method.

5.3 Human Interface

A program is good if it is easily understood. Much effort has gone into the design, content, and control of the human interface. The human interface provides the necessary interface to the software. All input and output is routed through the various modules contained within the interface. These include: (i) a menu bar, (ii) pull down menus, (iii) hierarchical menus, (iv) dialog boxes, (v) alert boxes, and (vi) descriptive indicators and sounds about the software environment. In previous systems, very poor interfaces were present. The descriptions of various functions and results, together with the operation, were difficult to understand and not conducive to extended session use.

5.3.1 Initialization

The first step in using the program is to double click the application icon. What appears next is the main menu bar (see Fig. 5.1). The only enabled selections at this point are *NEW*, *OPEN*, and *QUIT* which are contained in the *FILE* menu. Assuming nothing exists, *NEW* is selected and the library entry dialog box appears. The dialog boxes used in this thesis are modal dialog boxes. Modal dialog boxes restrict the active environment to the dialog box until either the approval or cancellation button is selected. In the library entry dialog box, the names of the individual and therapist and up to ten utterance labels to be used during the session are entered. Following these entries, correct acknowledgement is achieved by clicking the *DONE* button which returns control to the main menu bar. Using Apple standard interface methods, a default button is indicated by bold highlighting surrounding the button. This is typically the button the person would click during a normal course of action. It is also the button that will become active if the person chooses to press *RETURN*. A snapshot of the *NEW* dialog box is found in Appendix A. A structure chart of the *NEW* method is given in Fig. 5.14.

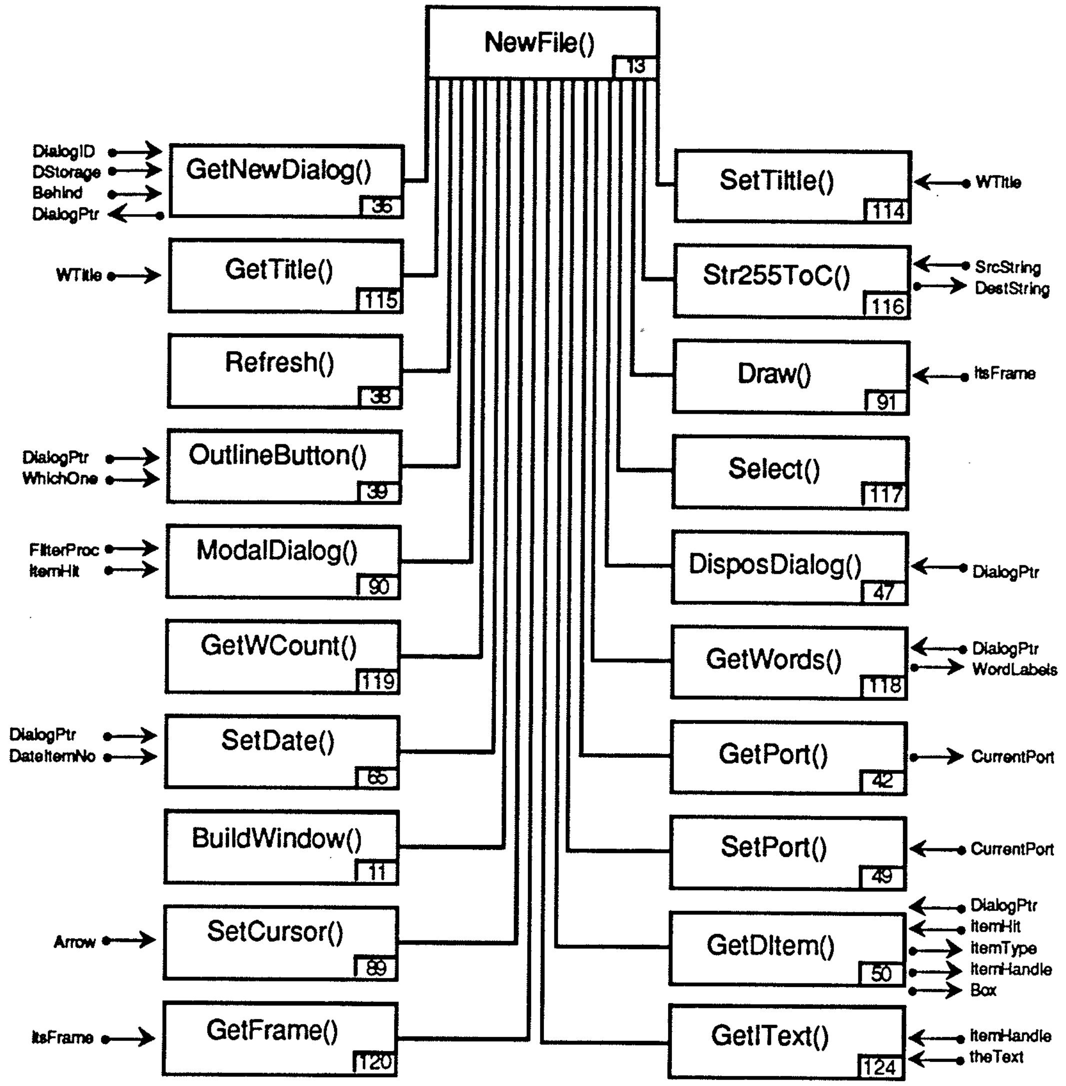


Fig. 5.14 *NEW* command.

5.3.2 Recording

The next step in the speech recognition and shaping process is to record the utterances. Activating the record dialog box is done by selecting *ACQUIRE* from under the *ACQUIRE* menu. In the record dialog box, the utterance and version to be recorded is selected using radio buttons. The utterances, which were entered into the *NEW* dialog box and labelled for the session, are copied into this dialog box to make the recording selection process logically intuitive. In addition to the ten possible utterance labels, up to five radio buttons may be used to indicate the recording quality. The quality selections include: (i) Excellent, (ii) Good, (iii) Satisfactory, (iv) Fair, and (v) Poor. Thus, during any one recording, a unique utterance and a quality version are selected. Below each version radio button are five check boxes (which are initially empty). These check box indicators are used to notify the person as to which utterance-version combinations have already been recorded, and will become checked once the recording has successfully finished. For example, if utterance one was previously selected with the *Excellent* version then an active check box will appear under the *Excellent* version whenever utterance one is selected, whether or not the *Excellent* version is selected. These check boxes change as different utterances are selected, reflecting the history of the recordings for a particular utterance. It is permitted to record overtop of an existing utterance but an *ALERT* box is displayed to notify the person that they are about to destroy the existing recording. Closing this *ALERT* box returns control to the individual to record over an existing utterance or, to change the version or utterance selection before proceeding. To record an utterance, the selection is made and the hardware activated. The instant that the record button is depressed, the IBM computer, through serial communication, begins to acquire data from the microphone and obtain the required LPC coefficients as described in Chapter II. These parameters are then serially transmitted to the Macintosh. A structure chart showing the flow of control and variables is given in Fig. 5.15. A snapshot of the Record dialog box is given in Appendix A.

5.3.3 Training

After all the utterances are recorded, the network is trained. The first step is to configure the network. This can be done by selecting *SETUP NETWORK* under the *TRAIN* menu. The network has a default configuration so if the person does not wish to become involved with custom configurations they can skip directly to training. The BP neural network can be custom configured for a variety of features.

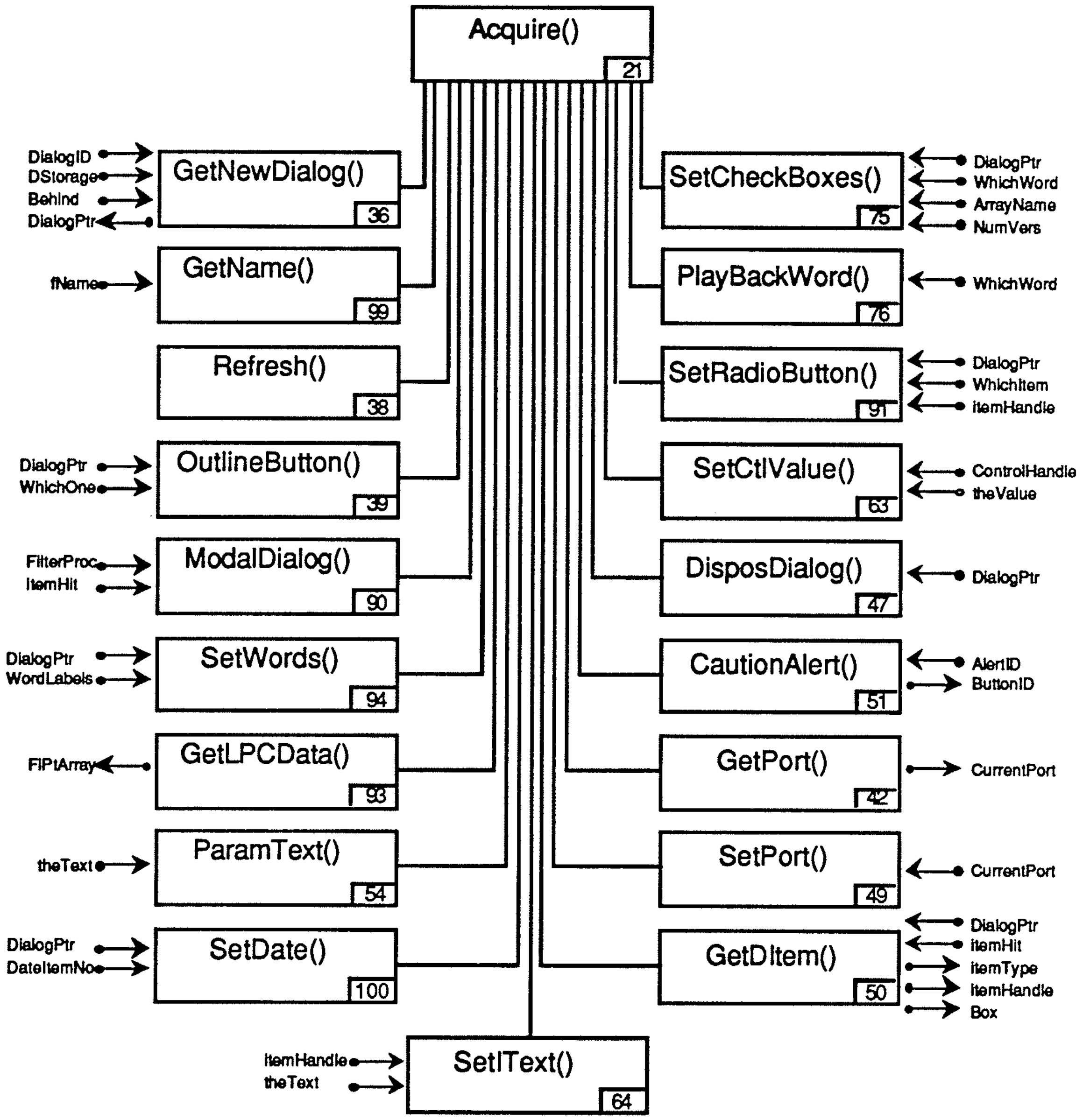


Fig. 5.15 *ACQUIRE* command.

Configuration parameters include: (i) random/sequential presentation of training patterns, (ii) biases on the hidden and output layers, (iii) initial settings of the momentum, learning rate and, stopping criteria, (iv) either batch or single updates for the weight space, and (v) variable number of hidden units. The input neurons can not be configured since every training pattern received from the IBM computer is a fixed size. Further, since the number of training patterns is known from the data file record, and since a classification scheme is used with the neural network, the number of output neurons is also automatically calculated. Contained in the *TRAIN* menu, is a command labelled *GRADING MEASURE*. Selection of this command, yields a further hierarchical menu selection adjacent to the selected one. The hierarchical menu consists of the following distance metrics: (i) *HAMMING*, (ii) *EUCLIDEAN*, (iii) *MINKOWSKI* (with powers of 3 to 7), and (iv) *CHEBYCHEV*. The metric is used to calculate the total internal error after forward propagation through the network. A structure chart describing the grading measure scheme is given in Fig. 5.16 and a structure chart describing the preparation of the neural network is given in Fig. 5.17.

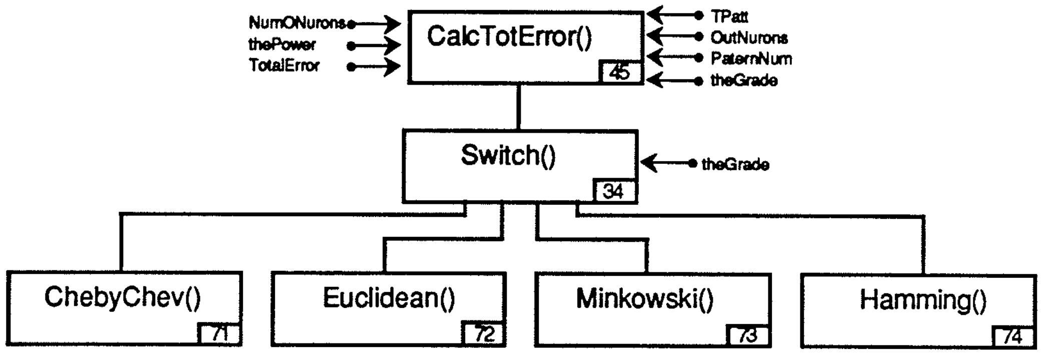


Fig. 5.16 Calculation of the neural network total internal error.

Following the network configuration, the *TRAIN NETWORK* command is selected from the *TRAIN* menu. Upon selection, the network begins training using the specified configuration. A special modal dialog box is displayed to inform the person of the networks status and progress. The dialog box displays the *Trail Number, Learning Rate, Pattern Number,* and *Total Internal Error*. The only interface feature is an *Exit* button. Allowing the network to train until the total internal error falls below the threshold (stopping) value will also return control to the main menu. If some parameters require adjustment, selecting the *Exit* button will halt training and returns control to the main

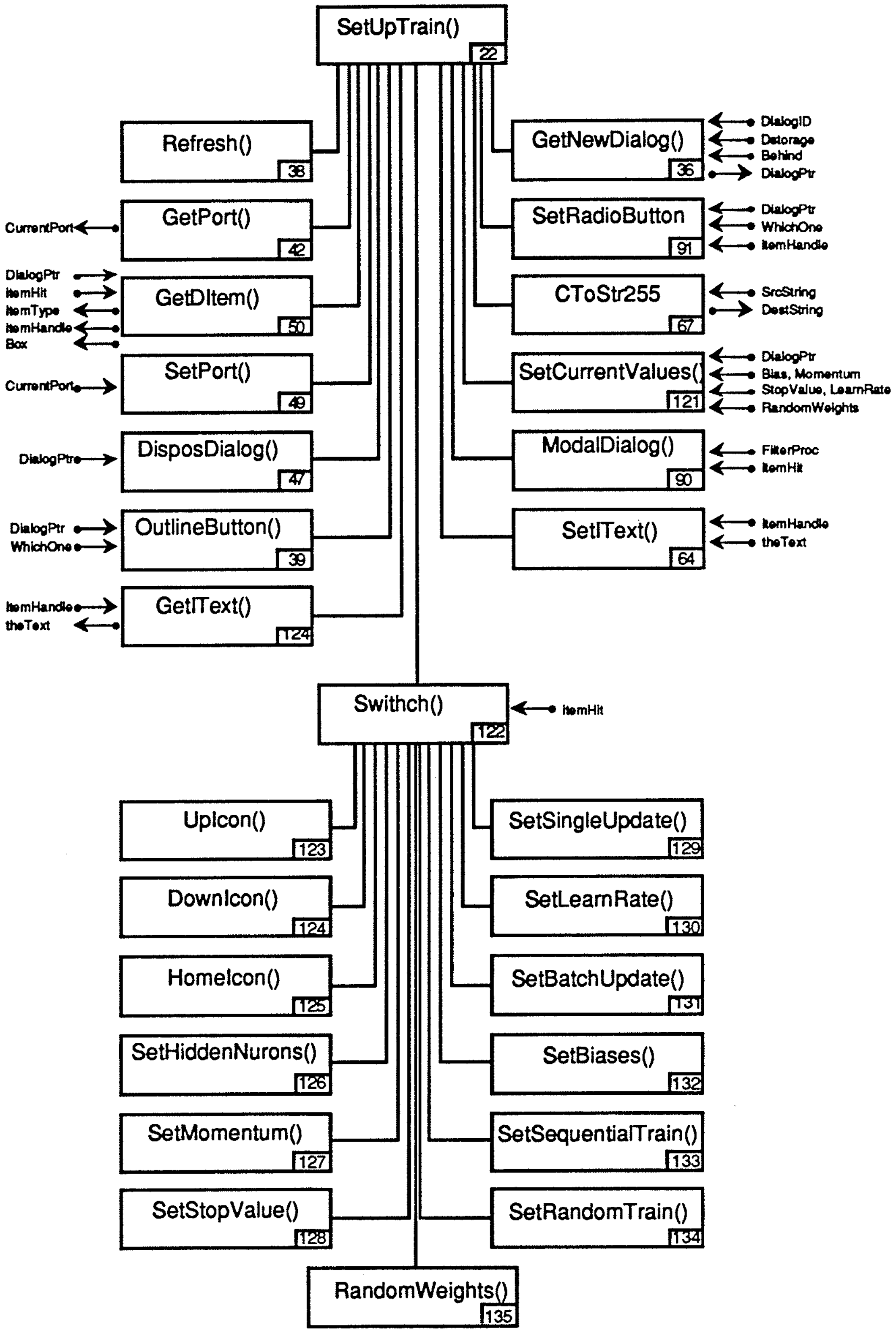


Fig. 5.17 *NETWORK SETUP* command.

menu bar. At this point, any parameters may be changed and training resumed via the *TRAIN NETWORK* command. A structure chart describing the *NETWORK UPDATE* dialog box is given in Fig. 5.18.

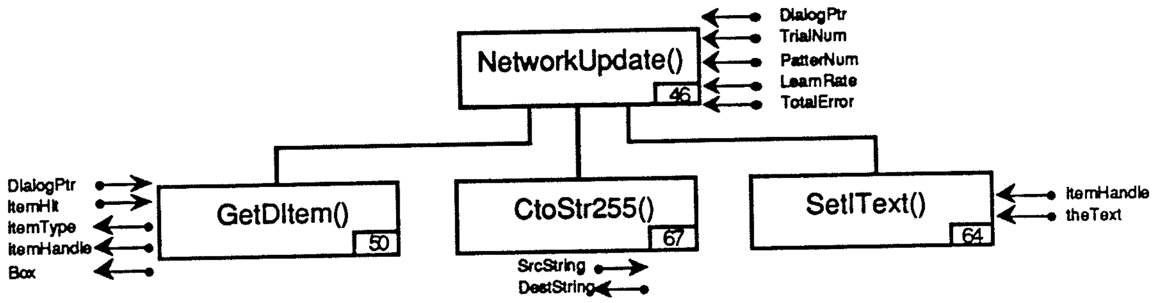


Fig. 5.18 Updating of the neural network modal dialog box during training.

Most of the routines used to implement the BP neural network are designed as methods in an object. They are very generic since they appear not only in this command sequence but also in the test and recognize command sequence. The *TRAIN NETWORK* command is implemented as a method within the document object so that any other object can easily access it. Also, to prevent this routine from crashing as a result of a person inadvertently selecting the *TRAIN NETWORK* command prior to recording data to present to the neural network a routine checks if input data exists and, if it does not exist, the system displays an *ALERT* box indicating the mistake and returns the program control to the main menu bar without incident. A snapshot of the *NETWORK SETUP* and *NETWORK UPDATE* dialog box is found in Appendix A. A structure chart of the neural network training procedure is provided in Fig. 5.19.

5.3.4 Testing

Following network training, the network should be tested with the training patterns to verify that training was indeed successful. This activity is enabled by choosing the *TEST NETWORK* command found under the *TRAIN* menu in the main menu bar.

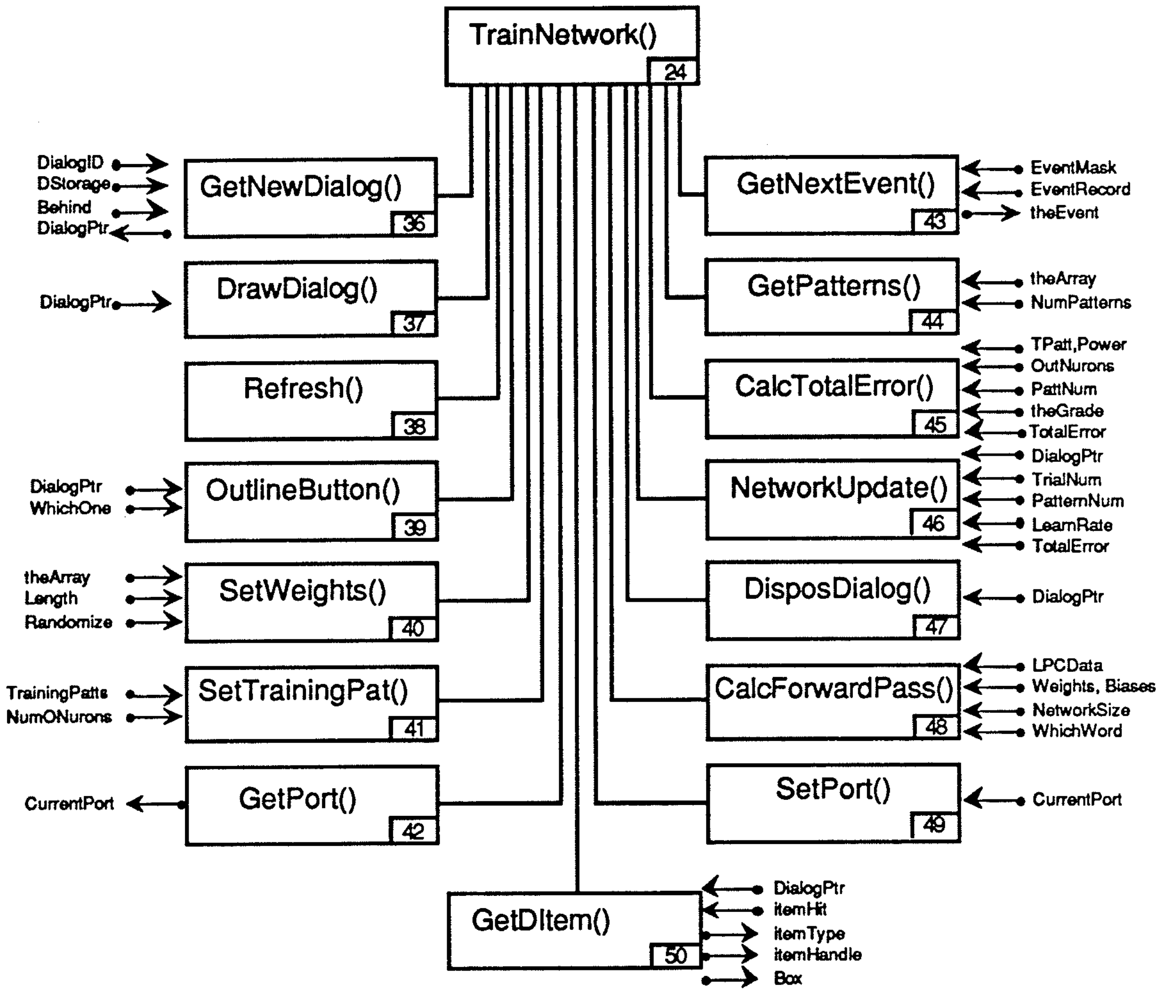


Fig. 5.19 *TRAIN* command.

This dialog box contains control buttons which are used to select training patterns. The selected training patterns are propagated through the network, the outputs are obtained and mapped to a textual equivalent of the selected output neuron. This text, which represents the utterance and version, is displayed in the identity segment of the dialog box. If the stopping criteria is not strict enough, it may be adjusted using the *NETWORK SETUP* dialog box. Training is resumed using the *TRAIN NETWORK* command. A snapshot of the *TEST NETWORK* dialog box is found in Appendix A. The *TEST NETWORK* structure chart is given in Fig. 5.20.

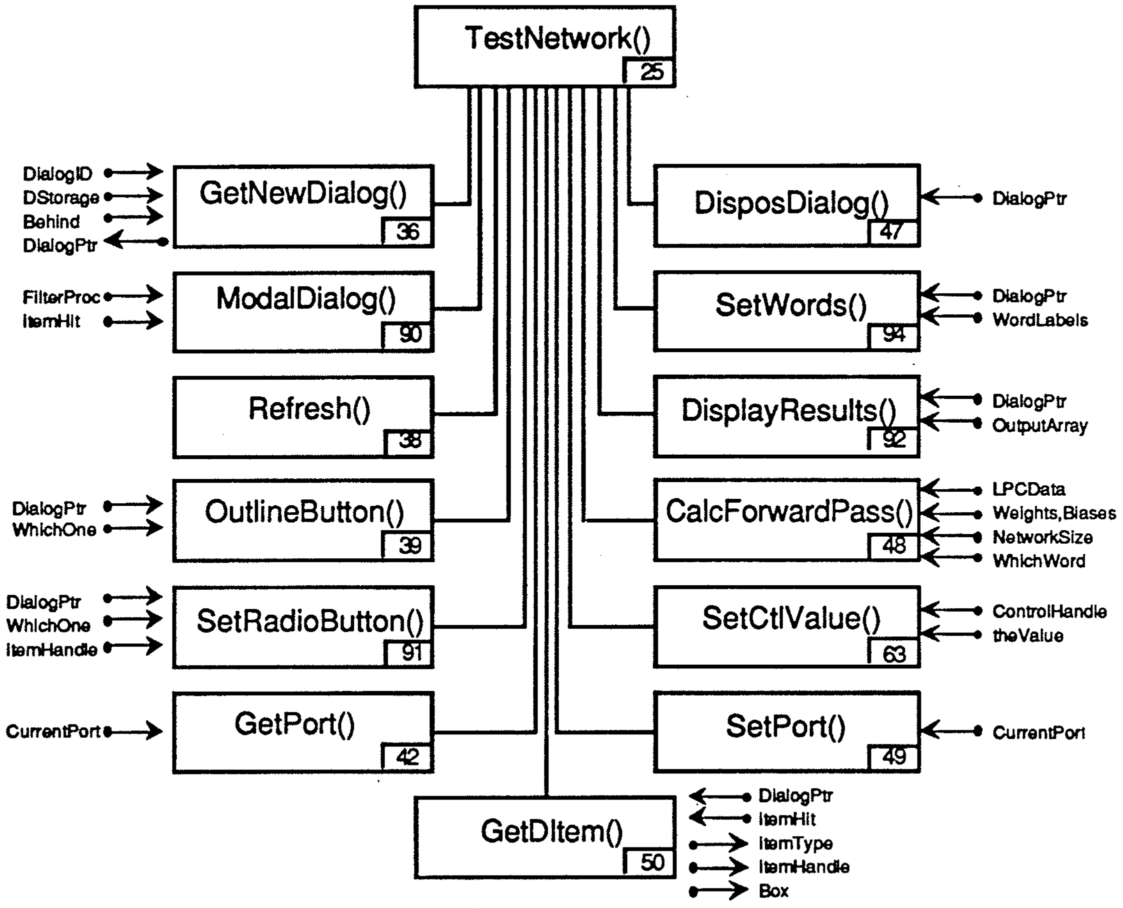


Fig. 5.20 *TEST NETWORK* command.

5.3.5 Recognition

The next major component in the software is recognition. Under the *RECOGNITION* menu is the *RECOGNIZE* command. Three sections are present in this dialog box. The top section is provided for utterance selection and the middle section is provided for version selection. This combination uniquely describes one of the prerecorded utterances which is played back to the autistic individual as input stimulus when they select the *PLAYBACK* command described by Fig. 5.21. A beta version of the playback command has been implemented on the Macintosh computer as opposed to the command routines developed for the IBM and Oki speech chip approach described in Chapter IV.

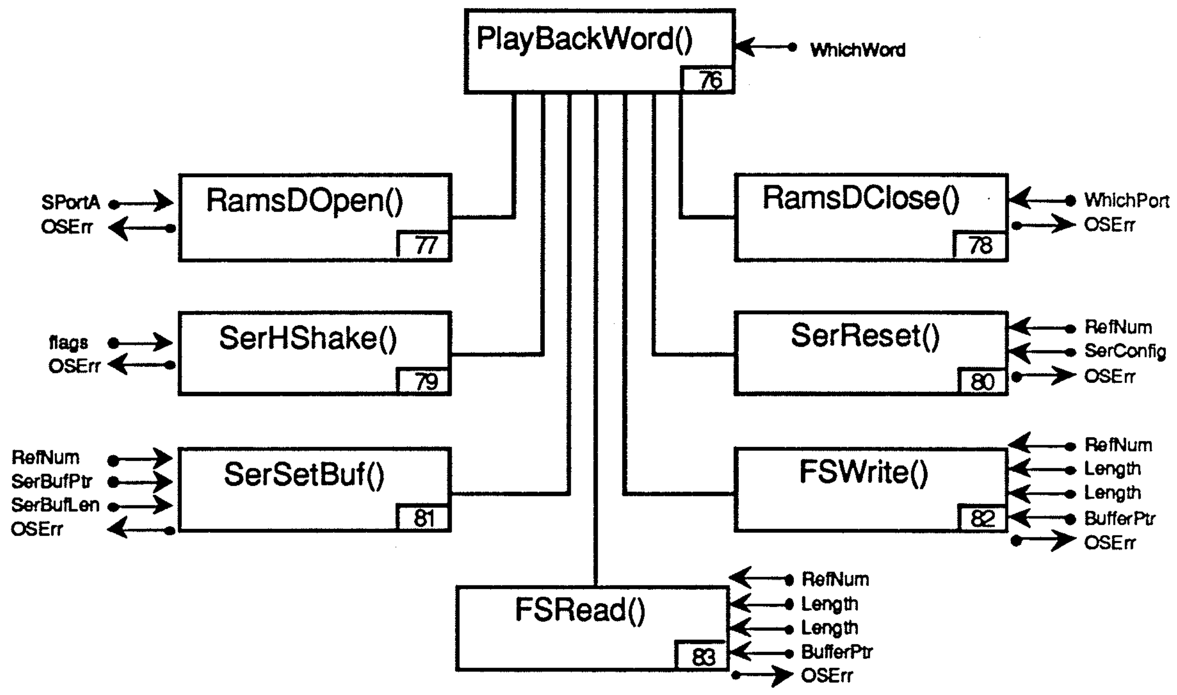


Fig. 5.21 Utterance playback.

The IBM computer would receive the requested utterance playback message, decode it, and synthesize the requested utterance for the autistic individual. This stimulus is used as the target which the individual then tries to achieve by speaking into the microphone. This is a key step in the vocal shaping process which involves providing an approximation which the autistic individual can achieve. The unknown utterance is acquired by having the person activate the *RECORD* button. The Macintosh would respond by sending a message to the IBM to record the utterance, and to obtain and return the LPC coefficients. These coefficients are placed at the inputs of the trained neural network, propagated through, and the resulting vector is collected and decoded by a recognition routine. Decoding involves choosing the largest output and mapping it to one of the utterance-version combinations contained in the data set. After decoding, the identified utterance and version is displayed in the third section of the recognize dialog box using the display results routine provided in Fig. 5.22. The playback of the selected utterance-version combination can be performed multiple times before the autistic individual attempts to replicate the

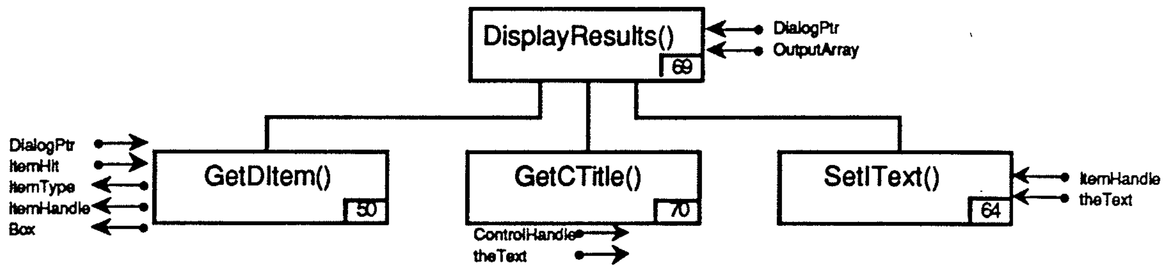


Fig. 5.22 Recognition results display.

prompted utterance. The tasks of playback and record were designed independently to allow this ability since it would seem that hearing the prompted sound a few times may give the individual a better idea of how to pronounce the prescribed sound. The selection of utterance-version combinations and recognition can be performed indefinitely with the program looping and displaying the new results every time *RECORD* is selected. Additional pieces of support software are also used. One piece involves the serial communication that is used in transferring protocol commands, LPC data, and various messages between the Macintosh and IBM computers. In this system, the IBM computer acts as a slave to the Macintosh computer, polling the communication channel for a request from the Macintosh computer. A snapshot of the recognize dialog box is found in Appendix A. The *RECOGNIZE* structure chart is given in Fig. 5.23.

5.4 Data Acquisition and Preprocessing

Speech is emitted from the human in the form of an APW. The APW hits the microphone face and vibrates the membrane located inside. These vibrations are converted into a faint electrical signal. This signal is sent through an AKAI tape deck and passed into the NEC DSP board where it is filtered using a bandpass filter with skirts about the toll quality band (300 to 3300 Hz). The filtered signal is sampled at a rate of 8 kHz using 8 bit samples. These samples are sent into a routine which obtain the LPC coefficients. This process is shown in Fig. 5.24.

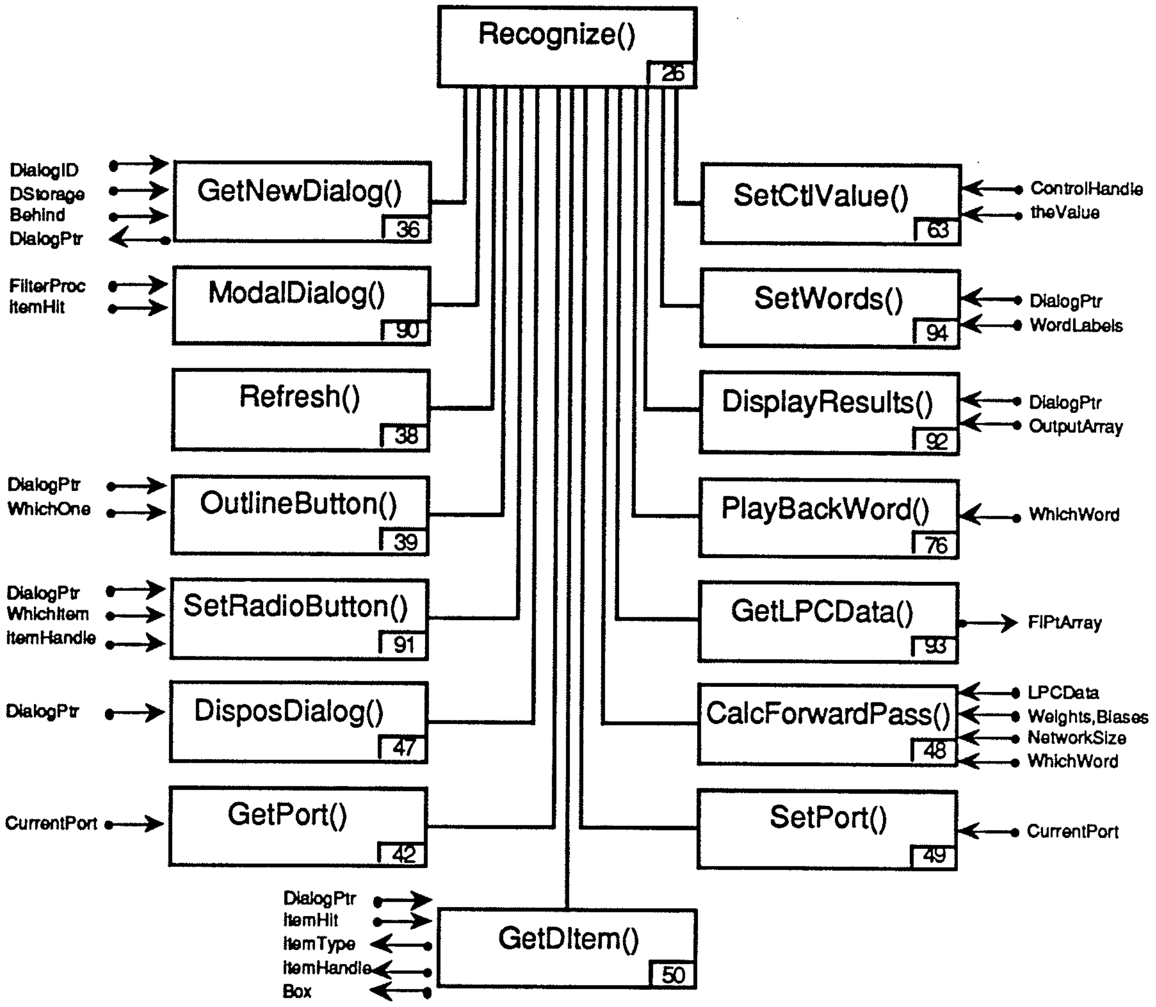


Fig. 5.23 *RECOGNIZE* command.

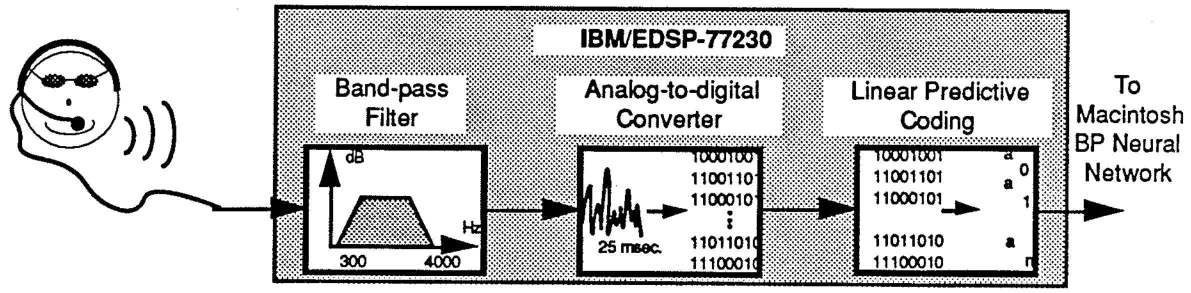


Fig. 5.24 Acquisition of the speech utterances by the IBM computer.

Choice segments, or frames, set at 25 ms are used for recognition from the one second utterance. Since the data for utterances is somewhat quasi periodic, not all the frames from the utterance are taken; only three frames are chosen as shown in Fig. 5.25. The frames are chosen this way to pass over the rise time to, and fall time from resonance, and to include the middle of the sample which is at resonance. Thus, the first frame sample is taken at 0.333 s, the second at 0.5 s, and the last at 0.667 s. Other frames adjacent to the selected ones are not taken since these coefficients have very near the same value. The reason they are approximately the same is because the vocal tract has a fixed rate of change, the length of recording is only one second, and the utterances used are quasi-periodic ones, such as vowels. Vowels are locally stationary for relatively long periods of time which make them good candidates for LPC analysis and recognition. This is evident from inspecting an amplitude plot of the vowel /i/ for instance. Every 10 ms to 15 ms a periodic waveform is observed after resonance has been achieved in the vocal tract The careful selection of frames eliminates redundant information that would otherwise decrease processing time and would not add new information. This does limit the type of utterances that the system can accurately recognize though. For utterances which are not locally stationary within the sample period of one second may cause difficulty to the system since a key frame may be missed. In situations like these, increasing the number of frames, and/or selecting different frame sample times could overcome this difficulty. In this thesis though, the utterances are select utterances which are based on vowel sounds.

The LPC coefficients, and all the other data, generated by the software are held locally in RAM during operation. The data is managed in this manner since many routines access and update various arrays and constants throughout the course of a session. Since the data is handled this way, much of the data, arrays, flags, and constants are made global to all the objects and their methods. In other cases, where these data are not global to an object, a pointer or handle to the data is passed. In this way, the location where the data is held

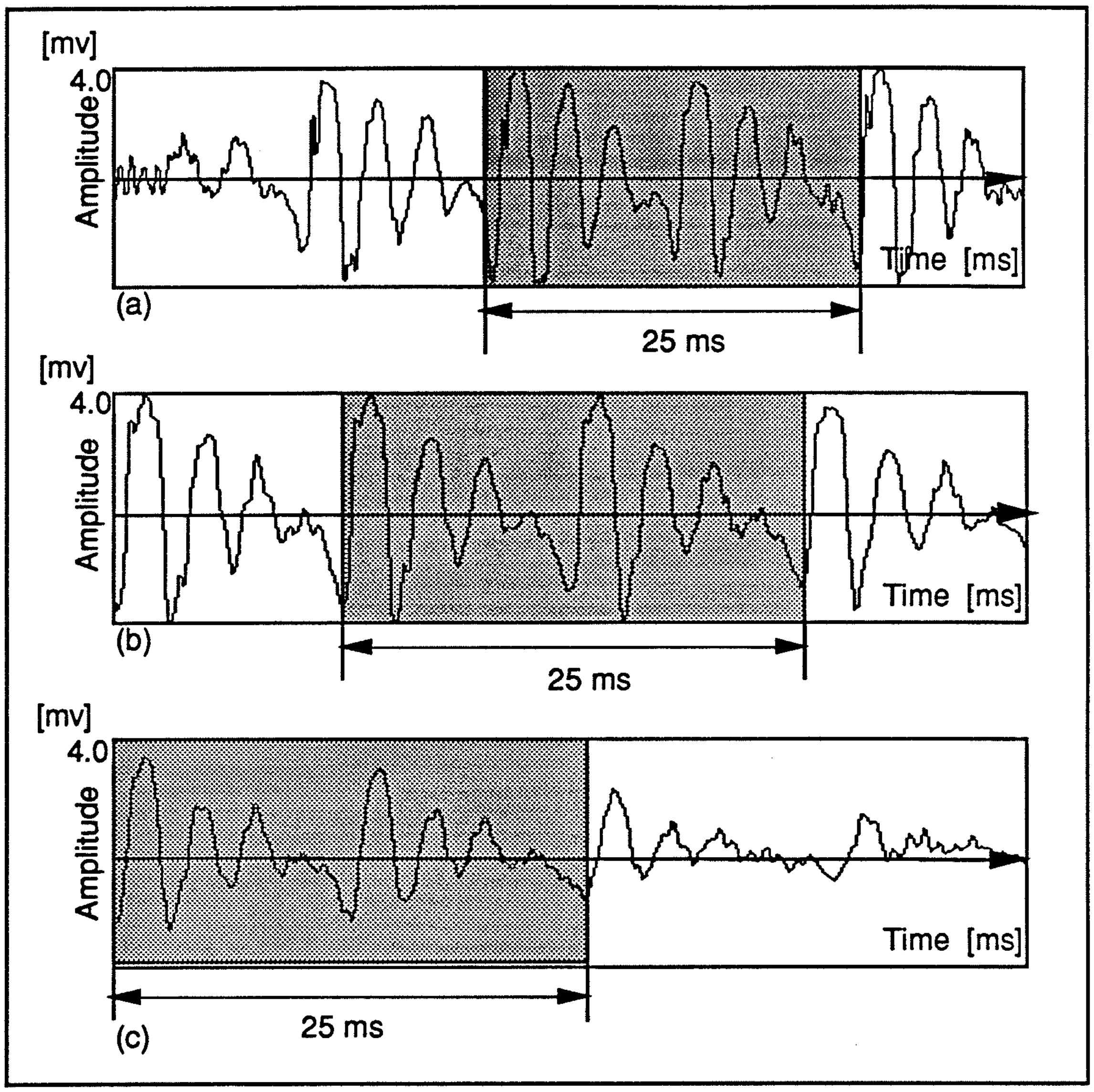


Fig. 5.25 (a) Approach to resonance of the vowel *i* (first LPC frame). (b) Sample of the resonant period of vowel *i* (second LPC frame). (c) Fall from resonance of the vowel *i* (third LPC frame).

is passed to the method or function, and transferring of data within RAM is minimized. This type of processing and handling of data is similar to *context switching*. In this process, an address where the new data are located is passed to the function which then modifies the original data, rather than a copy.

5.5 Interfacing Software

The serial communication software is essential to the success of the speech recognition system. Although this thesis is concerned only with recognition of isolated utterances, this would not be possible without speech data being acquired from the EDSP-77230 processor and coefficients obtained from the LPC software. Thus, in order to incorporate these essential system components, a communication path and protocol is established. The communication between the IBM computer and the Macintosh computer is serial. The hardware involves mapping the Macintosh's balanced modem port lines to the unbalanced lines of the IBM's RS-232 serial port as shown in Appendix B. In this thesis, the XON/XOFF protocol is used. The next step is to choose a particular protocol format. The format values are: (i) a baud rate of 9600, (ii) no parity, (iii) 8 data bits, and (iv) 1 stop bit. In addition to this, some form of software hand shaking is required. The hardware hand shaking is taken care of by the XON/XOFF characters, but the actual transmission software communication is performed using a special protocol. This is accomplished by using two protocol messages, *OK* and *GO*. Following initialization of both computers serial port with the same hardware protocol, the following set of events constitutes successful data communication. First, the IBM computer continuously sends an *OK* message to the Macintosh computer until the IBM receives an *OK* message back from the Macintosh. The IBM computer then sends the data header to the Macintosh terminal. The header contains the number of bytes to be transmitted. The Macintosh computer reads the port to obtain the data header. Using this value, it creates a sufficiently large data buffer in RAM. After completion, the Macintosh computer sends back an *OK* message. The IBM computer, upon reception of the acknowledge message, begins transmission of the data to the Macintosh computer. The Macintosh computer reads the data port the number of times specified in the data header, since each read constitutes a single byte, and transmits an *OK* message after completion. The transmitted data is in ASCII characters which is converted to floating point numbers since this is the necessary format. The data, after being converted to floating point, are held in a local array which is sent to another function. This function places the LPC data into the appropriate segment in the pattern set array. This process of insertion is shown in Fig. 5.26.

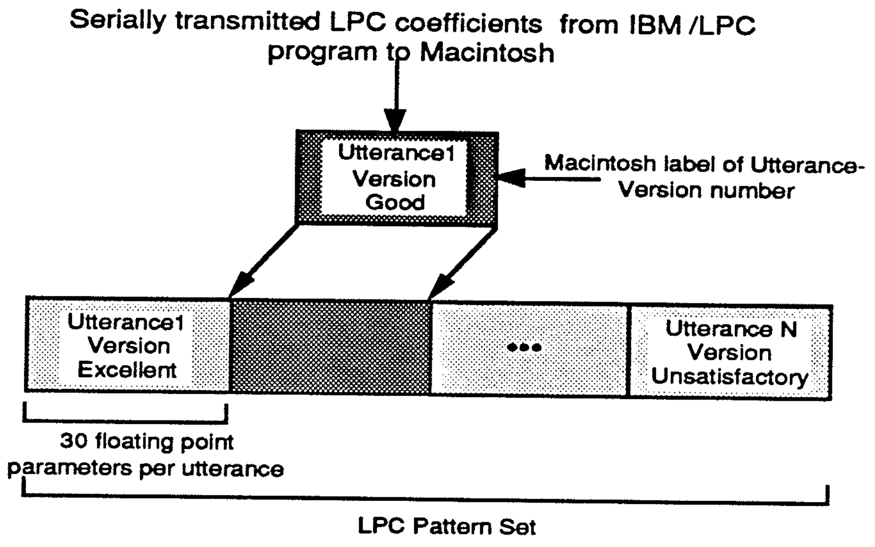

Serially transmitted LPC coefficients from IBM /LPC program to Macintosh
Utterance1 Version Good
Macintosh label of Utterance-Version number
Utterance1 Version Excellent
•••
Utterance N Version Unsatisfactory
30 floating point parameters per utterance
LPC Pattern Set

Fig. 5.26 Management of the LPC data.

The process of acquiring, converting, and storing, represent the set of events that occur during the recording of an utterance. In the case where an utterance lasts less than one second, the software buffers the remainder of the allotted space with zeros.

5.6 The BP Neural Network

The BP neural network is where the recognition occurs. In Chapter III, a detailed description BP and how it is used to perform utterance recognition was given. Here, some of the details concerning the implementation of BP into software and its association and relationships with the system are described. Basically, the creation, training, and use of the BP program occurs externally through the human interface described in Sec. 5.2.

As most of us already know, or have at least heard, one common complaint concerning BP is the lengthy training periods associated with training many problems. Although this thesis is not concerned with how long the overhead may be, it is useful to use some knowledge about the characteristics and properties of training, if possible, to decrease this time. In the paper [VMRZ88], research has shown that by using an adaptive learning strategy, learning time is reduced. In one test performed, the revised algorithm which had an adaptive learning rate converged in under 1 000 trials while the fixed learning rate did not converge for more than 30 000 trials. A similar methodology is used in this BP neural network.

The BP network is contained in the document object, central to all the data management and processing. BP involves many small two-dimensional array routines which are repeatedly employed. Since the external functions are somewhat generic, many of the two-dimensional array routines used in the BP program call the same external function, passing only their respective array pointers and local constants. In the BP method, the arrays are initialized based on the file object. If the data file object instance, *itsFile*, is *NULL* then the chain of command came from the *NEW* command, hence all the arrays and constants are read from the *NETWORK UPDATE* method variables or default method. In the case of *itsFile* being defined, the chain of command originated from the *OPEN* command therefore, all the necessary data, arrays, and constants are read from the data file using the data file instance and associated methods. The location and length of the data is known a priori because of the standard method upon which it was saved. The structure of the data file record is shown in Fig. 5.27. The situation for weights and biases is different. Here, the worst case scenario is assumed and all necessary arrays are created and zeroed according to this size. If the data is smaller than the allocated size, then the remainder of the array remains unused. The reason for allocating the memory in this manner is because

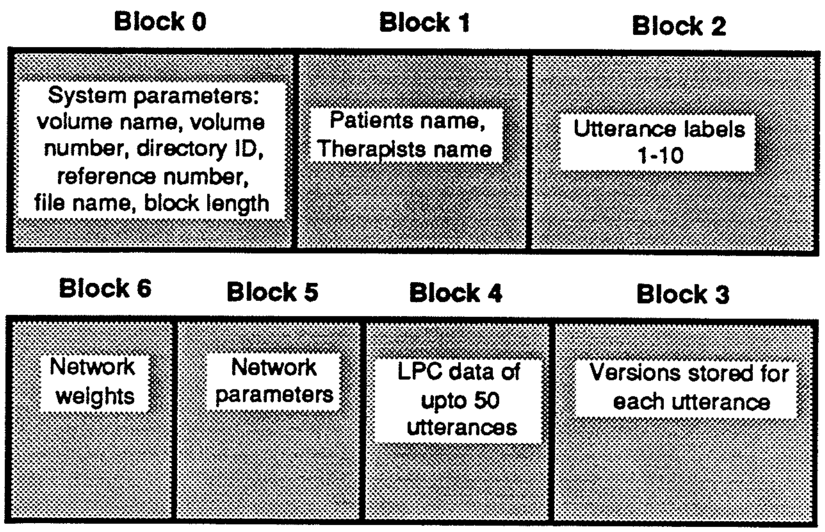

Block 0
Block 1
Block 2
System parameters: volume name, volume number, directory ID, reference number, file name, block length
Patients name, Therapists name
Utterance labels 1-10
Block 6
Block 5
Block 4
Block 3
Network weights
Network parameters
LPC data of upto 50 utterances
Versions stored for each utterance

Fig. 5.27 Data record structure.

it becomes too complex to allocate and deallocate the exact array sizes and keep track of these if the person decides on changing the network structure during the session. Also, from a memory management perspective, with reallocation occurring, heap fragmentation may occur rendering those sections of RAM useless. The reading of the weights and biases, or initialization is done using the *SetNetWts* routine found in Fig. 5.28 which is contained in the *TRAIN* menu.

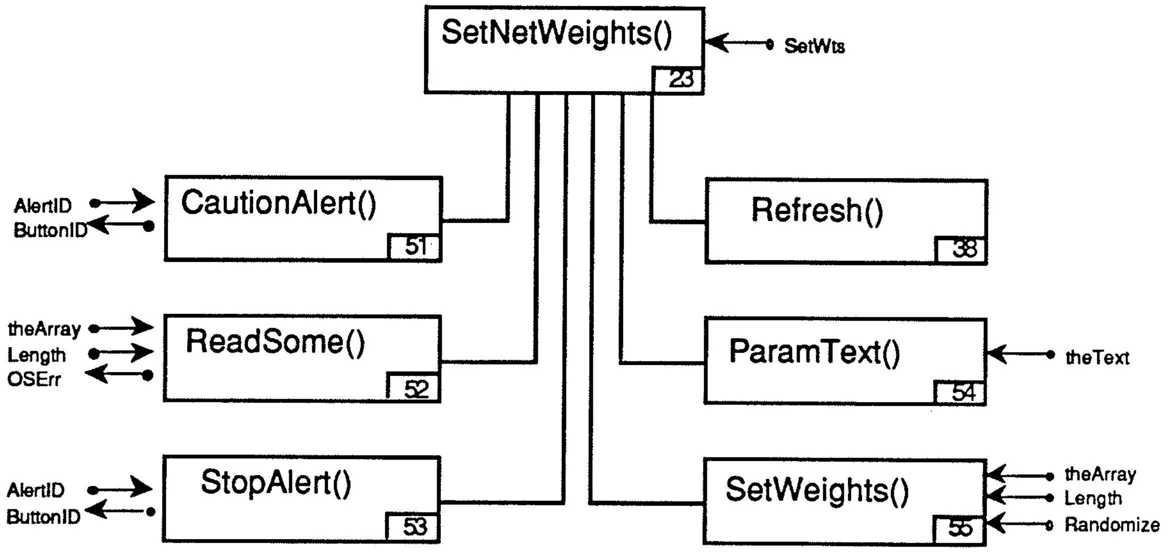


Fig. 5.28 Neural network weights.

5.7 Utility Routines

The utility routines are an important component in the systems performance. Many routines such as outlining of a button or activating a radio button are examples of the sophistication of these routines. The software uses these routines very often though. The following set of structure charts consist of the utility routines.

The check box routine provides the ability of toggling values of a check box.

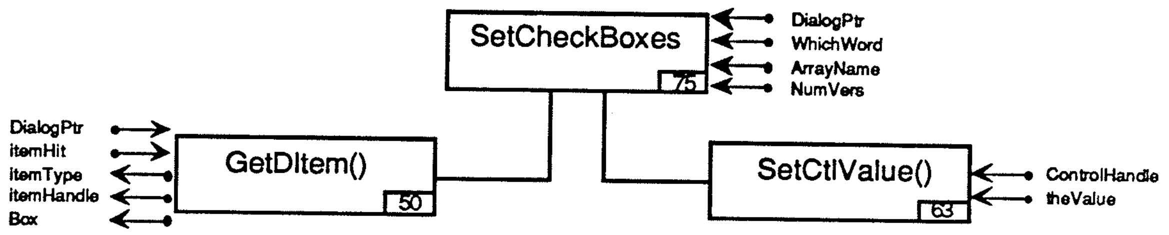


Fig. 5.29 Check boxes.

The *SetCurrent Values* function sets the current values into the *NETWORK SETUP* dialog box when the dialog box is opened. This is convenient so the person need not remember all the previous settings.

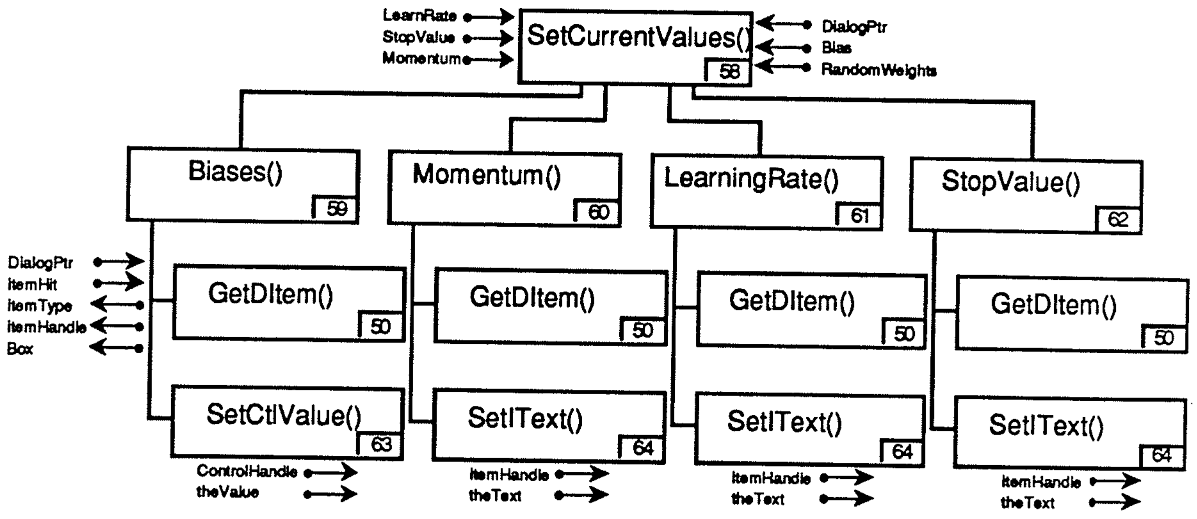


Fig. 5.30 Setting the values in the *SETUP NETWORK* dialog box.

The *SetRadioButton* function toggles the value between two radio buttons.

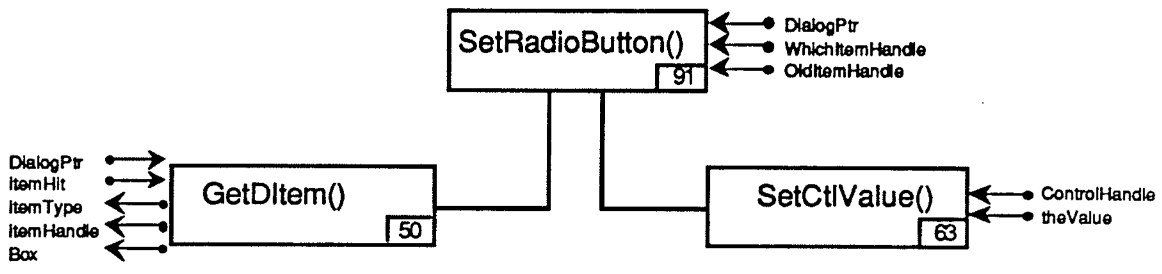


Fig. 5.31 Radio button management.

The *GetWords* function reads in all the words from either a dialog box or text file depending from where it was called. The words refer to the utterance labels entered in when the session began, following the opening of the new library dialog box.

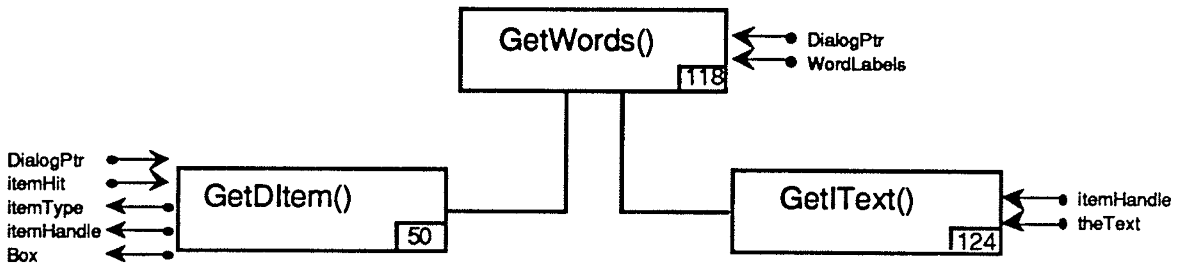


Fig. 5.32 Loading of the utterance library.

The *SetWords* function takes the pascal array and displays the word labels onto a dialog box pointed at by the dialog pointer.

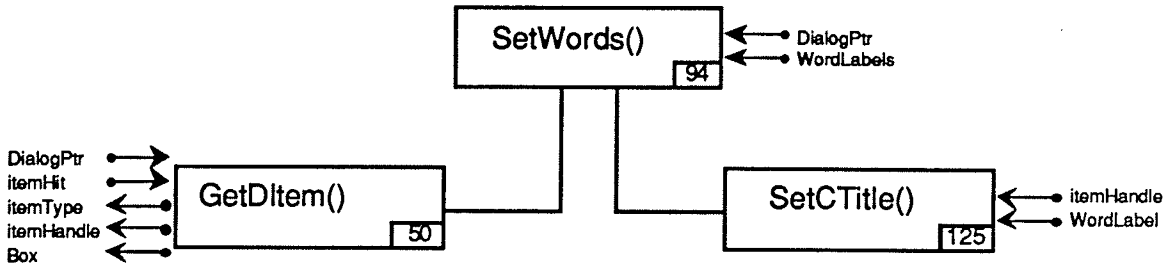


Fig. 5.33 Display of the utterance library.

The outline button routine performs the default button outlining described in the Apple standard interface guidelines given in [Appl88].

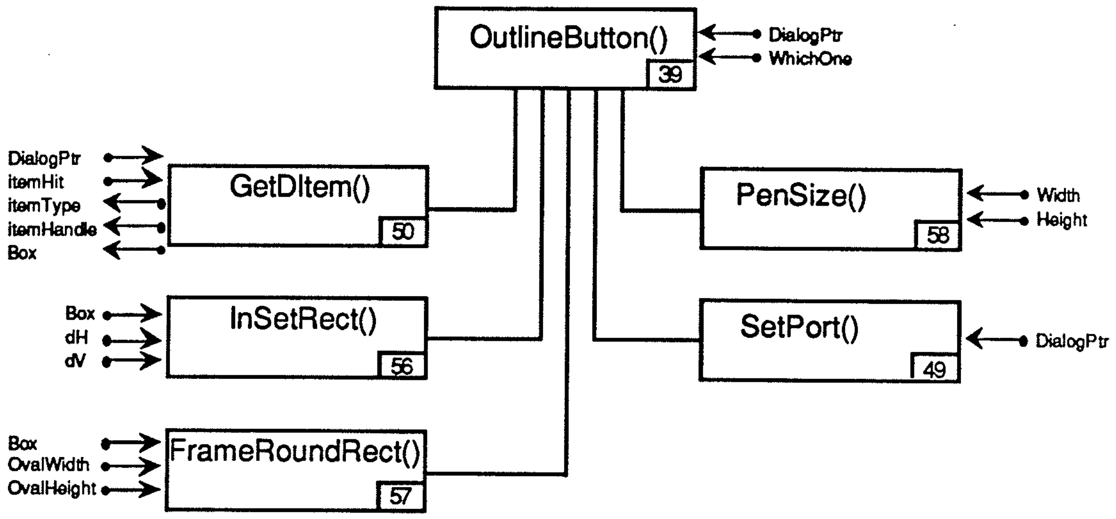


Fig. 5.34 Button outline display.

The *SetDate* function displays the current date onto the screen in a non-editable text field.

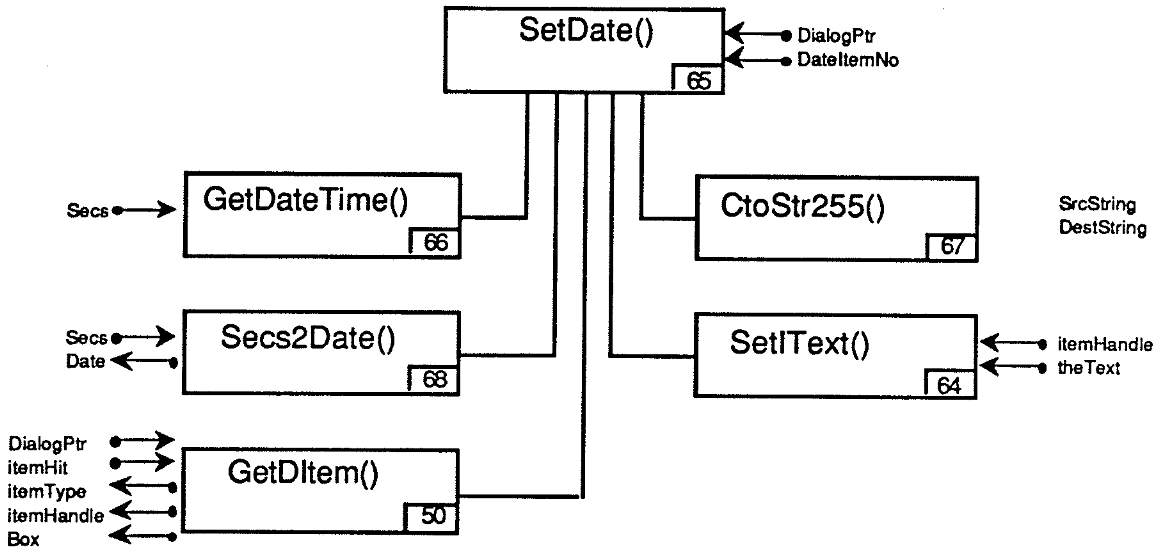


Fig. 5.35 Date display.

5.8 Summary

The software environment involving *C* object-oriented language and the relationships between objects are described. Further relations to the software are established which is followed by a detailed discussion of the operation of the software in carrying out the various functions necessary in performing isolated speech recognition. Each object is divided into its methods, and then each method is described. The various functions used in performing the background tasks are included for completeness. Since the procedures have been discussed, the software written and in place, experiments and results now follow.

CHAPTER VI
SPEECH RECOGNITION EXPERIMENTS

6.1 Introduction

Verification and experimentation of a speech recognition system involve many tests in order to sufficiently evaluate its performance. First, verification of the source code must be performed to prove its correctness. These tests have nothing to do with the performance of the system, but are a necessary foundation for the recognition experiments. The next series of software tests are created to verify the entire operation at the system level. Next, hardware verification is performed to verify that the hardware is functioning correctly. Finally, a set of experiments is conducted using speech data. The purpose of these experiments is to evaluate the speech recognition systems performance based on multi-level grading and strict classification of a difficult data set.

6.2 System Verification

System verification is designed to demonstrate correct operation of the entire system rather than its performance. In software verification, tests are used to show correct program logistics. This entails display agreement, mathematical agreement, and functionality correctness. Hardware verification involves testing of the hardware in processing and communicating the speech signal to and from system components to show its correctness.

6.2.1 Software Verification

These tests are designed to detect logic errors and to demonstrate correct system functionality. Calculation agreement of hand generated results are verified against computer generated results. This may solve problems where floating point data, casted into integer, resulted in a null value or when the size of a variable, such as a long data type is required and only short has been assigned. Attempts are made to use the software in a manner that it was not intended. This type of action is used to discover sequences which may lead to an incorrect solution. It may be impossible to completely *debug* the software for all possible combinations. Therefore, only the most commonly used routines are extensively tested. There are no benchmarks that can be applied to the software to eradicate all possibilities of error since software objectives are different. To reduce errors later in the development cycle, software verification testing was performed at every stage. To further reduce problems, objects, methods, and functions were developed in isolation. They were verified, and then integrated with the existing software. This stage of software development is sometimes referred to as the *Alpha* testing. Following *Alpha* testing is *Beta* testing. Here, a set of experiments are conducted to evaluate the performance of the speech recognition system.

6.2.2 Hardware Verification

In the case of speech recognition, hardware verification amounts to verifying the filter bandwidth, frequency response of the microphone, and any noticeable distortion throughout the signal path. In the first test, the NEC board and signal path is tested to verify that the signal path does not introduce any frequency or amplitude distortion. In the latter case, this reduces to verifying that the A/D and D/A work correctly. Also, the bandwidth of the NEC DSP is verified against what is provided in the manual. The first test involved generating white noise using the spectrum analyzer external oscillator, which contains frequencies up to 50 kHz, and connecting it to the NEC DSP input. The spectrum analyzer was set to 0.5 kHz/division and 10 dB log scale. The DSP ran a test routine which performed A/D and then D/A on the waveform, and finally passed the analog signal to the NEC output port. The output was directed into the input port of the spectrum analyzer. The block diagram describing the configuration is given in Fig. 6.1 and the resulting frequency response is given in Fig. 6.2. The frequency response that was captured on a black and white photograph off of the spectrum analyzer screen, was later scanned using Apple Scanner, and then imported into MacDraw II.

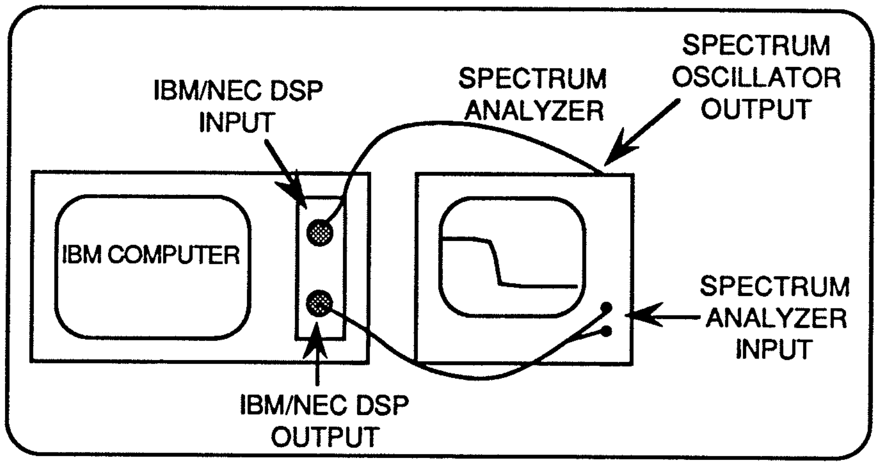


Fig. 6.1 Hardware verification of the NEC DSP A/D and D/A conversion process and filter bandwidth.

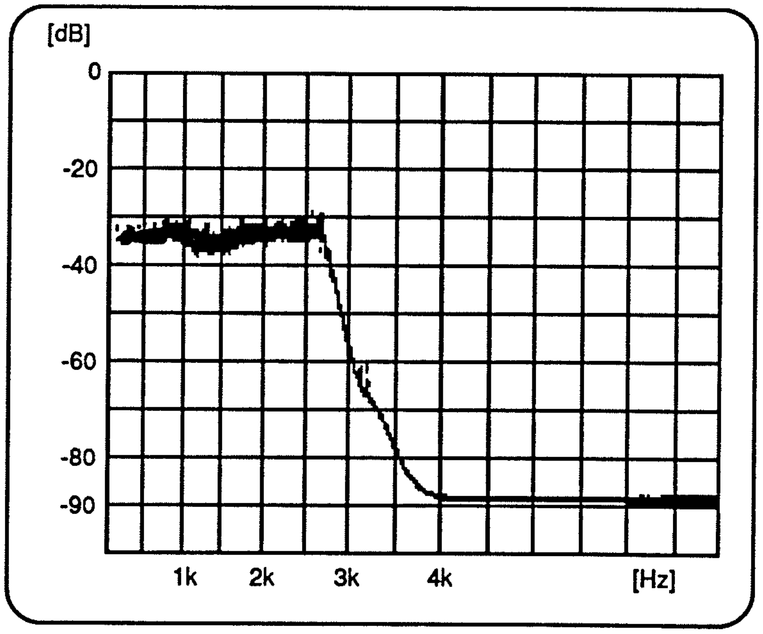


Fig. 6.2 Frequency response of the NEC DSP using the spectrum analyzer. (Scanned photograph using Apple Scanner and imported into MacDraw II).

The drawing was labelled and given a grid since the grid on the spectrum analyzer was not illuminated nor resolved by the photograph. The frequency bandwidth was determined to be approximately 300 Hz to 3.3 kHz. Note that although the spectrum analyzer did not resolve the lower cutoff of 300 Hz, it was checked on a oscilloscope for both scanned frequency responses.

The second test is designed to show that significant distortion is not present in the signal path when including the microphone. The spectrum analyzer was used to generate frequencies less than 50 kHz. This signal was passed to the input of a speaker which was captured by the microphone. The microphone was connected to the AKAI tape deck input. The output of the tape deck was connected to the input of the NEC board. The NEC board ran the same A/D and D/A routine used in the former test. The analog output was connected to the input of the spectrum analyzer. Again, no significant distortion in the frequency response was observed. A similar bandwidth was calculated from the resulting waveform. The frequency response diagram was prepared in the identical manner to the previous test. The test equipment configuration is shown in Fig. 6.3 and the resulting frequency response is provided in Fig. 6.4.

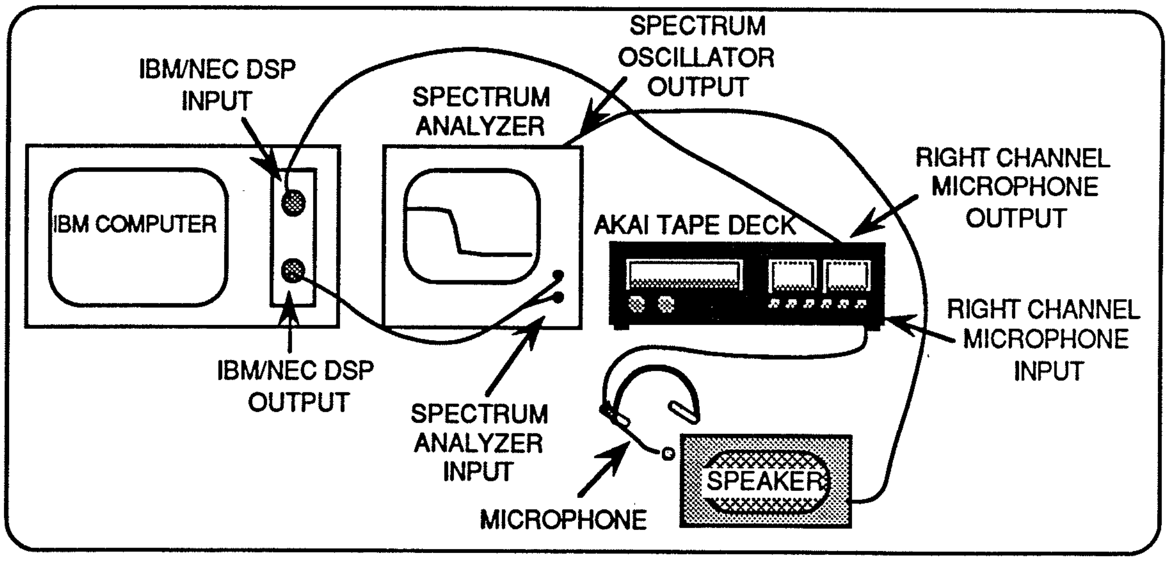


Fig. 6.3 Hardware verification test of microphone distortion in the signal path.

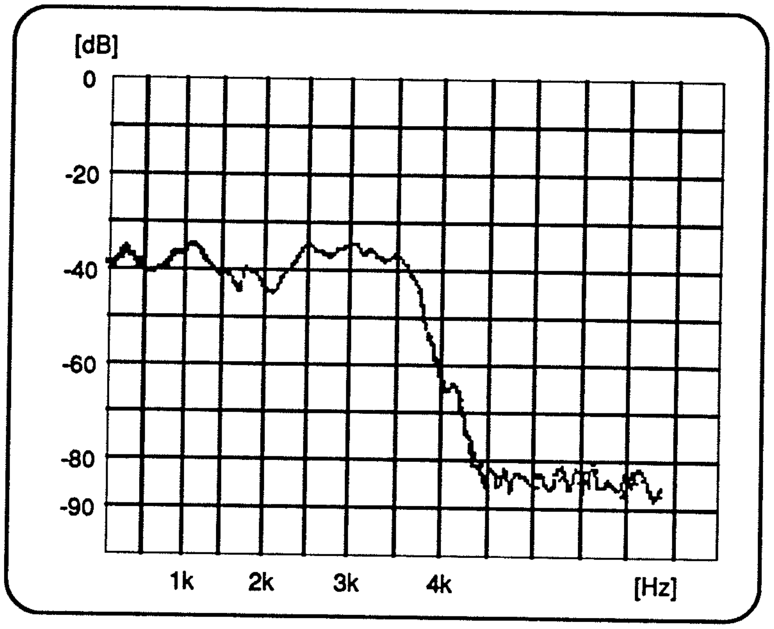


Fig. 6.4 Frequency response of the NEC DSP after passing through the microphone using the spectrum analyzer. (Scanned photograph using Apple Scanner and imported into MacDraw II).

The last issue to address concerning hardware is the real-time factor. Two computers are involved in the speech recognition system. The IBM AT portable uses an 8 MHz clock to run the LPC routine. The time taken to recognize an utterance is on average 4.5 s. This time is very dependent on the IBM machine used. If for instance, an 80386 machine running at 16 MHz was used, it would effectively halve the processing time, and adding a floating point card to perform the multiplications would further reduce this time making the system effectively run real-time. The Macintosh on the other hand, once having received the data, *does* interpret and display the results in real-time. Since the IBM necessary hardware is not available, the system does not run real-time in its present format but, this is strictly a hardware issue.

6.3 Descriptions and Results of Experiments

The purpose of the recognition experiments is to evaluate the systems multi-level classification ability and strict recognition accuracy. Before the tests are conducted, a description of the training sets is given.

6.3.1 Data Acquisition and Training

The acquisition process to be described is common for all speech tests performed. The tests are conducted in a small office with many objects. Noise of people talking in the background, construction outside the window, and noise from other people in the room was present. A total of 11 people were randomly chosen from the faculty. The random sample consisted of ten males and one female. All individuals were told how to use the system and what was to be done. Each person generated a data set which was acquired using the speech recognition software. The data set consisted of the selected utterances using two versions, *excellent* and *good*. The sound was first generated to provide an example sound so the operator could correctly adjust the recording level on the AKAI tape deck input. The sound was then acquired using the *ACQUIRE* dialog box described in Chapter V. The neural network was then trained using the batch update technique following the acquisition of data. All networks achieved a global minimum of 2%. The consonant networks used four hidden neurons and the vowel networks used five hidden neurons. A further description of the training procedure will be presented later in this section. One to two days later, the individual returned and was tested using the trained neural network. A description of the specific hardware characteristics and methods used during the testing process now follows.

The uni-directional, noise cancelling, dynamic microphone has a bandwidth of 15 kHz (50 to 15000 Hz). The frequency response plot is shown in Fig. 6.5 and the polar pattern is provided in Fig. 6.6. The sensitivity of the microphone is -47 dB (0 dB = 1 V/μ bar, 1 kHz).

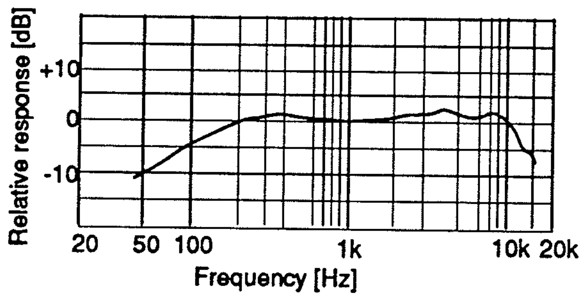


Fig. 6.5 Frequency response plot of the SM10A microphone (after [Shur91]). (Scanned using Apple Scanner and imported into MacDraw II).

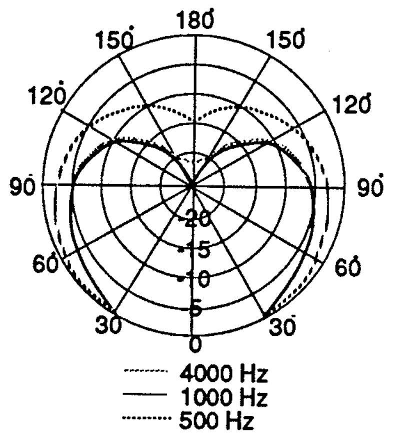


Fig. 6.6 Polar pattern of cardioid (uni-directional) microphone response (after [Shur91]). (Scanned using Apple Scanner and imported into MacDraw II).

A full technical description of the microphone is provided in Appendix B. The data is acquired using a microphone which is fixed to the person's head. This is done is because a hand held microphone will not pick up the speech signal consistently since movement by the speaker is unpredictable. "To keep the microphone a constant distance from the child's mouth for accurate computer assessment of the sound, the child wore a set of headphones with a small microphone attached for recording the child's response" [Cair90].

The specifics of the training procedure involve initially configuring the neural network for a minimum of four hidden neurons. If the network does not converge using four hidden neurons then the number is increased by one until it does converge. A network is considered to have not converged if the weights exceed their value range of 96 bits, or if the global minimum is not achieved after a few thousand trials. This second criteria is subjective, but from convergent networks, the training length was significantly less (order of thousands), and for some training tests in which the network was allowed to train indefinitely, it eventually exceeded the weight space range of 96 bits. "However, when no solution can be found, the rate of decrease slows down drastically before the network error has reached the desired value" [Rein90]. The number of hidden neurons is not an issue in this thesis, except from the perspective of real-time recognition. The classification time is reduced when fewer hidden neurons are used since fewer calculation between layers are performed. Also, if a network uses too many hidden neurons, it will *memorize* the data set. This does not lead to generalization since the dimension of the weight space is greater than what is minimally required for correct representation. Memorizing not only learns the underlying features but also weak features which may be considered as noise. "If the number is too large, many different solutions will exist, most of which will not result in the ability to generalize correctly for new input data, and the network will usually fail in the operation stage" [Rein90]. "All experiments show an increasing recognition rate with the number of hidden units up to...the critical number. Above this point, no further improvement occurs and even a slight decrease in accuracy can be observed" [Burr88].

A *momentum* of 0.3 and an initial *learning rate* of 0.5 is also used to begin training. The network update is set to *single* and the *random presentation* format is used to present the patterns. Biases on the hidden and output layer are made active and *random weight* initialization, using a dynamic range of ± 0.5, is used to initially seed the network weights. The weights are actually created by using the *CREATE NEW WEIGHTS* command which is found under the *TRAIN MENU*. The network *stopping criteria* is set at 2% and the network is trained until the total internal error fell below this threshold. The adaptive learning rule described in Chapter III is used with a learning window from 0.1 to 2.5. Following training, the network is verified.

6.3.2 Network Verification Test

Testing of the network involved opening the *TEST* dialog box and selecting every combination of the pattern set. Results of this test were that every tested pattern yielded the correct utterance identity. To test a pattern using the trained weights and biases, a training set pattern is presented to the inputs of the network and propagated forward using the BP algorithm described in Chapter III. The outputs are thresholded and the largest output neuron is chosen and mapped to an utterance-version combination which is displayed in the *TEST* dialog box identity segment.

6.3.3 Recognition Tests

Following verification network testing, recognition tests are performed to evaluate the multi-level classification accuracy and strict classification accuracy. The first set of tests includes an special vowel set. Since vowels are locally stationary, their spectrum is easily evaluated by LPC. The next set of tests are used to evaluate consonant combinations with vowels. The basis of all the tests is directly or indirectly related to the spectrum obtained from vowels. The difficulty of the data set is shown using the following two figures. Figure 6.7 describes the vowel triangle which relates the first formant, F1, to the second formant, F2. It will become more apparent after inspecting Fig. 6.8 that the vowels and their loci overlap. In particular, this figure is used to select distinct groups which vary in difficulty based on their relative locations to one another. For instance, the vowel sounds /OO/, /IY/, and /A/ provide the greatest spectral separation, while the vowel sounds /I/, /E/, and /AE/ provide less separation based on the first two formants. From the tests then, it

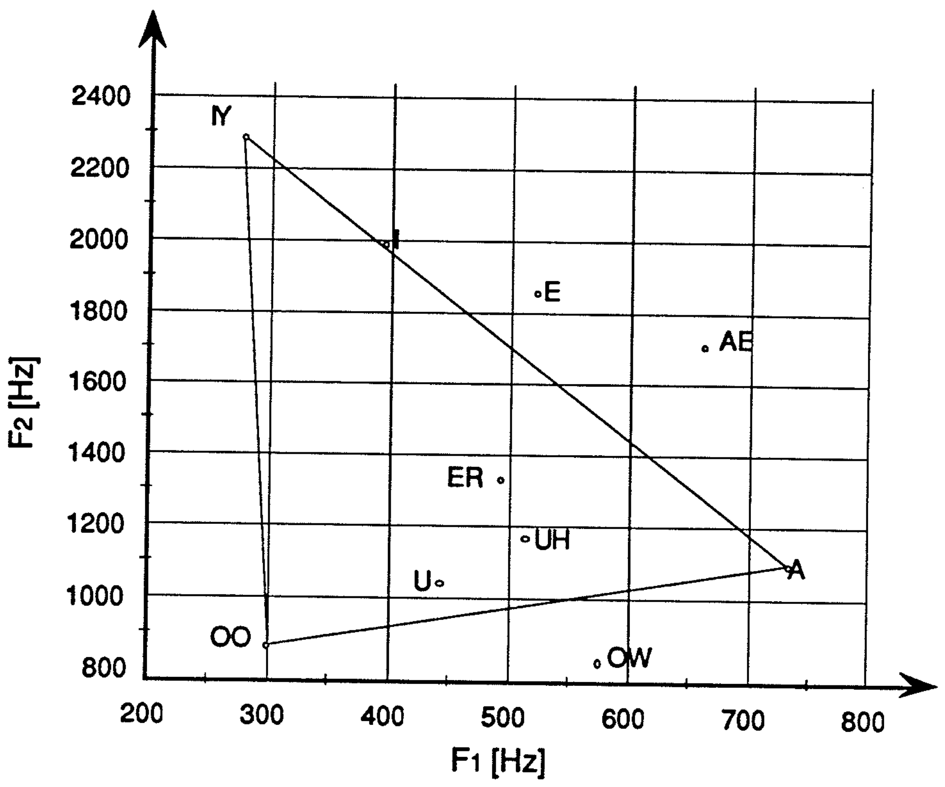


Fig. 6.7 The vowel triangle (after [RaSc78]).

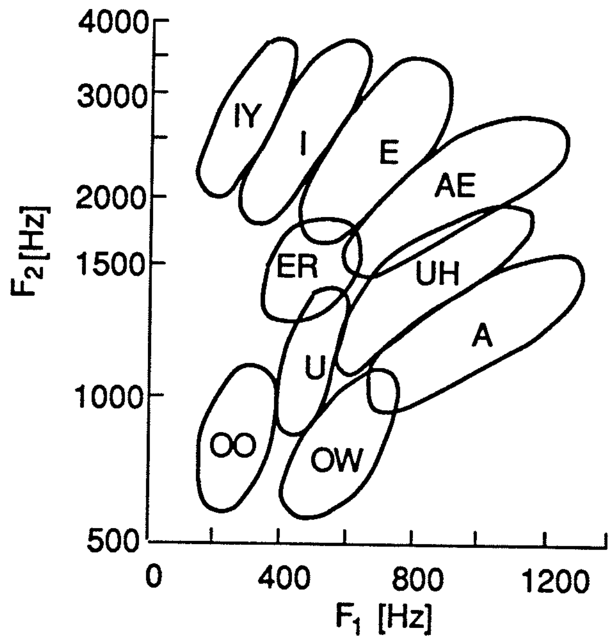


Fig. 6.8 Loci of vowels for a wide range of speakers (after [Pars86]).

would be expected that the first group will provide better results than the latter group. The first group will be used to test the multi-level classification scheme, while the latter group will be used to test the strict classification ability of the recognizer using a difficult data set. Testing involves repeating each utterance ten times and recording the identity, and quality if necessary, from the recognizer. The speaker is the same person that generated the training set. The results are compiled and entered into a modified confusion matrix for each test conducted.

One other issue related to the recognition process is the feasibility of vocal shaping using this system. This system has been designed in a fashion to be used as a vocal shaping system, and as a vocal shaping system, it necessarily requires playback of the utterances for stimuli to the autistic individuals. Since this system is intended to be used in this fashion, and to be contained in the future on the Macintosh, much effort has gone into developing a operational beta version of *PLAYBACK* command on the Macintosh for the various utterances as described in Chapter IV.

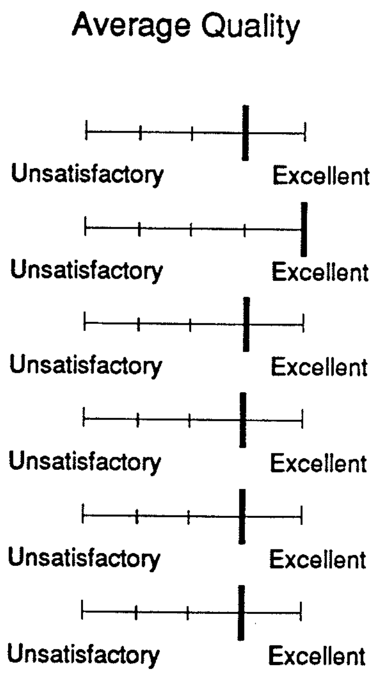
6.3.3.1 Vowel Tests

In this experiment, the system is trained with a limited set of vowels by an individual with concise pronunciation. Next, the system is expected to take this data set, which it has mapped into a neural networks weight space and recognize speech from an individual who has difficulty in pronouncing an utterance from the trained data set. It not only must recognize which utterance was spoken, but must also indicate how well it was spoken by way of a quality indicator as described in Chapter III. In this test, six vowel sounds and their two versions are trained. The data recorded in this test are the two vowel groups described in Sec. 6.3.3 which comprise: (i) /IY/, (ii) /OO/, (iii) /A/, (iv) /I/, (v) /E/ and, (vi) /AE/.

The data was collected as described in Sec 6.3.1 . Following testing, the results were compiled into a modified confusion matrix. This confusion matrix is used to describe the extent of recognition and confusion between utterances within the data set. Table 6.1 describes the results obtained from the vowel tests using 11 speakers. Each row indicates the percentage of times that each utterance was identified by the speech recognition software. The quality line indicator located at the right hand side of each row is used to describe the average quality measured for the vowel sound sent in that row. The quality ranges from *unsatisfactory* through to *excellent*. The bold tick indicates what the average quality is for the target utterance of that row. The first test, used to show multi-level classification indicates that from the confusion matrix given in Table 6.1 that the three vowel sounds, /IY/, /A/, and /OO/, provided an average recognition accuracy of 61.8% with an average quality classification of *good*.

Table 6.1 Confusion Matrix representing Recognition Results from 11 Speakers Evaluated on Six Vowel Sounds.

		Vowel Received					
		IY	OO	A	I	E	AE
Vowel Sent	IY	59.1	0.9	1.8	24.5	10.9	2.7
	OO	3.6	68.0	10.0	2.7	9.1	6.4
	A	1.8	3.6	58.2	3.6	19.1	11.8
	I	12.7	0.0	10.9	41.8	23.6	10.9
	E	8.2	0.0	4.5	12.7	60.0	14.5
	AE	3.6	0.9	9.1	10.9	25.4	50.0



The results of the second test, used to show the class separation ability were based on a difficult data set involving the vowels sounds /I/, /E/, and /AE/. From this data set (Fig. 6.8), the speech recognition system was able to separate the close vowel sounds with an accuracy of 50.8%. In particular, the vowel sound /I/ was confused with the other vowel sounds in close proximity being, /E/, and /IY/. This form of misclassification is clearly evident with other vowel sounds within the closely related group. Also, the misclassifications drop off as the spectral distance to neighboring vowel sounds become further from the target. The vowel sounds /IY/ and /A/ also appeared to be a factor in classification when considering the two vowel sounds /I/ and /AE/ respectively. Although the vowel triangle (Fig. 6.7) shows distinct separations, Fig. 6.8 indicates that these specific points are simply a reference point for the loci of the sound and that variance does exist. Also, a speakers personal vocal characteristics may differ, even from this graph, explaining stronger discrepancies between related vowel sounds. At least though, these discrepancies support the findings provided in Fig. 6.8 [Pars86]. Thus, the first test yielded 11% higher recognition results than the second test. This result is reasonable since the second groups relative formant locations were closer to one another.

6.3.3.2 Consonant Tests

A second test investigating consonant-vowel combinations is performed. This test is based on the second formant transition in the utterance. By this, the consonant-vowel combinations are distinguished based on the consonant that precedes the vowel. The consonants, /b/, /d/, /g/ (voiced stop), and /m/ (nasal) are used in this experiment to evaluate the performance of strict classification of consonants. The spectrograph of each sound is characterized mainly by the consonant than the vowel, "The formant transition for each point of articulation was characterized by a target frequency or *locus* largely independent of the vowel" as shown in Fig. 6.9 [Pars86]. The first formant trajectory, F1, appears largely similar in nature while the second format trajectory, F2, is strongly affected by the preceding consonant. In this manner, a study of consonant recognition can be done since LPC reflects vowel spectrums well. Since each consonant distorts the vowel spectrum differently, the recognition, although mainly based on the vowel, is reflected by the preceding consonant.

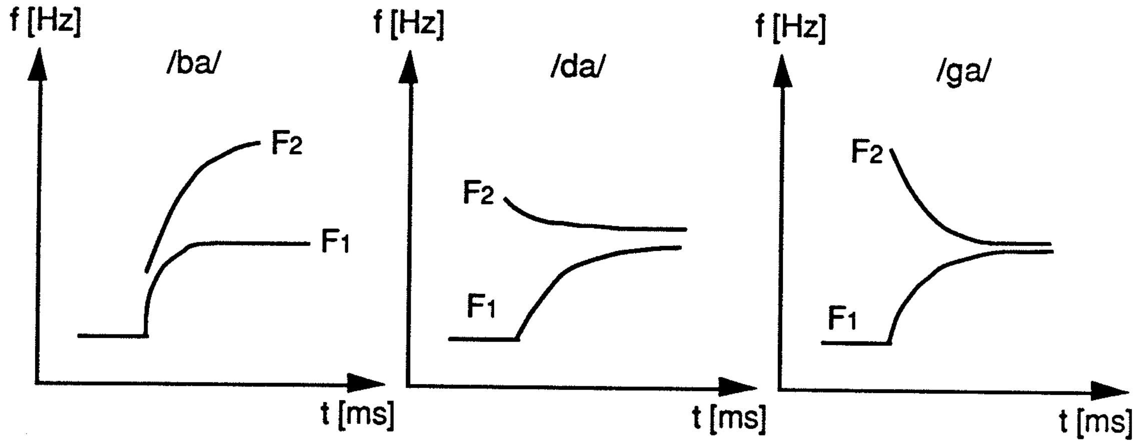


Fig. 6.9 Pattern playback diagrams showing effect of F2 transitions perceived on consonant type (after [Pars86]).

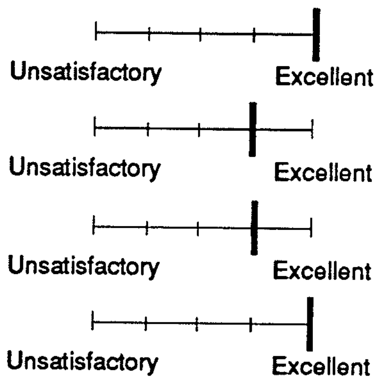
However, the vowel following the consonant is affected much more, "because of formant transitions, more detailed articulatory information is to be found in the adjacent vowels than in the consonants themselves" [Pars86]. Based on this information, two sets of tests are presented to determine the classification ability with the above mentioned set of consonants using the vowels /a/ and /e/. These vowels are used since it was determined that they provide the highest computer-human assessment agreement [Cair90].

The results of the two consonant tests, obtained from the 11 speakers involving the vowels /a/ and /e/, are shown using the two modified confusion matrices found in Tables 6.2 and 6.3, respectively. The consonants appear to have a lower classification accuracy than do the results achieved from the vowel data set. The difference may be attributable to the fact that the vowels produce easily sustainable resonant frequencies, while the consonants distort the vowel formants. The vowel distortion may not be as consistently reproducible throughout the trials, both training and testing, thus providing more differences leading to class overlaps during the network training process. This would result in lower generalization, or class separability. The vowels on the other hand, are easily sustainable during resonance which may be one of the reasons that the consistency in both training and testing led to better class separations, thus yielding both higher strict and multi-level classification results.

Table 6.2 Confusion Matrix representing Recognition Results from 11 Speakers Evaluated on Four Consonant Combination using the Vowel /a/.

		Consonant Received			
		BA	GA	MA	DA
Consonant Sent	BA	57.0	11.0	12.0	20.0
	GA	22.0	33.0	14.0	31.0
	MA	18.0	17.0	43.0	22.0
	DA	20.0	14.0	13.0	53.0

Average Quality



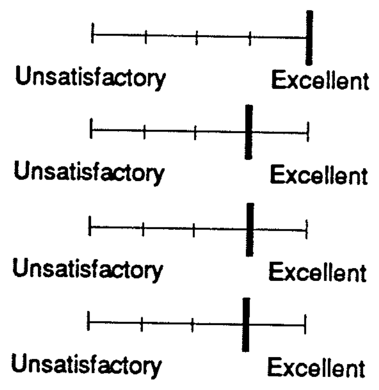
An average recognition of 46.5% was achieved using the vowel /a/. /BA/ and /DA/ achieved approximately 10% higher overall classification over the other two consonants due to their unique F2 formant trajectories. In contrast, the vowel set achieved on average 10% better recognition than did this consonant set.

Similar results were obtained from the vowel /e/. Overall though, the consonant /e/ only provided about 2% higher classification results than the vowel /a/ which implies that for this data and system the two consonant sets resulted in very near the same overall results. Also, the distribution of misclassifications appears to be fairly well spread over the other consonants about the desired target utterance, indicating that no specific consonant was strongly confused with one another, unlike that of the vowels.

Table 6.3 Confusion Matrix representing Recognition Results from 11 Speakers Evaluated on Four Consonant Combination using the Vowel /e/.

		Consonant Received			
		BE	GE	ME	DE
Consonant Sent	BE	50.0	23.0	13.0	14.0
	GE	18.0	53.0	15.0	14.0
	ME	21.0	22.0	40.0	17.0
	DE	22.0	16.0	16.0	46.0

Average Quality



6.4 Summary

This chapter describes the tests and results conducted on the speech recognition system. These include software and hardware verification tests, and recognition tests. The verification tests were performed to demonstrate the correct operation of the system upon which the recognition testing foundation lies. The recognition tests are broken into two sections which consist of vowel sounds and consonant combinations. The vowel tests are designed to test both strict classification of the speech recognition system and the multi-level classification scheme. The two consonant tests are designed to test the strict classification ability of the recognizer. Results of the strict vowel test achieved 50.8% recognition on the difficult data set. The multi-level classification scheme worked well on the specific vocabulary applied by the speakers achieving an average classification rating of *good* and an average recognition of 61.8%. It should be noted though, that the misclassifications indicated the general tendency to choose the nearest neighbors (in a spectral sense) to the target based on the first two formants approximately 22% of the time. These results supports the findings of [Pars86] vowel loci diagram provided in Fig. 6.8. The consonant tests indicate strict classification is possible 46.5% of the time for the vowel /a/ and 48% of the time for the vowel /e/, given the specific data set used. In general though, classification results were lower than the vowel results obtained for strict classification by approximately 11%.

CHAPTER VII
CONCLUSIONS AND RECOMMENDATIONS

The work described in this thesis was motivated by the need for an isolated, limited vocabulary, automatic, real-time, speech recognition system employing an improved multi-level classification scheme. This system is intended to be used as a vocal shaping tool for autistic individuals. A study of various features and recognition methods resulted in the selection and implementation of a recognition system employing a standard BP neural network using LPC coefficients as the recognition feature. In addition, the BP neural network possessed an adaptive learning rule and modified batch update procedure to reduce the network training time. The development of the system took place on an IBM and Macintosh computers. A NEC DSP board was used with the IBM computer to acquire the raw analog speech data. The IBM program obtained the LPC coefficients and transmitted these coefficients serially to the Macintosh where the recognition occurred.

The speech recognition system was designed, assembled, and tested, to evaluate both strict classification and the improved multi-level classification scheme. Its evaluation results are very encouraging. In particular, the testing was performed using two types of experiments which were then applied to 11 randomly selected speakers. The first test involved six vowel sounds which were selected to reveal two issues. The first set of three vowel sounds possessed large formant separations between F1 and F2. This property was used to demonstrate the multi-level classification ability. The results of this test, shown using a modified confusion matrix, demonstrated that the system was able to successfully differentiate the quality of an utterance 61.8% of the time if sufficient spectral separation existed. The second group of the six vowels was also selected based on the formant separation but was chosen such that this separation was much less. Thus, this test was designed to evaluate strict classification given a difficult data set. The results of this experiment, shown again using a modified confusion matrix, demonstrated that the recognizer was able to successfully classify this difficult data set accurately (50.8%). Considering the close spectral grouping of this data set, these results are promising. From the vowel tests, the next closest neighbor to the target vowel sound was chosen approximately 22% of the time.

The next set of tests were performed using consonant combinations based on the formant transition found in the preceding vowel. The four consonants used were /b/, /d/, /g/, and /m/. Associated with the four consonants were two vowels, /a/ and /e/, which followed the consonant. Each sound was tested ten times as in the vowel tests. Both consonant sets achieved similar results within 2% with the vowel /a/ data set achieving 46.5% correct recognition while the vowel /e/ data set achieving approximately 48% correct recognition. Overall though, the vowels provided approximately 11% better classification results than did the consonants based on the strict classification criteria. This result is explained in part by the fact that the consonant lasts a short time, that it contains a broad band of frequencies, is not consistently reproduceable with any given accuracy and hence, provides variations onto the vowel. Also, better results may have resulted if more frames were taken earlier in the utterance segment since most of the transitions occurred here. However, the vowel sounds are easily sustainable and thus, the spectral characteristics are much more stable leading to higher consistency of reproduction. This observation is supported by the results obtained. From the consonant tests, it appeared that the misclassified consonants had a relatively flat distribution, i.e., the recognizer did not tend to any specific consonant group when an incorrect recognition was made. The custom designed human interface improved the overall ease of operation of the system as subjectively tested through the use by the individuals.

As demonstrated throughout the chapters, this thesis has contributed to the general and technical knowledge through the following ways:

(a) A new system has been developed which employs BP as applied to isolated speech recognition using LPC coefficients as the recognition feature.

(b) Tests demonstrating the systems performance based on strict classification and multi-level classification were presented and explained.

(c) An improved multi-level grading scheme was introduced and tested.

(d) An improved human interface was designed and implemented with the new speech recognition system.

Further research is required to either improve or extend this work, including:

(a) Development of the system using current technology and hardware exists which facilitates adaptation and promotes continuation by others to further improve this system for a larger and more difficult vocabulary.

(b) The format for an improved system consisting of only the Macintosh computer has already been partially developed and is in place for further improvements including incorporation of the LPC algorithm onto the Macintosh for obtaining the coefficients and of the playback feature using the new sound manger as one of the necessary requirements for vocal shaping.

(c) The multi-level grading scheme may be improved by further reducing the number of categories from five to four thus, forcing the individual to choose about the halfway point.

REFERENCES

[AIFF89] Apple Computer, Inc., "Audio Interchange File Format: AIFF." *Technical Report*. Cupertino, CA: Apple Computer Inc., 1989, 22 pp.

[Appl88] Apple Computer, Inc., *Inside Macintosh*. Cupertino, CA: Addison-Wesley, vol. 1-5, 1988.

[Appl90] Apple Computer, Inc., *Inside Macintosh*. Cupertino, CA: Addison-Wesley, vol. 6, 1991.

[Burr87] D. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans., on Acoust., Speech, and Signal Proc.*, vol ASSP-36, no. 7, pp. 1162-1168, 1988.

[Cair90] S. Cairns, "Computer aided speech shaping," *M.A. Thesis*. University of Manitoba; Canada, April 1990, 77 pp.

[Card90] H. Card, Artificial Neural Networks. *Lecture Notes*, University of Manitoba; Canada, 1989.

[ChFa88] M. Chang and F. Fallside, "Implementation of neural networks for speech recognition on a transputer array," *Technical Report*, University of Cambridge; USA, March 1988, 13 pp.

[DeBo90] Z. Deiri and N. Botros, "LPC-Based neural network for automatic speech recognition," *Proc. IEEE Engineering in Medicine and Biology Soc.*, IEEE Cat. no. 90 CH2936-3, pp. 1429-1430, 1990.

[Desr90] M. Desrochers, "Computer-based versus human assessment of vocal responses with developmentally handicapped individuals," *Ph. D. Thesis*. University of Manitoba; Canada, April 1990, 267 pp.

[FeLo89] K. Ferens and C. Love, "A speech recorder and synthesizer using ADPCM," *B. Sc. Thesis*. University of Manitoba; Canada, May 1989, 114 pp.

[HaWa90] J. Hampshire II and A. Waibel, "A novel objective function for improved phoneme recognition using time delay neural networks," *IEEE INNS International joint conference on neural networks*, vol. 1, pp. 235-241, 1989.

[Holm88] J. Holmes, *Speech Synthesis and Recognition*. UK: Van Nostrand Reinhold, 1988, 198 pp.

[KiPR87] J. Pear, W. Kinsner, and D. Roy, "Vocal shaping of retarded and autistic individuals using speech synthesis and recognition," *Proc. IEEE Engineering in Medicine and Biology Soc.*, IEEE Cat. no. 87 CH2513-0, pp. 1787-1788, 1987.

[Klim87] G. Klimenko, "A study of ADPCM, CVSD, and phoneme speech coding techniques," *M.Sc. Thesis*. University of Manitoba; Canada, August 1987, 228 pp.

[KlKi87] G. Klimenko and W. Kinsner, "A study of CVSD, ADPCM, and PSS speech coding techniques," *Proc. IEEE Engineering in Medicine and Biology Soc.*, IEEE Cat. no. 87 CH2513-0, pp. 1797-1798, 1987.

[KMRW87] D. Kewley-Port, D. Maki, D. Reed, and C. Watson, "Speaker-dependent speech recognition as the basis for a speech training aid," *Proc. IEEE Neural Networks*, Cat. no. 87 CH2396-0, pp. 372-375, 1987.

[Koho88] T. Kohonen, "The "neural" phonetic typewriter," *Computer Magazine*, pp. 11-22, March 1988.

[LeRo90] S. Levinson and D. Roe, "A perspective on speech recognition," *IEEE Communications Magazine*, pp. 28-34, Jan. 1990.

[LoFK89] C. Love, K. Ferens, and W. Kinsner, "A speech recorder and synthesizer using ADPCM," *Proc. IEEE Engineering in Medicine and Biology Soc.*, IEEE Cat. no. 89 CH2770-6, pp. 659-660, 1989.

[LoKi90] C. Love and W. Kinsner, "A phonemic recognizer for speech therapy using a neural network model," *Proc. Canadian Medical and Biological Engineering Soc.*, CMBS Cat. no. 90 CMBC-16-CCGB, pp. 93-94, 1990.

[McRu89] J. McClelland and D. Rumelhart, *Explorations in Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1989, 344 pp.

[Pars86] T. Parsons, *Voice and Speech Processing*. New York: McGraw Hill, 1986, 402 pp.

[PeKR87] J. Pear, W. Kinsner, and D. Roy, "Vocal shaping of retarded and autistic individuals using speech synthesis and recognition," *Proc. IEEE Engineering in Medicine and Biology Soc.*, IEEE Cat. no. 87 CH2513-0, pp. 1787-1788, 1987.

[Pete88] D. Peters, "A speech recognizer using LPC and DTW," *B.Sc. Thesis*. University of Manitoba; Canada, May 1988.

[RaSc78] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*. Englewood Cliffs, NJ.: Prentice Hall, 1978, 512 pp.

[Rein90] B. Müller and J. Reinhardt, *Physics of Neural Networks: Neural Network- An Introduction*. Berlin, GE.: Springer-Verlag, pp. 73-76, 1990.

[SaCh78] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans., on Acoust., Speech, and Signal Proc.*, vol. ASSP-26, no. 1, pp. 43-49, 1978.

[Swan87] C. Swanson, "A study and implementation of real-time linear predictive coding of speech," *M.Sc. Thesis*. University of Manitoba; Canada, August 1987, 238 pp.

[Syma89] Symantec Corp., *Think C*, Cupertino, CA: Symantec Corp., 1989, 511 pp.

[VMRZ88] T. Vogl, J. Mangis, A. Rigler, W. Zink, and D.Alkon, "Accelerating the convergence of the back-propagation method," *Biological Cybernetics*, vol. 59, pp. 257-263, 1988.

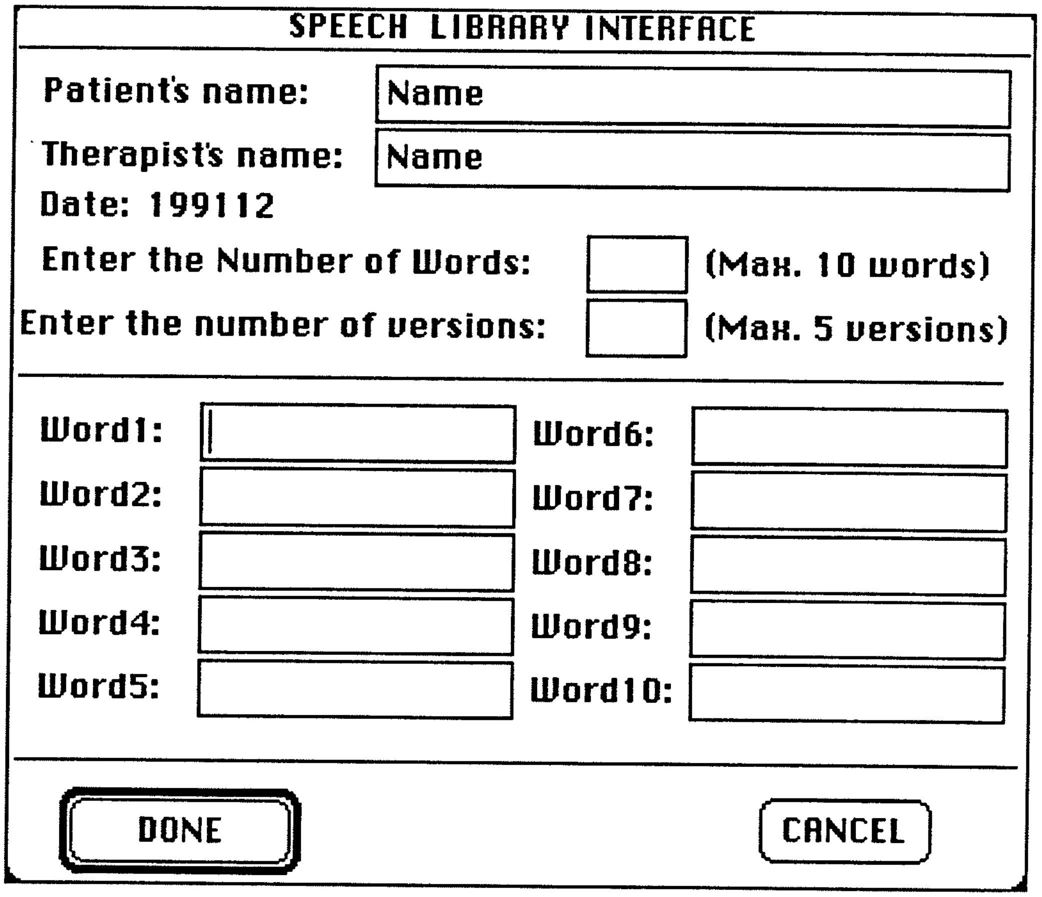
[Watr90] R. Watrous, "Phoneme discrimination using connectionist networks applied to /b/, /d/, and /g/ discrimination," *J. Acoust. Soc. Am.*, vol. 87, no. 3, pp. 1301-1309, March 1990.

APPENDIX A

Recognition System Software Dialog Boxes

NEW LIBRARY:

The *SPEECH LIBRARY INTERFACE* dialog box is used to acquire the various utterance labels to be used during the session. The speech therapists name and individuals name are entered for future reference. The label length for an utterance is 20 characters. Next, the number of utterances and versions are entered. If, for example, only 5 words are to be recorded, then only 5 labels should be entered from the ten possible locations. Upon completion, the individual either selects *DONE* or *CANCEL* to continue. *DONE* saves the entered data while *CANCEL* reverts to the previous values.



SPEECH LIBRARY INTERFACE

Patient's name: Name

Therapist's name: Name

Date: 199112

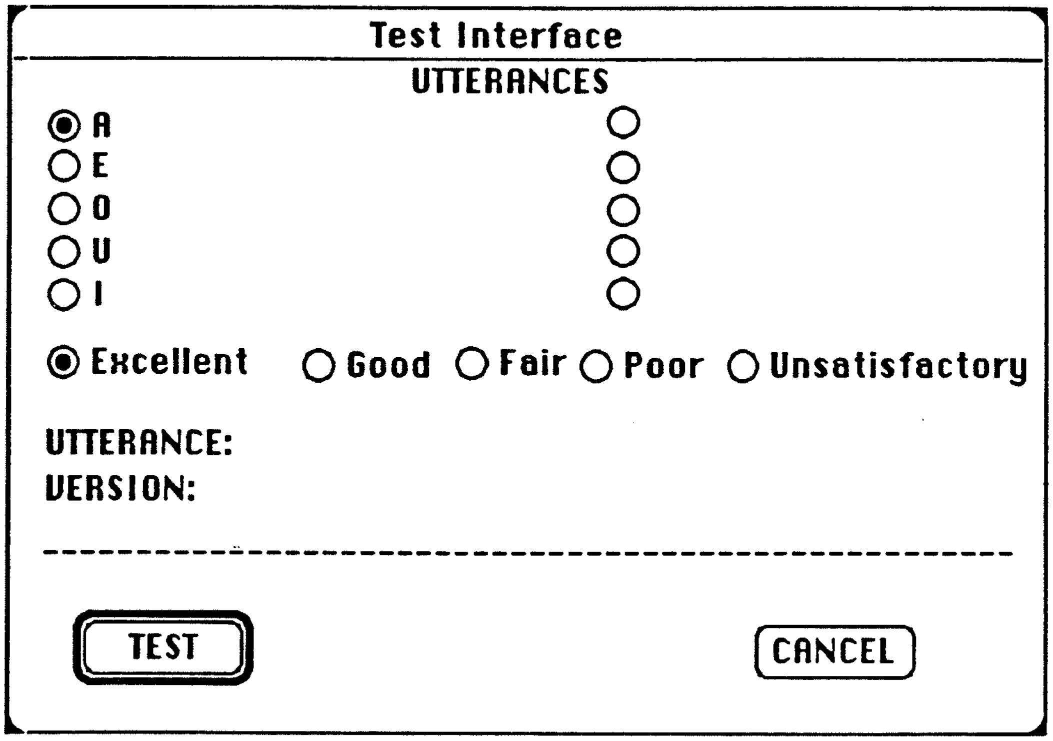
Enter the Number of Words: (Max. 10 words)

Enter the number of versions: (Max. 5 versions)

Word1:		Word6:	
Word2:		Word7:	
Word3:		Word8:	
Word4:		Word9:	
Word5:		Word10:	

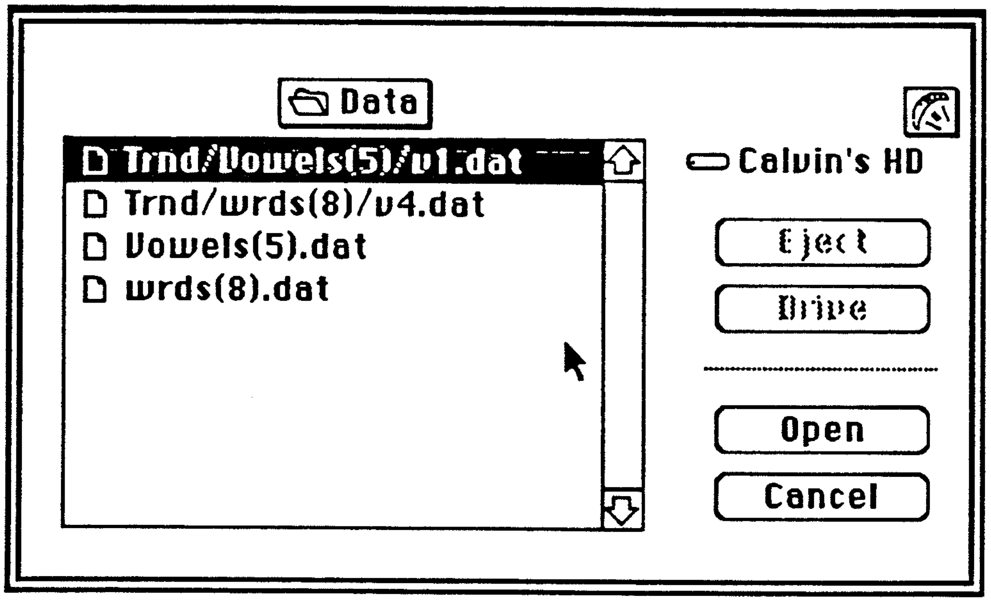
NETWORK TEST INTERFACE:

The test interface involves three sections. The top segment uses radio buttons to allow the individual to select the utterance and version they wish to use to test the BP neural network. There can be up to 10 utterance labels with up to 5 versions for every utterance implying that the neural network can accommodate up to a maximum of 50 patterns. The second segment is used for display purposes. The classified utterance and version resulting from propagating the selected pattern through the neural network is displayed in the middle segment. The bottom segment is used to either initiate testing using the *TEST* button or *CANCEL* to exit from the dialog box.



FILE SELECTION INTERFACE:

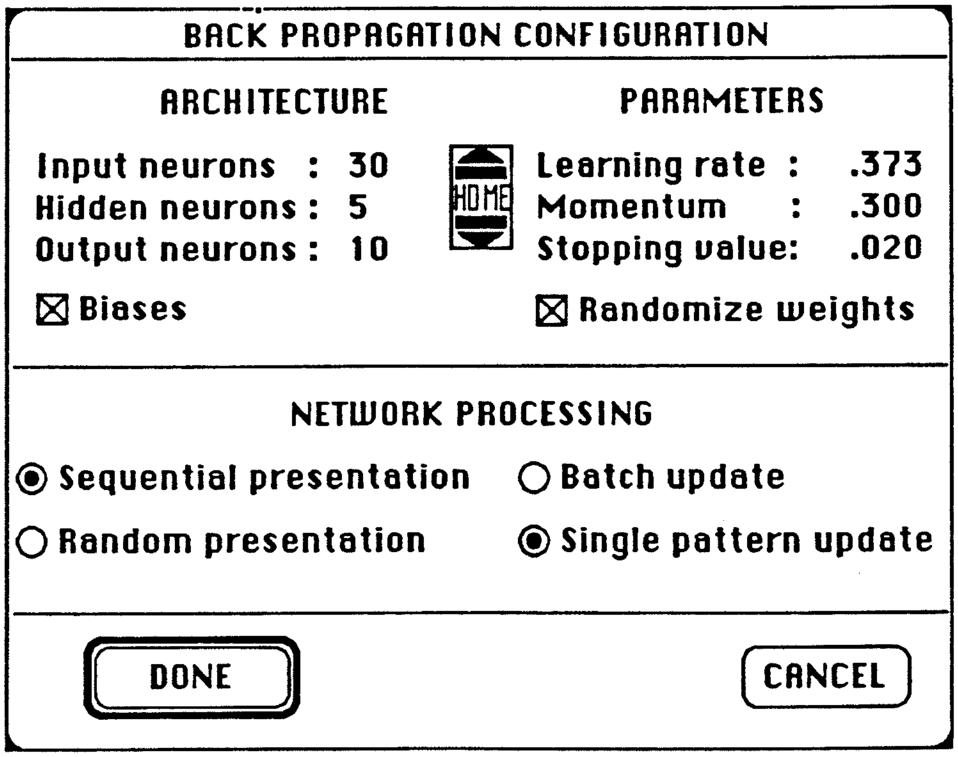
The file selection interface is the default standard interface provided in the Macintosh tool box. File filters, filter out the files that do not have a *CREATOR* ('cris') and *FILETYPE* ('TEXT'). The dialog interface allows the individual to load data that is to be used for the speech recognition software.



Trnd/Vowels(5)/v1.dat
Trnd/wrds(8)/v4.dat
Vowels(5).dat
wrds(8).dat

NETWORK SETUP :

The *NETWORK SETUP* dialog has three sections. The top section contains two subsections. The sub-section on the left describes the network architecture while that on the right describes the network training parameters. The input and output neurons are not adjustable but depend on how many utterances are being used in the session. They are automatically set by the software. The hidden neurons value is adjustable. To adjust a parameter in the top section, first click the *value* of the parameter. Next, click on the adjustor boxes indicated by the *up/down* arrows icons and the *home* icon. The *up/down* arrow icons increase and decrease the parameter value by a fixed, predetermined, increment. The *home* icon will change the parameter to the middle range value. The *biases* and the *random weights* checkboxes toggle between on and off. An active check box is indicated by an *X* filled in the box while an empty check box is indicated by an empty square. To change a check box value, simply click on it once. The middle segment, indicated by *NETWORK PROCESSING,* is used to select the network training methods. The two radio buttons on the left describe the presentation method of patterns to the network, which are either random or sequential. Next, the middle right sub-section indicates the mode of updates, which are either batch or single. The last section is used to either *SAVE* the selections or *CANCEL* and revert to what was used prior to opening the dialog box.


BACK PROPAGATION CONFIGURATION

ARCHITECTURE

Input neurons : 30
Hidden neurons : 5
Output neurons : 10

☒ Biases

PARAMETERS

Learning rate : .373
Momentum : .300
Stopping value: .020

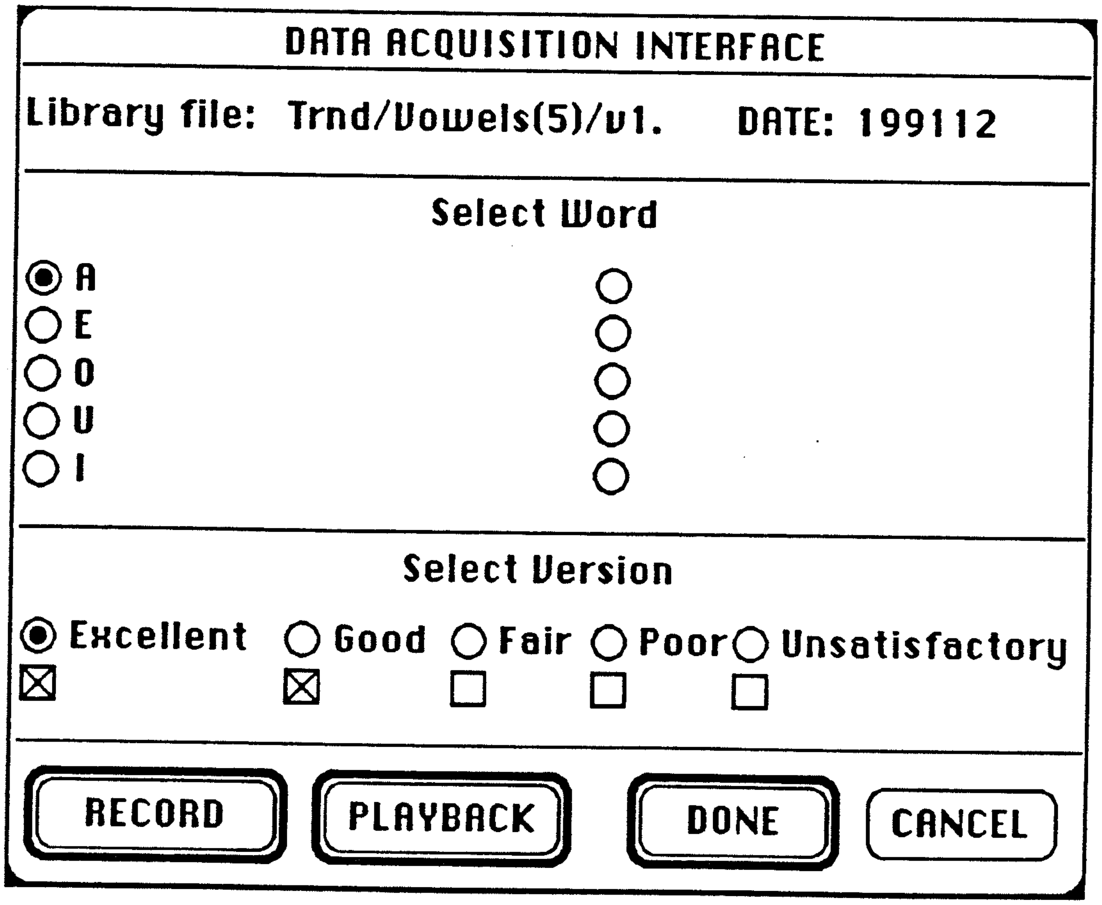
☒ Randomize weights

NETWORK PROCESSING

● Sequential presentation
○ Random presentation
○ Batch update
● Single pattern update

DATA ACQUISITION INTERFACE:

The data acquisition interface is used to assist in obtaining utterance data. To record an utterance, select an utterance label and an utterance version. If fewer than five versions were selected then only those versions can become active. Once a version and utterance label have been chosen, click the *RECORD* to record the utterance. The recording will last one second, and the small check box located below the utterance version will become active upon completion of the recording. If attempts are made to record over an existing recording, an *ALERT* dialog box will be displayed indicating this action. It is possible to re-record an utterance by closing the *ALERT* dialog box and recording the selection. The *PLAYBACK* button allows the selected utterance to be played back.



DATA ACQUISITION INTERFACE

Library file: Trnd/Vowels(5)/v1.   DATE: 199112

Select Word

[✓ A]
E
O
U
I

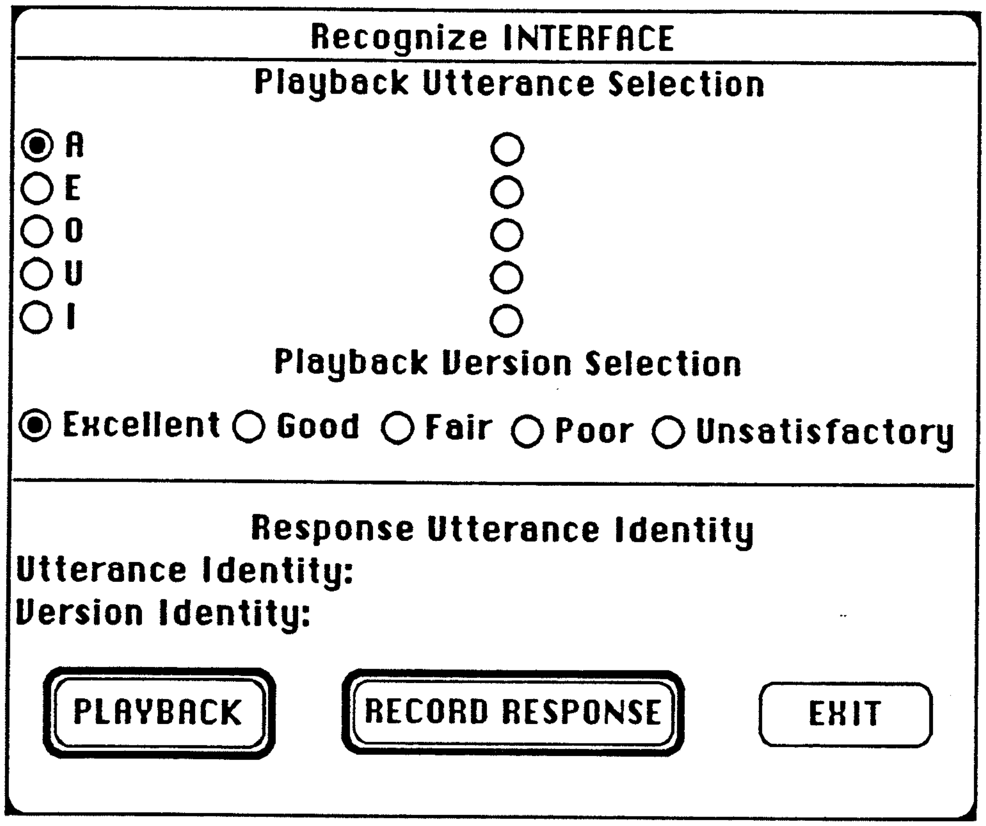
Select Version

[✓ Excellent] Good Fair Poor Unsatisfactory

☒ ☒ ☐ ☐ ☐

RECOGNITION INTERFACE:

The *RECOGNIZE* interface is used to identify an unknown utterance using the trained neural network. The utterance and version are first selected in the top section and the *PLAYBACK* button is depressed to hear what the desired utterance should sound like. Next, click the *RECORD* to record the utterance and speak into the microphone. One second of speech is recorded and the unknown pattern will be identified using the BP neural network. The result is automatically displayed in the *Response Utterance Identity* section of the dialog box.


Recognize INTERFACE

Playback Utterance Selection

[✓ A]
E
O
U
I

Playback Version Selection

[✓ Excellent] Good Fair Poor Unsatisfactory

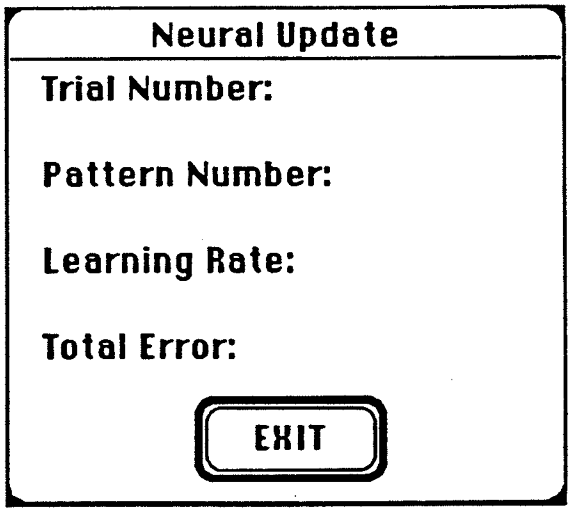
Response Utterance Identity

Utterance Identity:

Version Identity:

NEURAL UPDATE:

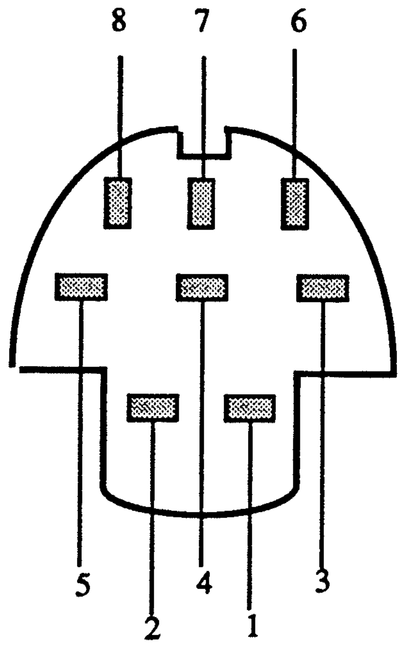
This dialog box is used to display the neural network training progress. The parameters displayed are: (i) *Trial Number,* which indicates how many trials have elapsed since beginning training (this value is *always* displayed as the number of complete pattern set presentation), (ii) *Pattern Number,* which represents the pattern number that is currently being presented to the neural network, (iii) learning rate describes the value of the learning rate used for that pattern set and, (iv) *Total Error* which represents the total internal error of the network over the entire pattern set, whether or not single or batch update is used. The reason for this is to provide consistency in the meaning throughout the various training configurations and in setting the *Stop Value.*



APPENDIX B

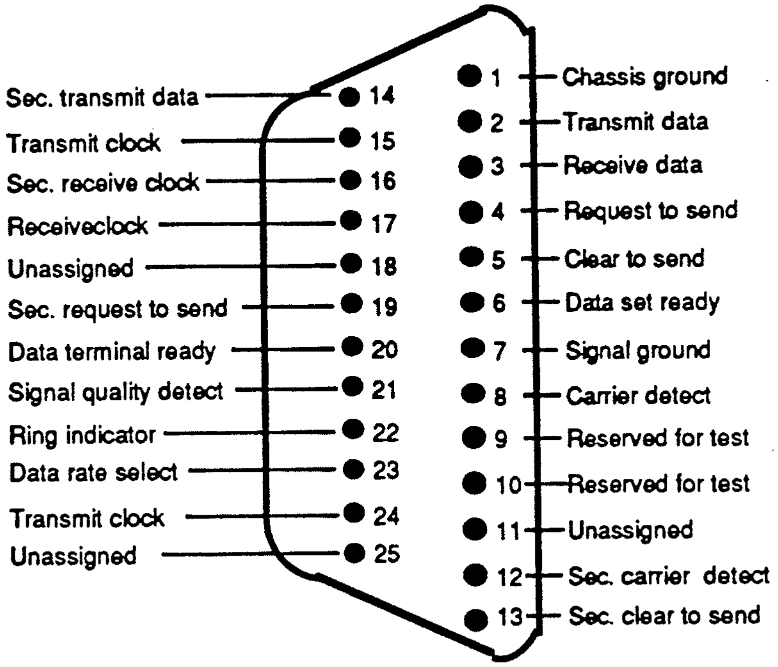
Speech Recognition System Hardware Description

Macintosh Serial Port Pinout description:

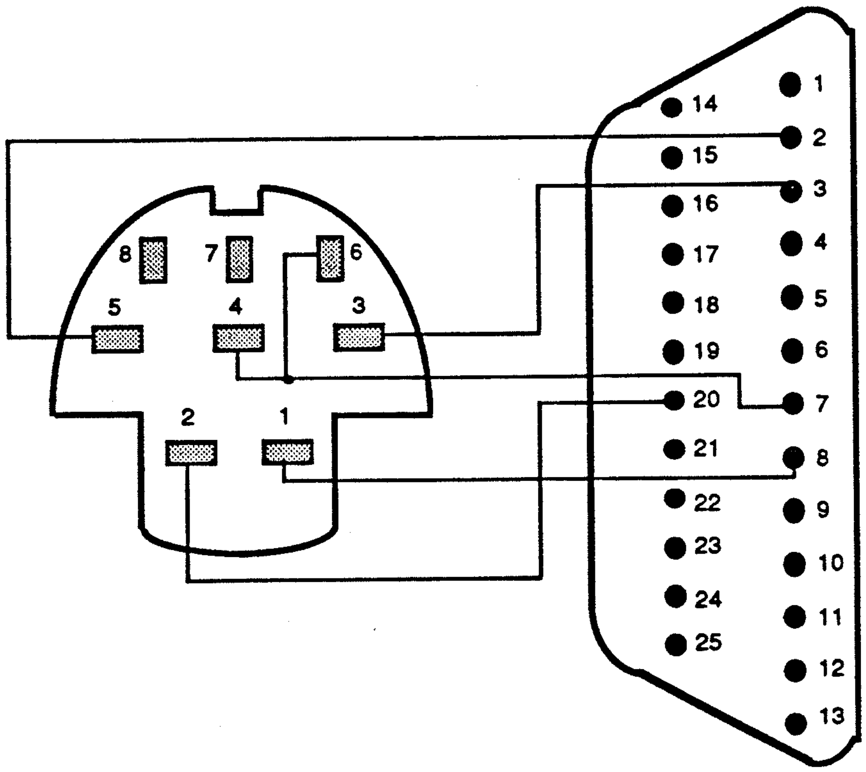


Pin Number	Signal Name	Signal Description
1	HSKo	Handshake out
2	HSKi	Handshake in
3	TXD-	Transmit data -
4	GND	Signal ground
5	RXD-	Receive data -
6	TXD+	Transmit data +
7	NC	No Connection
8	RXD-	Receive data -

IBM RS-232 serial port pinout description:



Macintosh to IBM serial connection description:



Microphone: (Model: SM10A)

Description: This is a hands free microphone that is used for close talking vocal pickup. It has been designed for environments including computer interactive systems. This microphone is noise cancelling and is uni-directional. It provides strong, professional sound quality voice response completely free of background noise and *pop*.

Technical specifications:

Bandwidth: 50 to 15000 Hz.

Microphone pickup: Cardioid (unidirectional)

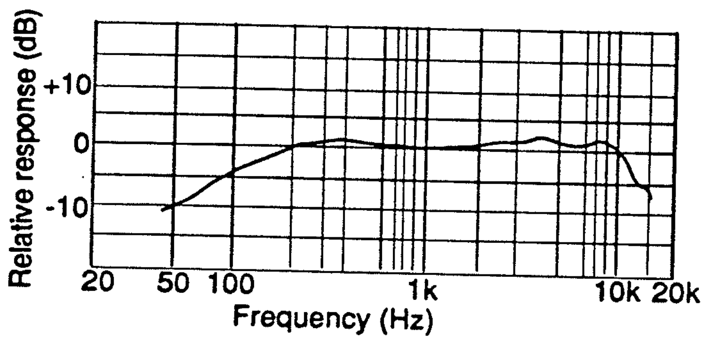
input impedance: 150 Ω

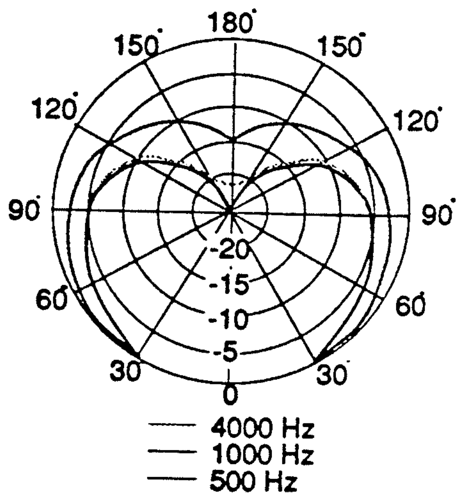
Signal strength: 4.5 mV (-47.0 dB, 0 dB = 1V/100μbars)

-66.0 dB, 0 dB = 1 mW/10μbar

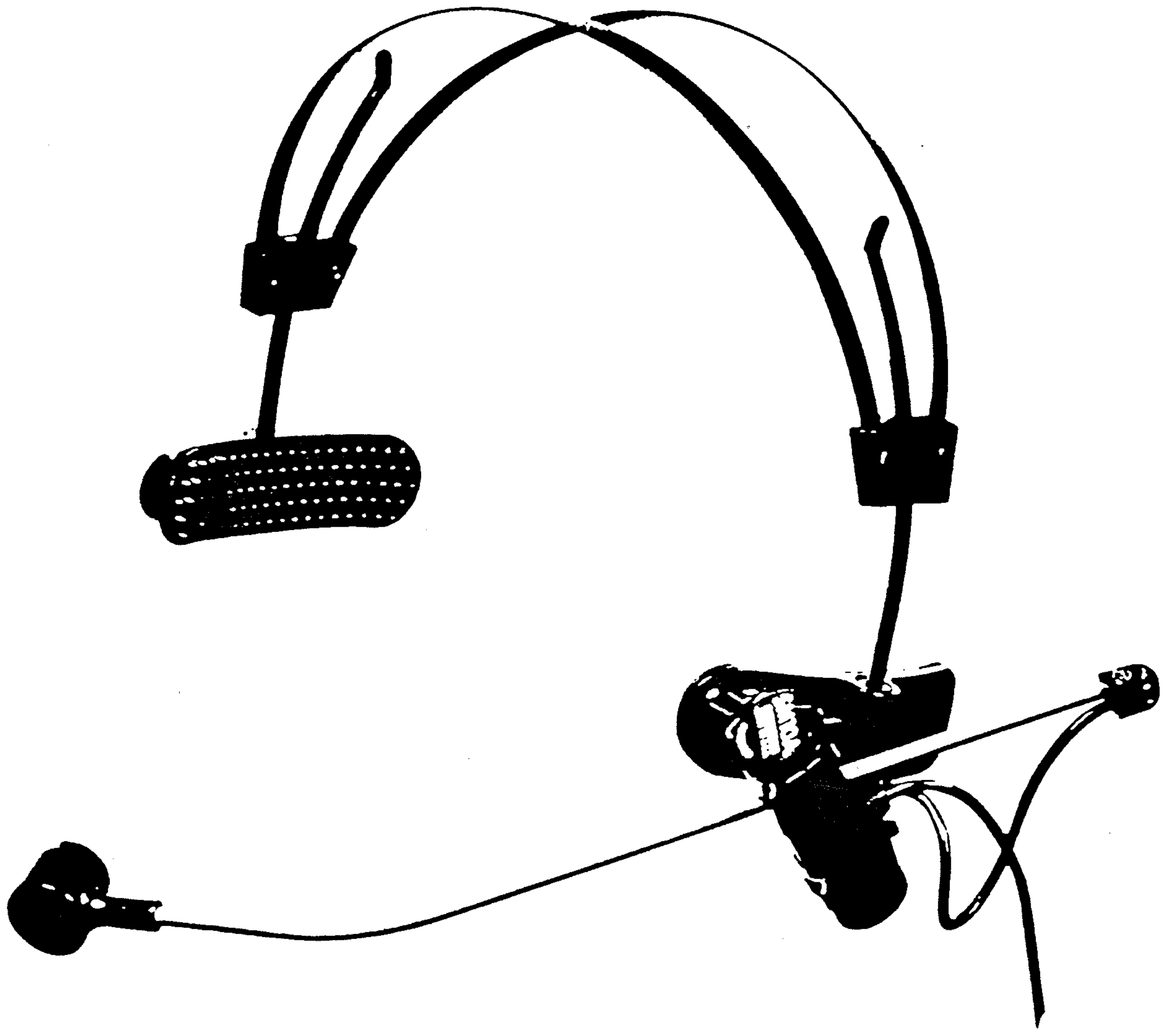
Microphone cabling: 1.5 m, two-conductor shielded with three-pin professional audio connector at equipment end.

Weight: 78 grams /Height: 14 mm /Diameter: 9/16 x 5/8 in.





SM10A microphone (courtesy of Shure Inc.)



SHURE®

SM10A
(HEAD-WORN MICROPHONE)

APPENDIX C

Source Code Listing of Macintosh Speech Recognition System Software

```
/*********************************************************************************
This software consists of the speech recognition program.  Included are the following components:
    Backprop.c
    MyFunctions2.c
    CBackpropApp.c
    CBackpropDoc.c
    CBackpropPane.c
    Serial Routines.c
    LPCIBM.c
This software is written by Chris D. Love.
University of Manitoba
Department of Electrical and Computer Engineering
Start Date:             June24, 1990
Completion Date:        January 10, 1991.
#define _H_CBackpropcmd
/*          HEADER FILE FOR MY APPLICATION COMMANDS                          */
/*                      My Back Propagation Menu Summary                     */

#define     MENUacquire         4000  /* Acquire data                        */
#define     MENUtrain           4001  /* Training Menu                       */
#define     MENUuse             4002  /* Use and Recognition Menu            */
#define     MENUutilities       4003  /* Utilities and stuff                 */
#define     MENUweights         203   /* Hier MenuID for network weights     */
#define     MENUgrademethod     204   /* Hier MenuID for grading method      */

/*                              My Dialog ID Summary                         */
#define     DLOGspeechlib       500   /* DLOG id for speech library          */
#define     DLOGrecord          501   /* DLOG id for recording interface     */
#define     DLOGrecognize       502   /* DLOG id for recognizing interface   */
#define     DLOGbp              503   /* DLOG id for BP setup                */
#define     DLOGtest            504   /* DLOG id for BP testing              */
#define     DLOGbpupdate        505   /* DLOG id for BP progress             */

/*                      WINDOW RESOURCE ID DEFINITIONS                       */
#define     WIND_Backprop       2000L           /* WIND resource ID          */

    /*******************My Back Propagation Command Summary******************/
/*                      Data Acquisition Commands                            */
#define   cmdAcquire            2000L           /* Record and Playback word  */
```

```
#define   cmdNewSound          2001L       /*Record and play new sound          */

/*                             Training Commands                            */
#define   cmdSetupTrain        2007L       /* Setup Network Training            */
#define   cmdTestNetwork       2008L       /* Test network(train patts)         */
#define   cmdTrainNetwork      2009L       /* Train Network                     */
#define   cmdNetWeights        2010L       /* Set Network Weights               */
#define   cmdNewWeights        2020L       /* Initialize weights                */
#define   cmdFileWeights       2019L       /* Load weight file from disk        */
#define   cmdGradeRate         2014L       /* Grading measure for network       */

/*                    Use and Recognition Commands                          */
#define   cmdRecognize         2012L       /* Synthesize and recognize utterance*/

/*                             Utilities                                    */
#define   cmdUtterSpect        2015L       /* Word Spectrogram plot             */
#define   cmdTimeAmpl          2016L       /* Time Amplitude Plot               */
#define   cmdSpeechSplice      2017L       /* Speech Splicing (Kens)            */
#define   cmdPrintPlot         2018L       /* Print plot                        */

/*                             MENUgrademethod                              */
#define   cmdchebychev         3000L       /* Chebychev distance measure        */
#define   cmdeuclidean         3001L       /* Euclidean distance measure        */
#define   cmdhamming           3002L       /* Hamming distance measure          */
#define   cmdminkowski3        3003L       /* Minkowski distance measure P=3*/
#define   cmdminkowski4        3004L       /* Minkowski distance measure P=4*/
#define   cmdminkowski5        3005L       /* Minkowski distance measure P=5*/
#define   cmdminkowski6        3006L       /* Minkowski distance measure P=6*/
#define   cmdminkowski7        3007L       /* Minkowski distance measure P=7*/

/*                             HierMENUminkowski                            */
#define   cmdThree             3036L       /* Minkowski power for metric        */
#define   cmdFour              3037L       /* Minkowski power for metric        */
#define   cmdFive              3038L       /* Minkowski power for metric        */
#define   cmdSix               3039L       /* Minkowski power for metric        */
#define   cmdSeven             3040L       /* Minkowski power for metric        */
/*                    End of Back propagation command Summary               */
```

```
/*******************************************************************************
 CBackpropDoc.h                        Interface for the BackpropDoc Class
*******************************************************************************/
#define _H_CBackpropDoc

#include <CDocument.h>                  /* Interface for its superclass       */
#include <CDataFile.h>                  /* Interface for its dataclass        */
    /*    Class Declaration */
    struct CBackpropDoc : CDocument {
                            /** Instance Variables **/
    Boolean     Presentation,UpdateType,Biases,RandomWeights,SetWts,*theVersions;
    Boolean     DoneNew,DoneRecord,DoneNetwork;
    short       Grade,Power,RecognizeVersion,TestType,NumOfPatt,NumOfWords;
    short       NumInNuron,NumHidNuron,NumOutNuron,NumOfVers;
    float       Momentum,StopValue,LearnRate,UpperLearnThresh,LowerLearnThresh;
    double      *LPCData,*HOWt,*IHWt,*HiddenBias,*OutputBias;
    Str255      PatientName,TherapistName;
    Str255      Wordlabel1,Wordlabel2,Wordlabel3,Wordlabel4,Wordlabel5;
    Str255      Wordlabel6,Wordlabel7,Wordlabel8,Wordlabel9,Wordlabel10;
    SFReply     macSFReply;
    Rect        itsFrame;

                /** Instance Methods  {OverRide from CDocument} **/
    void        IBackpropDoc(CBureaucrat *itsSupervisor);
    void        DoCommand(long theCommand);
    void        UpdateMenus(void);

    void        NewFile(void);
    void        OpenFile(SFReply *macSFReply);
    void        BuildWindow(Handle theData);

    Boolean     DoSave(void);
    Boolean     DoSaveAs(SFReply *macSFReply);
    void        DoRevert(void);

    /*      MY FUNCTIONS FOR THE BACK PROPAGATION SOFTWARE                    */
    void  Acquire();                        /* recording & playback process   */
    void NewSounds();                       /* Mac Sound Manager              */

    void  SetupTrain();                     /* BP construction                */
    void TrainNetwork();                        /* BP Training                 */
    void TestNetwork();                         /* BP Testing                  */
    void SetNetWts(Boolean Weights);            /* BP Set Net Weight structure */
    void Recognize();                           /* Recognition of word         */

    void Spectplot();
    void Speechsplice();
    void Amplplot();
};



/*******************************************************************************
 CBackpropApp.h          Interface for the BackpropApp Class
*******************************************************************************/
#define _H_CBackpropApp

#include <CApplication.h> /* Interface for its superclass                  */

struct CBackpropApp : CApplication {  /* Class Declaration                 */
/*  Declare your Application's instance variables                          */
    SFReply             macSFReply;            /* Standard File reply record */

/* Instance Methods                                                        */
    void IBackpropApp(void);
    void SetUpFileParameters(void);
    void SetUpMenus(void);
    void Exit(void);
    void DoCommand(long theCommand);
    void UpdateMenus(void);
    void CreateDocument(void);
    void OpenDocument(SFReply *macSFReply);
};
/*******************************************************************************
 Backprop.c     Template for a main program for an application built with the THINK
                Class Library
                Written by Chris D. Love / June 1990-January 3,1991
*******************************************************************************/
```

```
#include "CBackpropApp.h"/* XXX Application subclass header                    */

                        /*              My Functions                          */
void    OutlineButton   (DialogPtr DialogName,short WhichOne);
void    SetRadioButton  (short WhichItem, DialogPtr DialogName, Handle *olditem);
void    SetDate         (DialogPtr DialogName, short DateItemNo);
void    SetWeights      (double *theArray, short Length,Boolean Randomize);
short   GetPatterns     (Boolean *anArray, short NumPatterns);
void    SetWords        (DialogPtr DialogName, Str255  *Word1, Str255 *Word2,
                                Str255 *Word3, Str255 *Word4, Str255 *Word5, Str255 *Word6,
                                Str255 *Word7, Str255 *Word8, Str255 *Word9, Str255*Word10);
void    GetWords        (DialogPtr DialogName, Str255 *PatName, Str255 *TherName,
                                Str255 *Word1, Str255 *Word2, Str255 *Word3, Str255 *Word4,
                                Str255 *Word5, Str255 *Word6, Str255 *Word7,
                                Str255 *Word8, Str255 *Word9, Str255 *Word10);
void    NetworkUpdate   (DialogPtr DialogName, short TrialNum, short thePattern, float LRate,
                                double TError);
void    PtoStr255       (char *srcString,  Str255 destString);
void    CtoStr255       (char *srcString, Str255 destString);
void    CalcTotError    (double *TPatt, double *OutNurons, short PatternNum,short theGrade,
                                short NumONurons, short thePower, double *TError);
void    SetCheckBoxes   (DialogPtr DialogName, short WhichWord, Boolean *ArrayName, short
                                NumVers);
void    GetLPCData      (double *FloatptArray);
void    AscToFloat      (char *AscFloat, double *FloatptArray);
void    DisplayResults  (DialogPtr DialogName, double *theArray, short NumPatt, short NumVers);
void    PlaybackWord    (short WhichWord);
void    SetTrainingPat  (double *TPatt, short NumberONurons);
void    CalcForwardPass (double *InputArray, double *HiddenArray, double *OutputArray,
                                double *HBias, double *OBias, double *WIH, double *WHO,
                                short INurons, short HNurons, short ONurons, short PatNumber);
void    SetCurrentValues (DialogPtr DialogName, Boolean Bias, float Momtum, float StopVal,
                                float LearnRt, Boolean RandomWts);

extern CApplication         *gApplication;

void main(void)
{
    gApplication = new(CBackpropApp);       /* XXX Create and intialize subclass of CApplication       */
    ((CBackpropApp*)gApplication)->IBackpropApp();
    gApplication->Run();
    gApplication->Exit();
}
/*****************************************************************************
 CBackpropApp.h          Interface for the BackpropApp Class
 *****************************************************************************/
#define _H_CBackpropApp

#include <CApplication.h> /* Interface for its superclass                    */

struct CBackpropApp : CApplication {  /* Class Declaration                   */
/*  Declare your Application's instance variables                            */
    SFReply              macSFReply;              /* Standard File reply record    */

/* Instance Methods                                                          */
     void  IBackpropApp(void);
     void  SetUpFileParameters(void);
     void  SetUpMenus(void);
     void  Exit(void);
     void  DoCommand(long theCommand);
     void  UpdateMenus(void);
     void  CreateDocument(void);
     void  OpenDocument(SFReply *macSFReply);
 };

/*****************************************************************************
 CBackpropApp.c
              The BackpropApp Class     SUPERCLASS = CApplication
              Written by Chris D. Love / June 1990-January 3,1991
 *****************************************************************************/
/* Remember to use angle brackets for headers that are part of the THINK Class Library      */
#include <Global.h>
#include "CBackpropApp.h"
#include "CBackpropDoc.h"

extern    OSType                 gSignature;       /* Creator for Application's files    */

/*****************************************************************************
 IBackpropApp                        Initialize a BackpropApp object
 ****************************************************************************/
void     CBackpropApp::IBackpropApp()
{
    /* Initialize superclass                                                  */
    CApplication::IApplication(5, 20000L, 2000L);

/* Initialize global variables defined by your application and the instance variables of your  */
/* Application subclass.                                                      */
}

/****************************************************************************
 SetUpFileParameters {OVERRIDE}   Set parameters used by the Standard File Package
 ****************************************************************************/
void     CBackpropApp::SetUpFileParameters()
{
    inherited::SetUpFileParameters();  /* Use defaults from superclass          */
                                       /* Specify file parameters for the Application   */
                                       /* Specify a four-character signature for your application */
    gSignature = 'cris';
                                       /* Specify the number and types of files recognized.   */
    sfNumTypes = 4;
    sfFileTypes[0] = 'LIBR';
    sfFileTypes[1] = 'LOVE';
    sfFileTypes[2] = 'FSSD';
    sfFileTypes[3] = 'TEXT';
}

/****************************************************************************
DoCommand {OVERRIDE}                 Execute a command
    It's important to pass along commands not handled by this class to the inherited
    DoCommand method. These will be handled by the THINK Class Library by default.
****************************************************************************/
void     CBackpropApp::DoCommand(long              theCommand)
{
    switch (theCommand)
    {
        /* &&& Cases for commands handled by application                      */
        default:  /* Invoke inherited method to handle other commands          */
            inherited::DoCommand(theCommand);
            break;
    }
}

/*****************************************************************************
 UpdateMenus {OVERRIDE}                    Perform menu management tasks
*****************************************************************************/
void CBackpropApp::UpdateMenus()
{
    /* Enable the commands handled by your Application class; Enable standard commands      */
    inherited::UpdateMenus();
}

/*****************************************************************************
 SetUpMenus {OVERRIDE}
    Set up menus which must be created at run time, such as a Font menu. You
    can eliminate this method if application does not have any such menus.
*****************************************************************************/
void CBackpropApp::SetUpMenus()
{
    inherited::SetUpMenus();
    /* Superclass takes care of adding menus specified in a MBAR id=1and  Code for creating run-time
    menus   */
}
/*****************************************************************************
 Exit {OVERRIDE}         Program is about to terminate. Last chance to clean up.
$$$ Here's the place to delete temporary files or to automatically save settings. However, most
applications don't need to do anything here.
*****************************************************************************/
void CBackpropApp::Exit()
{
        /* &&& Exit code for your application                                        */
}

/*****************************************************************************
 Create Document {OVERRIDE}
    Make a document. This message is sent when the user chooses the "New" command.
*****************************************************************************/
void CBackpropApp::CreateDocument()
{
    CBackpropDoc          *theDocument;
    /* Create and initialize a Document  $$$ Passing "this" to IBackpropDoc                 */
    /*   means that the Application object is the supervisor of the Document                */

    theDocument = new(CBackpropDoc);
    theDocument->IBackpropDoc(this);
    theDocument->NewFile();
    /* Tell Document to make a new file $$$ Document is responsible for do                   */
    /*whatever's necessary to set up a new file (eg.displaying a  blank window)              */
}

/*******************************************************************************
OpenDocument {OVERRIDE}
Open an existing file and create a document object for displaying information. This message is sent
when the user chooses the "Open..." command or when the application was launched by double-
clicking on a file or selecting file(s) and choosing "Open" from the Finder. The  macSFReply
parameter is a record which contains information about the file to open (name, volume number, etc.).
 *******************************************************************************/
void CBackpropApp::OpenDocument(SFReply *macSFReply)          /* Standard File reply record*/
{
    CBackpropDoc          *theDocument;

    /* Create and initialize a Document $$$ Passing "this" to IBackpropDoc                   */
    /*   means that the Application object  is the supervisor of the Document                */
    theDocument = new(CBackpropDoc);
    theDocument->IBackpropDoc(this);

    /* Tell Document to open the file. $$$ The Document is responsible for                   */
    /*   creating whatever windows are necessary to display the file and                      */
    /*   actually reading the contents of the file from disk                                  */

    theDocument->OpenFile(macSFReply);
}
```

```
/*************************************************************************
 CBackpropDoc.c          The BackpropDoc Class          SUPERCLASS = CDocument
              Written by Chris D. Love / June 1990-January 3,1991
*************************************************************************/
#include  <Global.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  <math.h>
#include  <Commands.h>
#include  <CBartender.h>
#include  <CDesktop.h>
#include  <CDecorator.h>
#include  <CScrollPane.h>
#include  <CPanorama.h>
#include  <TBUtilities.h>
#include  <CControl.h>
#include  "CBackpropPane.h"
#include  "CBackpropcmd.h"
#include  "CBackpropDoc.h"

extern   CBartender     *gBartender;    /* Manages all menus                          */
extern   OSType         gSignature;     /* Creator for Application's files            */
extern   CDesktop       *gDesktop;      /* The visible Desktop                        */
extern   CDecorator     *gDecorator;             /* Decorator for arranging windows  */
extern   CBureaucrat    *gGopher;       /* The current boss in the chain of command   */

/*************************************************************************
 IBackpropDoc                          Initialize a BackpropDoc object
*************************************************************************/
void CBackpropDoc::IBackpropDoc(CBureaucrat                  *itsSupervisor)
{
    CDocument::IDocument(itsSupervisor, TRUE);

    /*   Default Program Settings                                                  */
    NumInNuron=30;       NumHidNuron=5;          IHWt=NULL;
    LearnRate=0.1;       Momentum=0.3;           StopValue=0.02;
    Grade=2;             RandomWeights=TRUE;     RecognizeVersion=1;
    DoneRecord=FALSE;    OutputBias=NULL;        theVersions=NULL;
    TestType=1;                 DoneNew=FALSE;          HiddenBias=NULL;
    Presentation=FALSE;         UpdateType=TRUE         LowerLearnThresh=0.05;
    Biases=TRUE;                Power=3;                UpperLearnThresh=2.5;
    DoneNetwork=FALSE;          HOWt=NULL;              LPCData=NULL;
}

/******************************************************************************
DoCommand {OVERRIDE}            Execute a command
    It's important to pass along commands not handled by this class to the inherited
    DoCommand method. These will be handled by the THINK Class Library by default.
******************************************************************************/
void CBackpropDoc::DoCommand(long  theCommand)
{
    char      theMessage[255];
    switch (theCommand)
    {
        case cmdAcquire:
            Acquire();
            break;
        case cmdNewSound:
            NewSounds();
            break;
        case cmdSetupTrain:
            SetupTrain();
            break;
        case cmdTestNetwork:
            TestNetwork();
            break;
        case cmdTrainNetwork:
            if (IHWt==NULL || LPCData[0]==0)
            {
                /* Set alert if no training data has been found                       */
                strcpy(theMessage,"\pNo utterances were recorded for training. Record the training data
                        by selecting 'Acquire' under the 'Acquire' menu.");
                ParamText(theMessage,' ',' ',' ');
                StopAlert(128,NULL);
                theCommand=cmdNull;
                itsMainPane->Refresh();
            }
        else
            TrainNetwork();
        break;
    case cmdNetWeights:
        case cmdNewWeights:     /*      SetWts=FALSE                                    */
        case cmdFileWeights:    /*      SetWts=TRUE                                     */
            SetWts=(Boolean)(2020-theCommand);
            SetNetWts(SetWts);
        break;
    case cmdRecognize:
        if (IHWt[0]==0)
        {
            /* Set alert if no network has been found                                   */
            strcpy(theMessage,"\pNo Network has been loaded or Trained. Train the network by
            selecting 'Train Network' under the 'Train' menu.");
            ParamText(theMessage,' ',' ',' ');
            StopAlert(128,NULL);
            theCommand=cmdNull;
            itsMainPane->Refresh();
        }
        else
            Recognize();
        break;
    case cmdGradeRate:
        case cmdchebychev:
        case cmdeuclidean:
        case cmdhamming:
        case cmdminkowski3:
        case cmdminkowski4:
        case cmdminkowski5:
        case cmdminkowski6:
        case cmdminkowski7:
            /* chebychev=1,euclidean=2 , hamming=3,minkowski=4-8                        */
            if (theCommand>cmdhamming)
            {
                    Grade=4;
                    Power=(short)(theCommand-3000);
            }
            else
                    Grade=(short)(theCommand-2999);
        break;
    case cmdUtterSpect:
        Spectplot();
        break;
    case cmdTimeAmpl:
        Amplplot();
        break;
    case cmdSpeechSplice:
        Speechsplice();
        break;

    default:  /* Invoke inherited method to handle other commands                    */
        if ((theCommand==cmdNew || theCommand==cmdOpen) && itsFile!=NULL)
        {
            /* Set alert box message                                                  */
            strcpy(theMessage,"\pYou must first close or Save the existing file first. Press 'Cancel'
            then proceed.");
            ParamText(theMessage,' ',' ',' ');
            StopAlert(128,NULL);
            theCommand=cmdNull;
            itsMainPane->Refresh();
        }

        if (theCommand==cmdClose || theCommand==cmdQuit)
        {
            gBartender->EnableCmd(cmdOpen);
            gBartender->EnableCmd(cmdNew);
            if (IHWt[0]!=0)          /* Could have come from 'forget' in train              */
            {
                    free(HOWt);
                    free(IHWt);
                    free(LPCData);
                    free(theVersions);

                    if (Biases)
                    {
                    free(HiddenBias);
                    free(OutputBias);
                    }
```

```
                }

                /* Set all Array pointers to NULL indicating not in use                    */
                theVersions=NULL;          HiddenBias=NULL;   OutputBias=NULL;
                DoneNetwork=FALSE;         LPCData=NULL;      IHWt=NULL;
                DoneRecord=FALSE;          HOWt=NULL;
            }
            inherited::DoCommand(theCommand);
            break;
    }
}

/*****************************************************************************
 NewFile {OVERRIDE}      Set up a document with a new file,
                         usually in response to the "New" command in the File menu.
*****************************************************************************/
void CBackpropDoc::NewFile()
{
    Str255      thePTxt;
    char        theCTxt[255];
    short       dialogID,itemType,itemHit,wCount;/* Index number of new window     */
    Str63       wNumber;                    /* Index number as a string      */
    Str255      wTitle;                     /* Window title string           */
    Ptr         dStorage;
    WindowPtr   behind;
    GrafPtr     CurrentPort;
    DialogPtr   Speechbox;
    Handle      item;
    Rect        box;

    /* Initialize space for data for the worst case                                */
        HiddenBias=calloc(15, sizeof(double));
        OutputBias=calloc(50, sizeof(double));
        HOWt=calloc(15*50, sizeof(double));
        IHWt=calloc(30*15, sizeof(double));
        LPCData=calloc(1500,sizeof(double));
        theVersions=calloc(50, sizeof(char));

    /*  Build an Operating Window                                                  */
        BuildWindow(NULL);
        itsWindow->GetTitle(wTitle);                  /* Append an index number to the         */
        wCount = gDecorator->GetWCount();             /*  default name of the window           */
        NumToString((long)wCount, wNumber);
        ConcatPStrings(wTitle, (StringPtr) "\p-");
        ConcatPStrings(wTitle, wNumber);
        itsWindow->SetTitle(wTitle);

    /* Set Cursor to arrow                                                                     */
        SetCursor(&arrow);
    /* Speech Library Dialog box                                                               */
        behind=(WindowPtr)(-1L);
        dialogID=DLOGspeechlib;
        dStorage=NULL;
        Speechbox = GetNewDialog(dialogID,dStorage,behind);

    /* Outline default buttons                                                     */
        GetPort(&CurrentPort);
        OutlineButton(Speechbox,1);
        SetPort(CurrentPort);

    /*Set Current date in dialog box                                               */
        SetDate(Speechbox,17);

    itemHit=0;
    while( itemHit !=OK && itemHit !=Cancel)
    {
        ModalDialog(NULL, &itemHit);
        GetDItem(Speechbox, itemHit, &itemType, &item, &box);
    }

    if (itemHit==OK)        /* Save all relevant data         to file instance variables       */
    {
        /* Obtain Data from Dialog box and make it global to the program                   */
           GetWords(Speechbox, &PatientName, &TherapistName, &Wordlabel1,
           &Wordlabel2, &Wordlabel3, &Wordlabel4, &Wordlabel5, &Wordlabel6,
           &Wordlabel7, &Wordlabel8, &Wordlabel9, &Wordlabel10);

        /* Obtain the number of words and versions to be used in session                   */
        GetDItem(Speechbox,31,&itemType,&item,&box);
        GetITexl(item,thePTxt);
        Str255toC(thePTxt, theCTxt);
        sscanf(theCTxt,"%d",&NumOfWords);
        GetDItem(Speechbox,34,&itemType,&item,&box);
        GetITexl(item,thePTxt);
        Str255toC(thePTxt, theCTxt);
        sscanf(theCTxt,"%d",&NumOfVers);
        NumOfPatt=NumOfWords*NumOfVers;
        NumOutNuron=NumOfPatt;

        dirty = TRUE;                /* There is data that needs to be saved          */
    }
    DisposDialog(Speechbox);         /* Dispose of Dialog Box                         */
    itsWindow->Select();  /* Make this the active window                              */
    gBartender->DisableCmd(cmdOpen);

    ((CBackpropPane*)itsMainPane)->GetFrame(&itsFrame);
    ((CBackpropPane*)itsMainPane)->Draw(&itsFrame);
    DoneNew=TRUE;
}

/**************************************************************************
 OpenFile {OVERRIDE}   Open a file for a document
**************************************************************************/
void CBackpropDoc::OpenFile(SFReply *macSFReply)
{
    Str63   theName;
    OSErr   theError;
    short   test;
    long    where;

    /* Initialize space for data for the worst case                                   */
        HiddenBias=calloc(15, sizeof(double));
        OutputBias=calloc(50, sizeof(double));
        HOWt=calloc(15*50, sizeof(double));
        IHWt=calloc(30*15, sizeof(double));
        LPCData=calloc(1500,sizeof(double));
        theVersions=calloc(50, sizeof(char));

    /* Create and initialize a new file object.                                       */
```

```
itsFile = new(CDataFile);
((CDataFile *)itsFile)->IDataFile();
itsFile->SFSpecify(macSFReply);

/* Open file with read/write access                                          */
theError = ((CDataFile *)itsFile)->Open(fsRdWrPerm);

/* Read in Patients Name, Therapists Name and Vocabulary                     */
     ((CDataFile *)itsFile)->SetMark((long) 0,fsFromStart);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&PatientName),   (long)31);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&TherapistName), (long)31);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel1),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel2),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel3),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel4),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel5),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel6),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel7),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel8),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel9),    (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Wordlabel10),   (long)20);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumOfWords),    (long)10);
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumOfVers),     (long)10);

/* Read the recorded versions from the data file                             */
     ((CDataFile *)itsFile)->ReadSome(theVersions,(long)(sizeof(char)*50));

/* Read the recorded LPC Data from disk This is 10bytes if the regular processor is used   =>
     1500*10= 15000 bytes This is 12bytes if the floating pt. co processor is used => 1500*12=
     18000 bytes                                                             */
     ((CDataFile *)itsFile)->ReadSome((Ptr)LPCData,(long)(sizeof(double)*1500));

/* Read the parameters, biases, and weights form the data file               */
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumInNuron),   (long)sizeof(short));
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumHidNuron),  (long)sizeof(short));
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumOutNuron),  (long)sizeof(short));
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&NumOfPatt),    (long)sizeof(short));
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&Biases),       (long)sizeof(char));
     ((CDataFile *)itsFile)->ReadSome((Ptr)(&UpdateType),   (long)sizeof(char));
```

```
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&Presentation),    (long)sizeof(char));
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&RandomWeights),(long)sizeof(char));
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&LearnRate),       (long)sizeof(float));
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&Momentum),        (long)sizeof(float));
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&StopValue),       (long)sizeof(float));
        ((CDataFile *)itsFile)->ReadSome((Ptr)(&Grade),           (long)sizeof(short));

    BuildWindow(NULL);
    /* Set window title to the name of the file                                   */
    itsWindow->SetTitle(macSFReply->fName);
    itsWindow->Select();  /* Make this the active window */
    gBartender->DisableCmd(cmdOpen);
}

/*****************************************************************************
 DoSave {OVERRIDE}      Save a file under the same name. Return TRUE if save was successful.
 *****************************************************************************/
Boolean  CBackpropDoc::DoSave()
{
    long      *thePosn;

    if (itsFile == NULL)   /* No file created yet. Save is the same as SaveAs.      */
        return(DoSaveFileAs());
     else
     {                            /*Write out information to itsFile             */
        ((CDataFile *)itsFile)->SetLength((long) 65536);

        ((CDataFile *)itsFile)->SetMark((long) 0,fsFromStart);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&PatientName),   (long)31);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&TherapistName),(long)31);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel1),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel2),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel3),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel4),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel5),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel6),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel7),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel8),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel9),    (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Wordlabel10),  (long)20);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumOfWords), (long)10);
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumOfVers),  (long)10);
        DoneNew=FALSE;

        ((CDataFile *)itsFile)->WriteSome(theVersions,(long)(sizeof(Boolean)*50));

        /* Write the recorded LPC Data to disk          (pointer at 332)              */
        ((CDataFile *)itsFile)->WriteSome((Ptr)LPCData,(long)(sizeof(double)*1500));
        DoneRecord=FALSE;

        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumInNuron),  (long)sizeof(short));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumHidNuron),(long)sizeof(short));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumOutNuron),(long)sizeof(short));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&NumOfPatt),   (long)sizeof(short));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Biases),      (long)sizeof(char));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&UpdateType),  (long)sizeof(char));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Presentation),  (long)sizeof(char));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&RandomWeights),(long)sizeof(char));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&LearnRate),   (long)sizeof(float));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Momentum),    (long)sizeof(float));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&StopValue),   (long)sizeof(float));
        ((CDataFile *)itsFile)->WriteSome((Ptr)(&Grade),       (long)sizeof(short));

        /* Write the weights and biases to disk         (pointer at 358+ 1500*double)      */
        ((CDataFile *)itsFile)-> WriteSome((Ptr)IHWt,(long)(sizeof(double) *NumInNuron
        *NumHidNuron));
        ((CDataFile *)itsFile)->WriteSome((Ptr)HOWt,(long)(sizeof(double)*NumOutNuron
        *NumHidNuron));
        if(Biases)
        {
            ((CDataFile *)itsFile)->WriteSome((Ptr)HiddenBias,(long)(sizeof(double)*NumHidNuron));
            ((CDataFile *)itsFile)->WriteSome((Ptr)OutputBias,(long)(sizeof(double)*NumOutNuron));
        }
        dirty = FALSE;/* Save makes document clean                                  */
        return(TRUE);   /* Save completed                                           */
    }
}
```

```
/*******************************************************************************
DoSaveAs {OVERRIDE}  Save a file under another name.
        Return TRUE if save was successful. Note that this message is sent by the
        DoSaveFileAs() method afterthe user picks a new name and confirms the save.
*******************************************************************************/
Boolean  CBackpropDoc::DoSaveAs(SFReply *macSFReply) /* Std File Reply              */
{
    /* If a file object already exists,  throw it out. This will also close the file.     */
    if (itsFile != NULL)
        itsFile->Dispose();
    /*Create and initialize a new file object.   Type casting will be necessary.        */
    itsFile = new(CDataFile);
    ((CDataFile *)itsFile)->IDataFile();
    itsFile->SFSpecify(macSFReply);                    /* Specify file parameters        */
    itsMainPane->Refresh();

    /* Create a new file on disk with desired creator and type.                          */
    itsFile->CreateNew(gSignature, 'TEXT');
    ((CDataFile *)itsFile)->Open(fsRdWrPerm);          /* Open file with read/write access  */

    /* Change name of window to match  new file name                                     */
    itsWindow->SetTitle(macSFReply->fName);

    return( DoSave() );                  /* Save contents to disk                        */
}

/*******************************************************************************
 BuildWindow {XXX}
    Build a window for a document. BuildWindow() takes care of the common stuff:
    Creating a window, and creating and arranging panes within the window.
*******************************************************************************/
void CBackpropDoc::BuildWindow(Handle       theData)
{
    CScrollPane *theScrollPane;
    Rect          sizeRect;

    /* Create a non-floating window using a WIND resource template                       */
    itsWindow = new(CWindow);
    itsWindow->IWindow(WIND_Backprop, FALSE, gDesktop, this);
    /* Specify maximum and minimum dimensions of our window                    */
    SetRect(&sizeRect,0,34,512,338);
    itsWindow->SetSizeRect(&sizeRect);

    /* Create a ScrollPane in the window                                       */
    theScrollPane = new(CScrollPane);

    /* Use a resource template to set parameters of the ScrollPane             */
    theScrollPane->IViewRes('ScPn', 1, itsWindow, this);

    /* Make ScrollPane fit snuggly to the frame of the window                  */
    theScrollPane->FitToEnclFrame(TRUE, TRUE);

    /* Put our BackpropPane inside the ScrollPane                              */
    itsMainPane = new(CBackpropPane);
    ((CBackpropPane*)itsMainPane)->IBackpropPane(theScrollPane, this, 0, 0, 0, 0,sizELASTIC,
    sizELASTIC);
    itsMainPane->FitToEnclosure(TRUE, TRUE);
    theScrollPane->InstallPanorama((CPanorama*)itsMainPane);

    /* BackpropPane may want to be the Gopher when Document is active so it can
    directly receive key and menu commands                                     */
    itsGopher = itsMainPane;
}

/********************************************************************************
DoRevert {OVERRIDE} Throw out the current contents of a document and revert to the last saved version.
********************************************************************************/
void CBackpropDoc::DoRevert()
{
    Point    homePos;            /* Home position of a Panorama                */

    /* Dispose of current (in memory) contents of the document                 */
    /* and read the information back from the file on disk                     */

    /* If you use Panoramas, remember to reset them back to the home position  */
    ((CPanorama*)itsMainPane)->GetHomePosition(&homePos);
    ((CPanorama*)itsMainPane)->ScrollTo(homePos, FALSE);
    itsMainPane->Refresh();          /* Force update of contents                 */
    dirty = FALSE;                   /* Reverts to a clean document              */
}

/******************************************************************************
 UpdateMenus {OVERRIDE}              Perform menu management tasks
******************************************************************************/
void CBackpropDoc::UpdateMenus()
{
    /* Enable commands handled by your Application class and std. commands    */
    inherited::UpdateMenus();

    /* Acquire MENU                                                           */
    gBartender->EnableCmd(cmdAcquire);
    gBartender->EnableCmd(cmdNewSound);

    /* Train MENU                                                             */
    gBartender->EnableCmd(cmdSetupTrain);
    gBartender->EnableCmd(cmdTestNetwork);
    gBartender->EnableCmd(cmdGradeRate);
    gBartender->EnableCmd(cmdTrainNetwork);
    gBartender->EnableCmd(cmdNetWeights);
    gBartender->EnableCmd(cmdNewWeights);
    gBartender->EnableCmd(cmdFileWeights);

    /* Recognize MENU                                                         */
    gBartender->EnableCmd(cmdRecognize);

    /* Utilities MENU                                                         */
    gBartender->EnableCmd(cmdUtterSpect);
    gBartender->EnableCmd(cmdTimeAmpl);
    gBartender->EnableCmd(cmdSpeechSplice);
    gBartender->EnableCmd(cmdPrintPlot);

    /* Hierarchical MENU grademethod                                          */
    gBartender->EnableCmd(cmdchebychev);
    gBartender->EnableCmd(cmdeuclidean);
    gBartender->EnableCmd(cmdhamming);
    gBartender->EnableCmd(cmdminkowski3);
    gBartender->EnableCmd(cmdminkowski4);
    gBartender->EnableCmd(cmdminkowski5);
    gBartender->EnableCmd(cmdminkowski6);
    gBartender->EnableCmd(cmdminkowski7);
}

/*********************************************************************************
Acquire()
    This method sends a start record message to the IBM unit to begin recording
    speech for 1 seconds and also sends a playback message to  re-synthesize the utterance.
    In turn, the IBM returns the LPC parameters in an array.
*********************************************************************************/
void CBackpropDoc::Acquire()
{
    char        Message1[255],Message2[255];
    short       Where,x,t,dialogID,itemType,itemHit,tempga,tempgb,RecordVers,RecordWrd;
    double      *FloatArray;
    Str63       filename;
    Ptr         dStorage;
    DialogPtr   Recordbox;
    WindowPtr   behind;
    GrafPtr     CurrentPort;
    Handle      tempitemga,tempitemgb,item,CBitem;
    Rect        box;

    /* Set alert box message*/
    strcpy(Message1,"\pThis version of the utterance has already been recorded. You may record over it, but
    the existing recording will be destroyed!");
    strcpy(Message2,"\pThis version of the utterance has not been recorded. After recording, you may then
    select it to playaback.");
    ParamText(Message1,Message2,' ',' ');

    /* Allocate space for LPCData                                                   */
        FloatArray =calloc(NumInNuron, sizeof(double));

    behind=(WindowPtr)(-1L);
    dialogID=DLOGrecord;
    dStorage=NULL;
    Recordbox = GetNewDialog(dialogID,dStorage,behind);
```

```
/* Set  File Name of Dialog, if opened                                    */
if (itsFile !=NULL)
{
     itsFile->GetName(filename);
     GetDItem(Recordbox,3,&itemType,&item,&box);
     SetIText(item,filename);
}

/*   Set Date for the Record box                                          */
     SetDate(Recordbox,4);

/*   Outline Record & Playback and Done buttons                           */
     GetPort(&CurrentPort);
     OutlineButton (Recordbox,1  );
     OutlineButton (Recordbox,25);
     OutlineButton (Recordbox,33);
     SetPort(CurrentPort);

/*   Initial Setting to the first utterance                               */
     tempga=5;
     RecordWrd=0;
     SetRadioButton(tempga, Recordbox, &tempitemga);

/*   Initial Setting to the Excellent version                             */
     tempgb=15;
     RecordVers=0;
     SetRadioButton(tempgb, Recordbox, &tempitemgb);

/*   Set Word names in the radio boxes                                    */
     SetWords(Recordbox,&Wordlabel1,&Wordlabel2,&Wordlabel3,&Wordlabel4,
     &Wordlabel5,&Wordlabel6,&Wordlabel7,&Wordlabel8,&Wordlabel9,&Wordlabel10);

itemHit=0;
while( itemHit !=33 && itemHit !=Cancel)
{
     /* Set the appropriate check boxes indicating pre-recorded words     */
     SetCheckBoxes(Recordbox,RecordWrd,theVersions,NumOfVers);

     ModalDialog(NULL, &itemHit);
```

```
GetDItem(Recordbox,itemHit,&itemType,&item,&box);

/* Set appropriate word selection based on radio button                         */
if (itemHit != tempga && itemHit > 4 && itemHit < (NumOfWords+5))
{
    SetCtlValue((ControlHandle) tempitemga, FALSE);
    tempga=itemHit;          /* Set old item number                              */
    tempitemga=item;         /* Set old handle name                              */
   /*   Record word #  starting at wordnumber=0                                  */
   RecordWrd=itemHit-5;
   SetCtlValue((ControlHandle)item,TRUE);

   /* Alert Message indicating the selection of an existing recorded version     */
   if (theVersions[RecordWrd*NumOfVers+RecordVers])
       CautionAlert(128,NULL);
}

/* Set selected versions radio button                                            */
if (itemHit != tempgb && itemHit > 14 && itemHit < (NumOfVers+15))
{
    SetCtlValue((ControlHandle) tempitemgb,FALSE);
    tempgb=itemHit;          /* Set old item number                              */
    tempitemgb=item;         /* Set old handle name                              */
    /*  Record version; Excellent=0, Unsatisfactory=4                            */
    RecordVers=itemHit-15;
    SetCtlValue((ControlHandle)item,TRUE);

    /* Alert Message indicating the selection of an existing recorded version    */
    if (theVersions[RecordWrd*NumOfVers+RecordVers])
        CautionAlert(128,NULL);
}

/* This playsback whichever word/version combination has been selected
if the playback button was selected and that utterance exists                    */
if (itemHit==25 && theVersions[RecordWrd*NumOfVers+RecordVers])
    PlaybackWord(RecordWrd*NumOfVers+RecordVers);

/* Tells user selected word can't be played back because it hasn't been recorded */
if (itemHit==25 && !theVersions[RecordWrd*NumOfVers+RecordVers])
    CautionAlert(129,NULL);
```

```
            if (itemHit==1)   /* Save Data since user selected Record                              */
            {
                theVersions[RecordWrd*NumOfVers+RecordVers]=TRUE;

                /*LPCData is a big dynamic array. FloatArray inserts the data from the acquired
                 utterance into the proper segment in the array LPCData as given below             */
                GetLPCData(FloatArray);

                Where=NumInNuron*(RecordWrd*NumOfVers+RecordVers);
                for (x=0; x<NumInNuron;x++)
                {
                    LPCData[Where+x]=FloatArray[x];
                    FloatArray[x]=0.0;
                }
            }
        }
        if (itemHit==33) /* Done*/
            DoneRecord=TRUE;
        free(FloatArray);
        DisposDialog(Recordbox);
        itsMainPane->Refresh();
}

/*******************************************************************************
NewSounds()
    This is the new sound acquisition routines to record,playback, and storing of the speech data
    and it also generates the LPCData.
*******************************************************************************/
void CBackpropDoc::NewSounds()        /* Sound test project                                    */
{
    /*char                  AlertText[255];
    unsigned char           *mySndPtr;
    short                   mySndId,myNumChans,mySampSize,volRefNum,myRefNum;
    short                   SndLevel,MeterLevel;
    long                    NumOfBytes,NumOfFrames,myHeaderLen,myBuffSize,myInRefNum;
    Fixed                   mySampRate;
    OSErr                   myErr;
    OSType                  myCreator,myFType,myCompType;
```

```
SPB                 mySPB;
Point               myCorner;
Str255              SoundName;
char                theText[255];
short               dialogID,itemType,itemHit,PlaybackMsg,RecordMsg;
Ptr                 dStorage;
DialogPtr           Soundbox;
WindowPtr           behind;
GrafPtr             CurrentPort;
Handle              item,mySndH;
Rect                box;

strcpy(AlertText,"\pThere has been a problem with opening the sound file. Please click on a file that
exists.");
ParamText(' ', AlertText,' ',' ');

behind=(WindowPtr)(-1L);
dialogID=600;
dStorage=NULL;
Soundbox = GetNewDialog(dialogID,dStorage,behind);

mySndId=30493;        30493 is my test sound
NumOfBytes=0L;
myCreator='cris';
myFType='TEXT';
volRefNum=0;
NumOfFrames=0L;
myBuffSize=22000;
itemHit=0;
PlaybackMsg=3;
RecordMsg=4;

Outline Record & Playback and Done buttons
    GetPort(&CurrentPort);
    OutlineButton (Soundbox,1 );
    OutlineButton (Soundbox,3);
    OutlineButton (Soundbox,4);
    SetPort(CurrentPort);
```

```
do
{
    ModalDialog(NULL, &itemHit);
    GetDItem(Soundbox,itemHit,&itemType,&item,&box);

    if (itemHit==PlaybackMsg)
    {
        myErr=FSOpen(SoundName,volRefNum, &myRefNum);
        if (myErr != 0)
            CautionAlert(129,NULL);    Uses ParamText 1
        else
            myErr=SndStartFilePlay (NULL, myRefNum, 0, myBuffSize, NULL, NULL, NULL,
            FALSE);
        FSClose(myRefNum);
    }
    if(itemHit==RecordMsg)
    {
         Create and open a file with name SoundName
        myErr=Create (SoundName, volRefNum, myCreator, myFType);
        myErr=FSOpen (SoundName, volRefNum, &myRefNum);

        Open the current sound device and get its settings
        myErr=SPBOpenDevice ('', siReadPermission, &myInRefNum);
        myErr=SPBGetDeviceInfo (myInRefNum, siNumberChannels, (Ptr)&myNumChans);
        myErr=SPBGetDeviceInfo (myInRefNum, siSampleRate, (Ptr)&mySampRate);
        myErr=SPBGetDeviceInfo (myInRefNum, siSampleSize, (Ptr)&mySampSize);
        myErr=SPBGetDeviceInfo (myInRefNum, siCompressionType, (Ptr)&myCompType);

        Allocate the buffer space that I intend on storing the sound data in
        mySndPtr= (unsigned char *) calloc (myBuffSize, sizeof(char));
        mySndH=NewHandle((Size) 22000);

         Define the structure of the Sound Parameter Block (SPB)
        mySPB.inRefNum=myInRefNum;
        mySPB.count=myBuffSize;
        mySPB.milliseconds=0L;
        mySPB.bufferLength=myBuffSize;
        mySPB.bufferPtr= (Ptr) mySndPtr;
        mySPB.completionRoutine=NULL;
        mySPB.interruptRoutine = NULL;
```

```
            mySPB.userLong=0L;
            mySPB.error=noErr;
            mySPB.unused1=0L;

            SetPt(&myCorner,100,100);
            myErr=SndRecord( NULL, myCorner, siBestQuality, &mySndH);

             Set up a file sound header and write it into the file with reference myRefNum
            myErr=SetupAIFFHeader(myRefNum, myNumChans, mySampRate, mySampSize,
            myCompType, myBuffSize, NumOfFrames);

            Record the sound
            myErr=SPBRecord(&mySPB, FALSE);

            SetPt(myCorner, 100, 100);
            myErr=SndRecordToFile(NULL, myCorner, siBestQuality, myRefNum);

            Use SPB record to file
            myErr=SPBRecordToFile(myRefNum, &mySPB, FALSE);

            Write the data to the open file named SoundName since the file
            position is set at the end of the header
            myErr=FSWrite(myRefNum, &myBuffSize, (Ptr)mySndPtr);

            GetFPos(myRefNum, &NumOfBytes);

             Close the sound device and the file
            myErr=FSClose(myRefNum);
            myErr=SPBCloseDevice(myInRefNum);

            Release the memory allocated for the sound data
            free(mySndPtr);
            mySndPtr=NULL;
        }

        GetDItem(Soundbox,5,&itemType,&item,&box);
        GetIText(item,SoundName);
}while( itemHit !=1 && itemHit !=2);
```

```
        if (itemHit==PlaybackMsg)
        {
            mySndChan=NULL;
            mySndHandle=GetResource('snd ',mySndId);
            myErr=SndPlay(mySndChan,mySndHandle,kAsync);
        }
        if(itemHit==RecordMsg)
        {
            SetPt(myCorner,50,50);
            mySndH=NULL;
            myErr=SndRecord( NULL, myCorner, siBestQuality, &mySndH);
            mySndHandle=mySndH;
            AddResource(mySndH,'snd ',mySndId+1,SoundName);
        }

    DisposDialog(Soundbox);
    DisposHandle(mySndH);
    itsMainPane->Refresh();*/
}

/*******************************************************************************
SetupTrain()    This method creates a dialog box to initialize the architecture of the
                network and style of data presentation.
*******************************************************************************/
void CBackpropDoc::SetupTrain()
{
    Boolean     Bias,PresType,UpdType,Randwts,IsText;
    char        OldText[255],NewText[255];
    short       dialogID,itemType,itemHit,OldItemHit,itemNo,tempga,tempgb,HidNurons;
    short       InpNurons,OutNurons;
    float       Stopat,Lrnrate,Momntm,UpperLimit,LowerLimit,Value,inc;
    Str255      theText;
    Ptr         dStorage;
    DialogPtr   Bpbox;
    WindowPtr   behind;
    GrafPtr     CurrentPort;
    Handle      tempitemga,tempitemgb,item,NewItem,OldItem;
    ControlHandle       macControl;
    Rect        box;
```

```
dialogID=DLOGbp;
behind=(WindowPtr)(-1L);
dStorage=NULL;
Bpbox = GetNewDialog(dialogID,dStorage,behind);

/* Outline DONE button                                                    */
     GetPort(&CurrentPort);          OutlineButton (Bpbox,1);

/*    Sequential presentation setting                                     */
     if (Presentation)
        tempga=16;
     else
        tempga=15;
     SetRadioButton(tempga, Bpbox, &tempitemga);

/*    Batch Update setting                                                */
     if (UpdateType)
        tempgb=18;
     else
        tempgb=17;
     SetRadioButton(tempgb, Bpbox, &tempitemgb);

/*    Initial Setting of input ,hidden, and output neurons                */
     sprintf (OldText, "%d", NumInNuron);
     GetDItem(Bpbox,8,&itemType,&item,&box);
     OldItem=item;     /* Initialization required                         */
     CtoStr255(OldText, theText);
     SetIText(item,theText);

     sprintf (NewText, "%d", NumHidNuron);
     CtoStr255(NewText, theText);
     GetDItem(Bpbox,9,&itemType,&item,&box);
     SetIText(item,theText);

     sprintf (NewText, "%d", NumOutNuron);
     CtoStr255(NewText, theText);
     GetDItem(Bpbox,10,&itemType,&item,&box);
     SetIText(item,theText);
```

```
/* Set initial values of method                                                  */
Value=0.0;                    itemHit=0;                IsText=FALSE;
UpdType=UpdateType;           PresType=Presentation;    InpNurons=NumInNuron;
Stopat=StopValue;             Randwts=RandomWeights;    HidNurons=NumHidNuron;
Lrnrate=LearnRate;            OutNurons=NumOutNuron;    Momntm=Momentum
Bias=Biases;                  OldItemHit=0;
SetCurrentValues(Bpbox, Bias, Momntm, Stopat, Lrnrate, Randwts);

while( itemHit !=1 && itemHit != Cancel )
{
    ModalDialog(NULL, &itemHit);
    GetDItem(Bpbox,itemHit,&itemType,&item,&box);
    macControl=(ControlHandle) item;

    switch (itemHit)
    {
    case 4:
    case 5:
    case 6:
        if (IsText)
        {
            TextMode(notSrcCopy);     /* Inverse Mode                             */
            switch (itemHit)
            {
            case 4:   /* Up increment                                             */
                      Value=Value+inc;
                      if (Value>UpperLimit)
                                Value=UpperLimit;
                      break;
            case 5:   /* Down Increment                                           */
                      Value=Value-inc;
                      if (Value<LowerLimit)
                                Value=LowerLimit;
                      break;
            case 6:   /* Middle Value                                             */
                                Value=UpperLimit/2.0;
                      break;
            }         /*End Case                                                  */
            switch(itemNo)/* Set the chosen variable to program level             */
            {
```

```
            case 9:    /* Hidden Neurons                                          */
                       HidNurons=(short)Value;
            case 11:/* Learning Rate                                              */
                       Lrnrate=Value;
                       break;
            case 12:/* Momentum                                                   */
                       Momntm=Value;
                       break;
            case 13:/* Stopping Value                                             */
                       Stopat=Value;
                       break;
            }          /*End Case                                                 */

            /* Do update and store as old value now                               */
            sprintf(NewText, "%f", Value);
            CtoStr255(NewText, theText);
            SetIText(NewItem,theText);
            strcpy(OldText,NewText);
            OldItem=NewItem;
        }   /* End if                                                             */
        break;
    case 9:
    case 11:
    case 12:
    case 13:
        switch(itemHit)
        {
        case 9:        /*Hidden Neurons                                           */
            UpperLimit=15.0;
            LowerLimit=2.0;
            inc=1.0;
            Value=(float)HidNurons;
            break;
        case 11:       /*Learning Rate                                            */
            UpperLimit=5.0;
            LowerLimit=0.0;
            inc=0.1;
            Value=Lrnrate;
            break;
    case 12:        /*Momentum                                          */
        UpperLimit=2.0;
        LowerLimit=0.0;
        inc=0.1;
        Value=Momntm;
        break;
    case 13:        /*Stopping Value                                    */
        UpperLimit=1.0;
        LowerLimit=0.02;
        inc=0.02;
        Value=Stopat;
        break;
    }

    /* Read in current value and redisplay highlighted                  */
    TextMode(notSrcCopy);
    NewItem=item;
    IsText=TRUE;
    itemNo=itemHit;
    GetIText(NewItem,theText);
    SetIText(NewItem,theText);
    break;
case 14: /*   Biases=TRUE-> on; Biases=FALSE-> OFF                      */
    SetCtlValue(macControl,(1 - GetCtlValue(macControl)));
    Bias=GetCtlValue(macControl);
    break;
case 15:
case 16: /*   15=Sequential  Presentation 16=Random Presentation        */
    if (itemHit !=tempga)
    {
        SetCtlValue((ControlHandle) tempitemga,FALSE);
        tempga=itemHit;     /* Set old item number                      */
        tempitemga=item;    /* Set old handle name*/
    }
    SetCtlValue(macControl,TRUE);
    /* Sequential Presentation=FALSE  Random Presentation=TRUE          */
    PresType=itemHit-15;
    break;
case 17:
        case 18: /*   17=Single update 18=Batch update                          */
            if (itemHit !=tempgb)
            {
                SetCtlValue((ControlHandle) tempitemgb,FALSE);
                tempgb=itemHit;/* Set old item number                        */
                tempitemgb=item;/* Set old handle name                       */
            }
            SetCtlValue(macControl,TRUE);
            /*Single update=FALSE   Batch update=TRUE                           */
            UpdType=itemHit-17;
            break;
        case 26:/* Random Weight selection                                      */
            SetCtlValue(macControl,(1 - GetCtlValue(macControl)));
            Randwts=GetCtlValue(macControl);
            break;
        }  /*   End Case                                                        */

        TextMode(srcCopy);/* Normal Mode                                        */
        if ((IsText) && (itemType!=32) &&(OldItemHit!=itemHit))
        {
            /* part 1 -> a text item is currently active                        */
            /* part 2 -> don't want to un-highlight if an icon incrementer      */
            /* part 3 -> different item from previous time                      */

            IsText=FALSE;/* set flag false -> no more text fields               */
            CtoStr255(OldText,theText);
            SetIText(OldItem,theText);/* un-highlight text field                */

            if (itemType == 8)              /* current item  is also at text field  */
            {
                strcpy(OldText,NewText);
                OldItem = NewItem;
                IsText=TRUE;
            }
        }
        OldItemHit=itemHit;
    }   /* End While                                                            */
    SetPort((GrafPtr)CurrentPort);

    if (itemHit==1)/* place dialog variables into program variables            */
    {
        UpdateType=UpdType;        Presentation=PresType;        Biases=Bias;
        Momentum=Momntm;           LearnRate=Lrnrate;            StopValue=Stopat;
        NumHidNuron=HidNurons;     RandomWeights=Randwts;        NumInNuron=InpNurons;
        NumOutNuron=OutNurons;
    }
    DisposDialog(Bpbox);
    itsMainPane->Refresh();
}

/*****************************************************************************
Set Weights - Creates new weight values for existing network
*****************************************************************************/
void CBackpropDoc::SetNetWts(Boolean SetWts)
{
    char        Message1[255],Message2[255];
    short       Number, AlertAction=2;

    /* Set alert box message            */
    strcpy(Message1,"\pThe existing weight structure will be destroyed. Please save before proceeding
    and/or SaveAs under a different file name.");
    strcpy(Message2,"\pThere is no data file to read weight structure from. Please choose set new weights
    instead or load an existing data file.");
    ParamText(Message1, Message2,' ',' ');

    if (SetWts && itsFile!=NULL)     /* File Weights - Check to see if a file has been opened first     */
    {
        /* Warning: Network weight data exists. This overwrites existing data => IHWt !=NULL        */
        if (IHWt[0]!=0)
        {
            AlertAction=CautionAlert(130,NULL);
             itsMainPane->Refresh();
        }

        if (AlertAction==2)          /* Overwrite                                                      */
        {
            /* Read the weights and biases from disk (pointer at 358+ 1500*double)                    */
            ((CDataFile *)itsFile)->SetMark((long) (358+1500*sizeof(double)), fsFromStart);
            ((CDataFile *)itsFile)->ReadSome((Ptr)IHWt, (long)(sizeof(double)*NumInNuron
            *NumHidNuron));
            ((CDataFile *)itsFile)->ReadSome((Ptr)HOWt, (long)(sizeof(double)*NumOutNuron*
            NumHidNuron));
            ((CDataFile *)itsFile)->ReadSome((Ptr)HiddenBias, (long)(sizeof(double)*NumHidNuron));
            ((CDataFile *)itsFile)->ReadSome((Ptr)OutputBias, (long)(sizeof(double)*NumOutNuron));
        }
    }

    /*  itsFile==NULL -> They chose to read from a file that doesn't exist.                    */
    if (SetWts && itsFile==NULL)
    {
        StopAlert (129,NULL);
        itsMainPane->Refresh();
    }

    /*  New Weights     or Reinitialize weights not having loaded from a file                  */
    if (!SetWts)
    {
        /* Warning: Network weight structure exists.Wants to reinitialize weights but has not saved the
        existing data file yet.                                                                */
        if(IHWt[0]!=0)
        {
            AlertAction=CautionAlert(130,NULL);
            itsMainPane->Refresh();
        }

        if (AlertAction==2)
        {
            SetWeights(IHWt, NumInNuron*NumHidNuron, RandomWeights);
            SetWeights(HOWt, NumHidNuron*NumOutNuron, RandomWeights);
            SetWeights(HiddenBias, NumHidNuron, RandomWeights);
            SetWeights(OutputBias, NumOutNuron, RandomWeights);
        }
    }
}

/*************************************************************************
TrainNetwork - Does it!
*************************************************************************/
void CBackpropDoc::TrainNetwork()
```

```
{
    Boolean      *PattNumArray, DoneTraining;
    short        x,y,z,PattNum,TrialNumber, Start, Finish, Count, itemType, itemHit;
    long         theSeed;
    double       *IHWtChange, *HOWtChange, *HiddenNurons, *OutputNurons, *TrainingPatt;
    double       *DeltaHidden, *DeltaOutput, WeightedSum, TotalError, OldTotalError;
    double       *BHOWtChge, *BIHWtChge;
    Ptr          dStorage;
    DialogPtr    Bpupdate;
    WindowPtr    behind;
    GrafPtr      CurrentPort;
    Handle       item;
    EventRecord  Event;
    Rect         box;

    behind=(WindowPtr)(-1L);
    dStorage=NULL;
    Bpupdate = GetNewDialog(DLOGbpupdate,dStorage,behind);
    DrawDialog(Bpupdate);

    /*   Outline Exit  buttons*/
         GetPort(&CurrentPort);
         OutlineButton (Bpupdate,1 );
         SetPort(CurrentPort);

    DoneTraining=FALSE;
    /*   Array  Allocation and Initialization Arrays are created in a 1-D form but accessed in the traditional
    2D format using the conversion formula Array(row,col) = ColDim*Row+Col.*/

         HiddenNurons    =calloc(NumHidNuron,                    sizeof(double));
         OutputNurons    =calloc(NumOutNuron,                    sizeof(double));
         TrainingPatt    =calloc(NumOfPatt*NumOutNuron,          sizeof(double));
         DeltaHidden     =calloc(NumHidNuron,                    sizeof(double));
         DeltaOutput     =calloc(NumOutNuron,                    sizeof(double));
         HOWtChange      =calloc(NumHidNuron*NumOutNuron,        sizeof(double));
         IHWtChange      =calloc(NumInNuron*NumHidNuron,         sizeof(double));

         SetWeights(IHWtChange, NumInNuron*NumHidNuron, RandomWeights);
         SetWeights(HOWtChange, NumHidNuron*NumOutNuron, RandomWeights);
```

```
        if (UpdateType)
        {
            BHOWtChge=calloc(NumHidNuron*NumOutNuron,      sizeof(double));
            BIHWtChge=calloc(NumInNuron*NumHidNuron,       sizeof(double));
        }
/* Set training patterns                                                    */
       SetTrainingPat(TrainingPatt, NumOutNuron);

/* Network method initialization of paramters                               */
Start=0;
TrialNumber=0;
itemHit=0;
OldTotalError=0;
TotalError=0.0;
Count=0;

/* Set the form of network UPDATING:
          UpdateType=TRUE  ---> Batch pattern update
          UpdateType=FALSE ---> Single pattern update                       */
if (UpdateType)
      Finish=NumOfPatt;
else
      Finish=1;

/* PattNumArray tells the program which patterns have been presented by placing a one
in the array position after presenting it. The program will continue to randomly look for
a pattern number that does not have a one in it until it has presented all patterns.     */
if (Presentation)
      PattNumArray=calloc(NumOfPatt,sizeof(Boolean));

do
{
      /* Set the form of network pattern PRESENTATION:
               Presentation=TRUE  ---> Random pattern presentation
               Presentation=FALSE ---> Sequnetial pattern presentation         */
      if (Presentation)
      {
         GetDateTime(&theSeed);
         srand(theSeed);
      }
```

```
for (z=Start; z<Finish; z++)
{
   if (Presentation)
       PattNum=GetPatterns(PattNumArray, NumOfPatt );
   else
       PattNum=z;

   /*  Calculates the forward pass through the neural network                    */
   CalcForwardPass(LPCData, HiddenNurons, OutputNurons, HiddenBias, OutputBias,
                IHWt, HOWt, NumInNuron, NumHidNuron,NumOutNuron, PattNum);

   /*  Calculate the TotalError based on 'Error measure' selection*/
   CalcTotError(TrainingPatt, OutputNurons, PattNum,Grade, NumOutNuron,Power,
   &TotalError);

   /*  Calculate Error vector for output Nurons                                  */
   for (x=0; x<NumOutNuron; x++)
       DeltaOutput[x]=OutputNurons[x]*(1.0-OutputNurons[x])*
                 (TrainingPatt[NumOutNuron*PattNum+x]-OutputNurons[x]);

   if (!UpdateType)/* Single update                                              */
   {
       /*Calculate Weight changes between OuputNurons and HiddenNurons           */
       for(x=0; x<NumOutNuron; x++)
       {
               for (y=0; y<NumHidNuron; y++)
               {
               HOWtChange[NumHidNuron*x+y]=LearnRate*DeltaOutput[x]*
               HiddenNurons[y]+Momentum*HOWtChange[NumHidNuron*x+y];
               HOWt[NumHidNuron*x+y]+=HOWtChange[NumHidNuron*x+y];
               if (Biases)
                      OutputBias[x]+=HOWtChange[NumHidNuron*x+y];
               }
       }

       /*Calculate Error vector for HiddenNurons                                 */
       WeightedSum=0.0;
       for (x=0; x<NumHidNuron; x++)
       {
            for (y=0; y<NumOutNuron; y++)
                    WeightedSum+=DeltaOutput[y]*HOWt[NumHidNuron*y+x];
            DeltaHidden[x]=HiddenNurons[x]*(1.0-HiddenNurons[x])*WeightedSum;
            WeightedSum=0.0;
    }

    /*Calculate Weight change between Hidden layer and Input layer          */
    for (x=0; x<NumHidNuron; x++)
    {
            for (y=0; y<NumInNuron; y++)
            {
            IHWtChange[NumInNuron*x+y]=LearnRate*DeltaHidden[x]*
            LPCData[NumInNuron*PattNum+y]+Momentum*IHWtChange
                                                        [NumInNuron*x +y];
                    IHWt[NumInNuron*x+y]+=IHWtChange[NumInNuron*x+y];
                    if (Biases)
                         HiddenBias[x]+=IHWtChange[NumInNuron*x+y];

            }
    }
}
else /* Batch update type calculations*/
{
/*  Calculate Weight changes between OuputNurons and HiddenNurons          */
    for(x=0; x<NumOutNuron; x++)
    {
       for (y=0; y<NumHidNuron; y++)
       BHOWtChge[NumHidNuron*x+y]+=LearnRate*DeltaOutput[x]*HiddenNurons[y]
                              +Momentum*HOWtChange[NumHidNuron*x+y];

    }

    /*Calculate Error vector for HiddenNurons                                */
    WeightedSum=0.0;
    for (x=0; x<NumHidNuron; x++)
    {
            for (y=0; y<NumOutNuron; y++)
                    WeightedSum+=DeltaOutput[y]*HOWt[NumHidNuron*y+x];
            DeltaHidden[x]=HiddenNurons[x]*(1.0-HiddenNurons[x])*WeightedSum;
            WeightedSum=0.0;
    }
```

```
            /*Calculate Weight change between Hidden layer and Input layer                */
            for (x=0; x<NumHidNuron; x++)
            {
                    for (y=0; y<NumInNuron; y++)
            BIHWtChge[NumInNuron*x+y]+=LearnRate*DeltaHidden[x]*
            LPCData[NumInNuron*PattNum+y]+Momentum*IHWtChange[NumInNuron*x+y];
            }
        }
        Count+=1;
        NetworkUpdate(Bpupdate, TrialNumber, PattNum+1, LearnRate, OldTotalError);
    }/* Last Pattern                                                                  */

    /*  If batch update, then change all the weights here                             */
    if (UpdateType)
    {
        /* Change weights between Output layer and Hidden layer                       */
        for(x=0; x<NumOutNuron; x++)
        {
            for (y=0; y<NumHidNuron; y++)
            {
                    HOWt[NumHidNuron*x+y]+=BHOWtChge[NumHidNuron*x+y];
                    HOWtChange[NumHidNuron*x+y]=BHOWtChge[NumHidNuron*x+y]/
                                                              (double)NumOfPatt;
                    if (Biases)
                            OutputBias[x]+=BHOWtChge[NumHidNuron*x+y];
                    BHOWtChge[NumHidNuron*x+y]=0.0;
            }
        }

        /* Change weights between Input layer and Hidden layers                       */
        for (x=0; x<NumHidNuron; x++)
        {
            for (y=0; y<NumInNuron; y++)
            {
            IHWt[NumInNuron*x+y]+=BIHWtChge[NumInNuron*x+y];
            IHWtChange[NumInNuron*x+y]=BIHWtChge[NumInNuron*x+y]/(double)NumOfPatt;
                    if (Biases)
                            HiddenBias[x]+=BIHWtChge[NumInNuron*x+y];
                    BIHWtChge[NumInNuron*x+y]=0.0;
            }
        }
    }/* end if loop                                                  */

    /* Exiting Routine                                               */
    GetNextEvent(mDownMask, &Event);
    if(IsDialogEvent(&Event))
    {
        DialogSelect(&Event, &Bpupdate, &itemHit);
        if (itemHit==1)/*Exit                                        */
        {
            DoneTraining=TRUE;/* Exit while loop                     */
            DoneNetwork=FALSE;/* Exited prematurely                  */
            dirty=TRUE;
        }
    }

    /* Completed one pattern set. Check Error and adjust Parameters  */
    if (TotalError<(double)StopValue && OldTotalError<(double) StopValue &&
    Count==NumOfPatt)
    {
        DoneTraining=TRUE;     /* Exit while loop                    */
        DoneNetwork=TRUE;      /* Exit flag set to return to main menubar */
    }

    if(Count==NumOfPatt)
    {
        if(TotalError>OldTotalError)     /*Increase in Error from previous time */
        {
        LearnRate/=1.2;
        if (LearnRate<LowerLearnThresh)
            LearnRate=LowerLearnThresh;
        }
        else /*Decrease in Error from previous time                  */
        {
        LearnRate*=1.01;
        if (LearnRate>UpperLearnThresh)
            LearnRate=UpperLearnThresh;
        }
        /* Reinitialize array used in generating random pattern numbers                */
        OldTotalError=TotalError;
        TrialNumber+=1;
        Count=0;
        TotalError=0.0;
        for (x=0; x<NumOfPatt; x++)
            PattNumArray[x]=0;
    }

    if (!UpdateType)/* SingleUpdate                                                     */
    {
        Start+=1;
        if(Start==NumOfPatt)
            Start=0;
        Finish=Start+1;
    }

}while(!DoneTraining);/*EndDo                                                           */

if(Presentation)
    free(PattNumArray);

if (UpdateType)
{
    free(BIHWtChge);
    free(BHOWtChge);
}

/*  De-allocation of Array memory                                                       */
    free(IHWtChange);
    free(HOWtChange);
    free(DeltaOutput);
    free(DeltaHidden);
    free(TrainingPatt);
    free(OutputNurons);
    free(HiddenNurons);

DisposDialog(Bpupdate);/* Clear dialog box                                              */
itsMainPane->Refresh();
```

```
}

/********************************************************************************
TestNetwork()    This method tests a backpropagation neural network
********************************************************************************/
void CBackpropDoc::TestNetwork()
{
    short       x,dialogID,itemType,itemHit,tempga,tempgb,TestWrd,TestVers;
    double      *HiddenNurons,*OutputNurons;
    Ptr         dStorage;
    DialogPtr   Testbox;
    WindowPtr   behind;
    GrafPtr     CurrentPort;
    Handle      tempitemga,tempitemgb,item;
    Rect        box;

    behind=(WindowPtr)(-1L);
    dialogID=DLOGtest;
    dStorage=NULL;

    Testbox = GetNewDialog(dialogID,dStorage,behind);

    /* Outline Record and Playback buttons                                      */
        GetPort(&CurrentPort);
        OutlineButton(Testbox,1);
        SetPort(CurrentPort);

    /*  Initial Setting of Radio Group1                                         */
        tempga=5;
        TestWrd=0;
        SetRadioButton(tempga, Testbox, &tempitemga);

    /*  Initial Setting of Radio Group2                                         */
        tempgb=15;
        TestVers=0;
        SetRadioButton(tempgb, Testbox, &tempitemgb);

    /*  Set Word names in the radio boxes                                       */
        SetWords(Testbox, &Wordlabel1, &Wordlabel2, &Wordlabel3,&Wordlabel4, &Wordlabel5,
            SetCtlValue((ControlHandle) tempitemgb, FALSE);
            tempgb=itemHit;          /* Set old item number                          */
            tempitemgb=item;         /* Set old handle name                          */
            /* Test version; Excellent=0, Unsatisfactory=4                          */
            TestVers=itemHit-15;
            SetCtlValue((ControlHandle)item, TRUE);
        }
    }
    DisposDialog(Testbox);
    itsMainPane->Refresh();
}

/*****************************************************************************
Recognize()
    This method perfroms the recognition. It is done by first selecting the word and version.
    Next, playback is selected which playsback the word. We simply call "Playback()" to
    do this for us. Finally the user selects record. The utterance is redorded and the identity
    is displayed in the dialog box with the associated version.
*****************************************************************************/
void CBackpropDoc::Recognize()
{
    short       x,y,dialogID,itemType,itemHit,tempga,tempgb,PlaybackWrd,PlaybackVers;
    double      *InputNurons,*HiddenNurons,*OutputNurons;
    Ptr         dStorage;
    DialogPtr   Recognizebox;
    WindowPtr   behind;
    GrafPtr     CurrentPort;
    Handle      tempitemga,tempitemgb,item,CBitem;
    Rect        box;

    behind=(WindowPtr)(-1L);
    dialogID=DLOGrecognize;
    dStorage=NULL;
    Recognizebox = GetNewDialog(dialogID, dStorage, behind);

    /* Outline Record and Playback buttons                                      */
        GetPort(&CurrentPort);
        OutlineButton(Recognizebox, 1);
        OutlineButton(Recognizebox, 2);
        SetPort(CurrentPort);

    /*  Initial Setting of Radio Group1                                          */
        tempga=5;
        PlaybackWrd=0;
        SetRadioButton(tempga, Recognizebox, &tempitemga);

    /*  Initial Setting of Radio Group2                                          */
        tempgb=15;
        PlaybackVers=0;
        SetRadioButton(tempgb, Recognizebox, &tempitemgb);

    /*  Set Word names in the radio boxes                                        */
        SetWords(Recognizebox, &Wordlabel1, &Wordlabel2, &Wordlabel3, &Wordlabel4,
        &Wordlabel5, &Wordlabel6, &Wordlabel7, &Wordlabel8, &Wordlabel9, &Wordlabel10);

    /* Allocate room for the arrays used to calculate the output from the trained neural network  */
            InputNurons =calloc(NumInNuron,            sizeof(double));
            HiddenNurons=calloc(NumHidNuron,           sizeof(double));
            OutputNurons=calloc(NumOutNuron,           sizeof(double));

    itemHit=0;
    while( itemHit !=20)   /* Exit=20                                                */
    {
        ModalDialog(NULL, &itemHit);
        GetDItem(Recognizebox,itemHit,&itemType,&item,&box);

        /* Playback button is hit                                                */
        if (itemHit==1)
            PlaybackWord(PlaybackWrd*NumOfVers+PlaybackVers);

        /* Record the word and identify it                                       */
        if (itemHit==2 && IHWt[0]!=0)
        {
            GetLPCData(InputNurons);
            /* Calculates the forward pass through the neural network            */
            CalcForwardPass(InputNurons, HiddenNurons, OutputNurons, HiddenBias,
            OutputBias, IHWt, HOWt, NumInNuron, NumHidNuron, NumOutNuron, 0);

            /* Calculate and display recognition results in the appropriate space */
            CalcForwardPass(InputNurons, HiddenNurons, OutputNurons, HiddenBias,
            OutputBias, IHWt, HOWt, NumInNuron, NumHidNuron, NumOutNuron, 0);
            /* Calculate and display recognition results in the appropriate space          */
            DisplayResults(Recognizebox, OutputNurons, NumOfPatt, NumOfVers);
         }
         /* Set appropriate word selection based on radio button                             */
         if (itemHit !=tempga && itemHit > 4 && itemHit < 15)
         {
            SetCtlValue((ControlHandle)tempitemga, FALSE);
            tempga=itemHit;          /* Set old item number                                 */
            tempitemga=item;         /* Set old handle name                                 */
           /*   Playback word#  starting at wordnumber=0                                    */
           PlaybackWrd=itemHit-5;
           SetCtlValue((ControlHandle)item, TRUE);
         }
         /* Set selected versions radio button                                               */
         if (itemHit !=tempgb && itemHit >14 && itemHit < 20)
         {
           SetCtlValue((ControlHandle)tempitemgb, FALSE);
           tempgb=itemHit;               /* Set old item number                             */
           tempitemgb=item;              /* Set old handle name                             */
           /*  Playback version; Excellent=0, Unsatisfactory=4                              */
           PlaybackVers=itemHit-15;
           SetCtlValue((ControlHandle)item, TRUE);
         }
      }
      free(OutputNurons);
      free(HiddenNurons);
      free(InputNurons);
      DisposDialog(Recognizebox);
      itsMainPane->Refresh();
}
/*************************************************************************
Speechsplice()           This method permits the use of speech splicing on the IBM.
*************************************************************************/
void      CBackpropDoc::Speechsplice()
{
   /* Code to initiate speech splicing                                                  */
}
```

```
/*********************************************************************************
Spectplot()        This method plots a frquency -time spectrograph of a utterance in memory
*********************************************************************************/
void CBackpropDoc::Spectplot()
{
    /* Nothing yet*/
}

/*********************************************************************************
Amplplot()         This plots the time-amplitude plot of the speech utterance.
*********************************************************************************/
void CBackpropDoc::Amplplot()
{
    /* Code to plot a time-amplitude plot of the speech utterance                    */
}
```

```
/******************************************************************************
 CBackpropPane.h                           Interface for the BackpropPane Class
 ******************************************************************************/
#define _H_CBackpropPane

#include <CPanorama.h>              /* Interface for its superclass          */

struct CBackpropPane : CPanorama {  /* Class Declaration                     */
                                    /** Instance Variables                   **/
                                    /** Instance Methods                     **/
                                    /** Contruction/Destruction              **/

    void    IBackpropPane(CView *anEnclosure, CBureaucrat *aSupervisor, short aWidth, short
        aHeight,short aHEncl, short aVEncl,SizingOption aHSizing, SizingOption aVSizing);

                                    /** Drawing                              **/
    void        Draw(Rect           *area);

                                    /** Mouse                                **/
    void        DoClick(Point hitPt, short modifierKeys, long when);
    Boolean     HitSamePart(Point pointA, Point pointB);

                                    /** Cursor **/
    void        AdjustCursor(Point where, RgnHandle mouseRgn);

                                    /** Scrolling **/
    void        ScrollToSelection(void);
};

/******************************************************************************
 CBackpropPane.c
    The BackpropPane Class          SUPERCLASS = CPanorama
                        Written by Chris D. Love / June 1990-January 3,1991
 ******************************************************************************/

#include <Global.h>
#include "CBackpropPane.h"

/*** Global Variables ***/
```

```
/*** Class Constants ***/

/*****************************************************************************
IBackpropPane                      Initialize a BackpropPane object
*****************************************************************************/
void CBackpropPane::IBackpropPane(
    CView               *anEnclosure,
    CBureaucrat         *aSupervisor,
    short               aWidth,
    short               aHeight,
    short               aHEncl,
    short               aVEncl,
    SizingOption        aHSizing,
    SizingOption        aVSizing)
{
    CPanorama::IPanorama(anEnclosure, aSupervisor, aWidth, aHeight, aHEncl, aVEncl, aHSizing,
                                                                            aVSizing);
}

/*****************************************************************************
Draw {OVERRIDE}         Draw the contents of the Pane. The area parameter, specified in
        the pane's Frame coordinates, indicates the portion of the Pane which needs to be drawn.
*****************************************************************************/
void CBackpropPane::Draw(
    Rect     *area)
{
    MoveTo(frame.left, frame.top);      /* XXX Draws an X from corner to              */
    LineTo(frame.right, frame.bottom); /*  corner in the Panorama                     */
    MoveTo(frame.right, frame.top);
    LineTo(frame.left, frame.bottom);
}

/*****************************************************************************
DoClick {OVERRIDE}     Respond to a mouse click within the view.
*****************************************************************************/
void CBackpropPane::DoClick(
    Point    hitPt,
    short    modifierKeys,
    long     when)
```

```
{
}

/*****************************************************************************
 HitSamePart {OVERRIDE}              Check whether two points hit the same part of the view.
*****************************************************************************/
Boolean     CBackpropPane::HitSamePart(
    Point   pointA,
    Point   pointB)
{
    return(TRUE);
}

/*****************************************************************************
 AdjustCursor {OVERRIDE}          Adjust the shape of the cursor according the position of the mouse.
*****************************************************************************/
void CBackpropPane::AdjustCursor(
    Point     where,        /* Mouse location in Window coords                    */
    RgnHandle    mouseRgn)          /* Region containing the mouse                */
{
    SetCursor(&arrow);    /* Use the standard arrow cursor                        */
}

/*****************************************************************************
 ScrollToSelection {OVERRIDE}Scroll a Panorama so that the current selection is visible withinthe frame.
*****************************************************************************/
void CBackpropPane::ScrollToSelection()
{
}
```

```
/********************************************************************************/
                              MyFunctions2.c
                    FUNCTIONS USED BY VARIOUS PROGRAMS
                 Written by Chris D. Love / June 1990-January 3,1991
/********************************************************************************/
#include  <stdlib.h>
#include  <stdio.h>
#include  <math.h>
#include  <string.h>
#include  <ctype.h>
                              /*    Function Prototypes                    */
void    OutlineButton    (DialogPtr DialogName, short WhichOne);
void    SetRadioButton   (short WhichItem, DialogPtr  DialogName, Handle *olditem);
void    SetDate          (DialogPtr DialogName, short    DateItemNo);
void    SetWeights       (double *theArray, short Length, Boolean Randomize);
short   GetPatterns      (Boolean     *anArray, short NumPatterns);
void    SetWords         (DialogPtr DialogName, Str255 *Word1, Str255 *Word2,Str255 *Word3,
                                  Str255 *Word4, Str255 *Word5, Str255 *Word6, Str255 *Word7,
                                  Str255 *Word8, Str255 *Word9, Str255 *Word10);
void    GetWords         (DialogPtr  DialogName, Str255 *PatName, Str255 *TherName,
                                  Str255 *Word1, Str255 *Word2, Str255 *Word3, Str255 *Word4,
                                  Str255 *Word5, Str255 *Word6, Str255 *Word7,
                                  Str255 *Word8, Str255 *Word9, Str255 *Word10);
void    NetworkUpdate    (DialogPtr DialogName, short TrialNum, short thePattern, float LRate,
                                  double TError);
void    PtoStr255        (char  *srcString, Str255  destString);
void    CtoStr255        (char  *srcString, Str255  destString);
void    CtoStr255        (char  *srcString, Str255 destString);
void    CalcTotError     (double *TPatt, double *OutNurons, short PatternNum, short theGrade,
                                  short NumONurons, short thePower, double *TError);
void    SetCheckBoxes    (DialogPtr DialogName, short WhichWord, Boolean *ArrayName, short
                                  NumVers);
void    AscToFloat       (char *AscFloat, double *FloatptArray);
void    DisplayResults   (DialogPtr DialogName, double *theArray, short NumPatt, short NumVers);
void    SetTrainingPat   (double *TPatt, short NumberONurons);
void    CalcForwardPass  (double *InputArray, double *HiddenArray, double *OutputArray,
                                  double *HBias, double *OBias, double *WIH, double *WHO,
                                  short INurons, short HNurons, short ONurons, short PatNumber);
void SetCurrentValues    (DialogPtr DialogName,Boolean Bias, float Momtum,float StopVal,
                                    float LearnRt,Boolean RandomWts);

                        /*          FUNCTIONS                              */
/***************************************************************************/
void OutlineButton (DialogPtr  DialogName,short WhichOne)
{
    short itemType;
    Handle   item;
    Rect     box;

    GetDItem(DialogName,WhichOne,&itemType,&item,&box);
    SetPort((GrafPtr)DialogName);
    PenSize(3,3);
    InsetRect(&box,-4,-4);
    FrameRoundRect(&box,16,16);
    PenSize(1,1);
}

/***************************************************************************/
void SetRadioButton (short WhichItem, DialogPtr DialogName, Handle *olditem)
{
    short        itemType;
    Handle       item;
    Rect         box;

    GetDItem(DialogName,WhichItem,&itemType,&item,&box);
    SetCtlValue((ControlHandle) item,TRUE);
    *olditem=item;
}

/***************************************************************************/
void SetDate(DialogPtr DialogName, short DateItemNo)
{
    DateTimeRec          Date;
    char                 DATE[16], month[3], day[3];
    short                itemType;
    long                 secs;
    Str255               theText;
    Handle               item;
    Rect                    box;

    GetDateTime (&secs);
    Secs2Date(secs , &Date);
    sprintf(DATE,"%d", Date.year);
    sprintf(month,"%d", Date.month);
    sprintf(day,"%d", Date.day);
    strcat(DATE, month);
    strcat(DATE, day);
    CtoStr255(DATE, theText);
    GetDItem(DialogName, DateItemNo, &itemType, &item, &box);/* Datebox=4                */
    SetIText(item,theText);
}

/***************************************************************************/
void SetWeights(double *theArray, short Length, Boolean Randomize)
{   /* Range -1.0 -> +1.0                                                    */
    short x;

    for(x=0;x<Length;x++)
    {
        if (Randomize)    /* Random Value                                    */
            theArray[x]=((double)rand()/(double)RAND_MAX)*2.0-1.0;
        else
            theArray[x]=0.1;          /* Fixed Value                         */
    }
}

/***************************************************************************/
short GetPatterns(Boolean *anArray, short NumPatterns)
{
    short RandPattNum;

    do
    {
        RandPattNum=(short)((float)NumPatterns*rand()/(float)RAND_MAX);
    }while(anArray[RandPattNum]);

    anArray[RandPattNum]=TRUE;
    return(RandPattNum);
```

```
}

/*****************************************************************************/
void SetWords (DialogPtr DialogName, Str255 *Word1, Str255 *Word2, Str255 *Word3,
                        Str255 *Word4, Str255 *Word5, Str255 *Word6, Str255 *Word7,
                        Str255 *Word8, Str255 *Word9, Str255 *Word10)
{
    short       itemType;
    Handle      item;
    Rect        box;

    GetDItem(DialogName,5,&itemType,&item,&box);    SetCTitle((ControlHandle)item,Word1);
    GetDItem(DialogName,6,&itemType,&item,&box);    SetCTitle((ControlHandle)item,Word2);
    GetDItem(DialogName,7,&itemType,&item,&box);    SetCTitle((ControlHandle)item,Word3);
    GetDItem(DialogName,8,&itemType,&item,&box);    SetCTitle((ControlHandle)item,Word4);
    GetDItem(DialogName,9,&itemType,&item,&box);    SetCTitle((ControlHandle)item,Word5);
    GetDItem(DialogName,10,&itemType,&item,&box);   SetCTitle((ControlHandle)item,Word6);
    GetDItem(DialogName,11,&itemType,&item,&box);   SetCTitle((ControlHandle)item,Word7);
    GetDItem(DialogName,12,&itemType,&item,&box);   SetCTitle((ControlHandle)item,Word8);
    GetDItem(DialogName,13,&itemType,&item,&box);   SetCTitle((ControlHandle)item,Word9);
    GetDItem(DialogName,14,&itemType,&item,&box);   SetCTitle((ControlHandle)item,Word10);
}

/*****************************************************************************/
void GetWords(DialogPtr DialogName, Str255 *PatName, Str255 *TherName, Str255 *Word1,
            Str255 *Word2, Str255 *Word3, Str255 *Word4, Str255 *Word5, Str255 *Word6,
            Str255 *Word7, Str255 *Word8, Str255 *Word9, Str255 *Word10)
{
    short       itemType;
    Handle      item;
    Rect        box;

                    /*Patients Name                                     */
    GetDItem(DialogName,15,&itemType,&item,&box);   GetIText(item,PatName);
                    /*Therapists Name                                   */
    GetDItem(DialogName,16,&itemType,&item,&box);   GetIText(item,TherName);

                    /* All the utterances                               */
    GetDItem(DialogName,5,&itemType,&item,&box);    GetIText(item,Word1);
    GetDItem(DialogName,6,&itemType,&item,&box);     GetIText(item,Word2);
    GetDItem(DialogName,7,&itemType,&item,&box);     GetIText(item,Word3);
    GetDItem(DialogName,8,&itemType,&item,&box);     GetIText(item,Word4);
    GetDItem(DialogName,9,&itemType,&item,&box);     GetIText(item,Word5);
    GetDItem(DialogName,10,&itemType,&item,&box);    GetIText(item,Word6);
    GetDItem(DialogName,11,&itemType,&item,&box);    GetIText(item,Word7);
    GetDItem(DialogName,12,&itemType,&item,&box);    GetIText(item,Word8);
    GetDItem(DialogName,13,&itemType,&item,&box);    GetIText(item,Word9);
    GetDItem(DialogName,14,&itemType,&item,&box);    GetIText(item,Word10);
}

/*************************************************************************/
void NetworkUpdate(DialogPtr DialogName, short TrialNum, short thePattern, float LRate, double TError)
{
    char        theText[255];
    short       itemType;
    Str255      NewText;
    Handle      item;
    Rect        box;

    /* Sets trial in Dialog box                                              */
    sprintf (theText, "%i", TrialNum);
    CtoStr255(theText,NewText);
    GetDItem(DialogName,3,&itemType,&item,&box);
    SetIText(item,NewText);

    /* Sets current pattern in Dialog box                                    */
    sprintf (theText, "%i", thePattern);
    CtoStr255(theText,NewText);
    GetDItem(DialogName,4,&itemType,&item,&box);SetIText(item,NewText);

    /* Sets learning rate in Dialog box                                      */
    sprintf (theText, "%f", LRate);
    CtoStr255(theText,NewText);
    GetDItem(DialogName,5,&itemType,&item,&box);
    SetIText(item,NewText);

    /* Sets network error in Dialog box                                      */
    sprintf (theText, "%f", TError);
    CtoStr255(theText,NewText);
    GetDItem(DialogName,6,&itemType,&item,&box);
    SetIText(item,NewText);
}
/***************************************************************************/
void PtoStr255(char *srcString, Str255 destString)
{
    BlockMove(srcString, destString, srcString[0] + 1L);
}

/***************************************************************************/
void CtoStr255(char *srcString, Str255 destString)
{
    destString[0] = (char) strlen(srcString);
    BlockMove(srcString, &destString[1], destString[0]);
}

/*****************************************************************************
 Str255toC   Convert Str255 string type to C string type
*****************************************************************************/
void Str255toC(Str255       srcString, char *destString)
{
    BlockMove(srcString + 1, destString, srcString[0]);
    destString[srcString[0]] = '\0';
}

/***************************************************************************/
void CalcTotError(double *TPatt, double *OutNurons, short PatternNum,short theGrade,
                  short NumONurons, short thePower,double *TError)
{
    short    x;
    double   PattError,theError;

/*  Calculation of Total Error  based on metric provided in Grade                  */
    PattError=0.0;
    theError=0.0;

    switch(theGrade)
    {
    case 1:  /* Chebychev */
```

```
            for (x=0; x<NumONurons; x++)
            {
                PattError=fabs(TPatt[NumONurons*PatternNum+x]-OutNurons[x]);
                if (PattError>theError)     /* Choose biggest pattern error                    */
                    theError=PattError;
            }
            *TError+=theError;
            break;
        case 2:  /* Euclidean                                                                  */
            for (x=0; x<NumONurons; x++)
                PattError+=pow(fabs(TPatt[NumONurons*PatternNum+x]-OutNurons[x]),(double) 2.0);
            *TError+=0.5*PattError;
            break;
        case 3:  /* Hamming                                                                    */
            for (x=0; x<NumONurons; x++)
                *TError+=fabs(TPatt[NumONurons*PatternNum+x]-OutNurons[x]);
            break;
        case 4:  /* Minkowski                                                                  */
            for (x=0; x<NumONurons; x++)
                PattError+=pow( fabs(TPatt[NumONurons*PatternNum+x]-OutNurons[x]),(double)thePower);
            *TError+=PattError/(double)thePower;
            break;
        }/* End Switch                                                                         */
}

/*************************************************************************/
void SetCheckBoxes(DialogPtr DialogName, short WhichWord, Boolean *ArrayName, short NumVers)
{
    short    itemType,x;
    Handle   CBitem;
    Rect     box;

    /* Set the appropriate check boxes based on the selected utterance/version combination      */
    for (x=0;x<NumVers;x++)
    {
        GetDItem(DialogName,(x+20),&itemType,&CBitem,&box);
        if (ArrayName[WhichWord*NumVers+x])
            SetCtlValue((ControlHandle)CBitem,TRUE);
        else
            SetCtlValue((ControlHandle)CBitem,FALSE);
    }
}
/*****************************************************************************/
void AscToFloat(char *AscFloat, double *FloatptArray)
{
    char      aCharacter[3],theString[20];
    short     x,Fcount;

    x=0;
    Fcount=0;
    while(AscFloat[x]!='\0')              /* Not EOF                                    */
    {
        /* Check for either a zero,negative sign, or EOF character                   */
        while((isdigit(AscFloat[x])==0) && (AscFloat[x]!='-') && (AscFloat[x]!='\0'))
            x++;

        if (AscFloat[x]=='\0')/* Not EOF of lpc data                                 */
            break;

        theString[0]='\0'; /* Initialize float character string to null             */
        while((isdigit(AscFloat[x])!=0) || ((AscFloat[x])=='.') || (AscFloat[x]=='-') || (AscFloat[x]=='+'))
        {
            sprintf(aCharacter,"%c",AscFloat[x]);
            strcat(theString,aCharacter);
            x++;
        }

        /* Convert character float data to Float number and put in FloatArray        */
        sscanf(theString,"%lf",&FloatptArray[Fcount]);
        Fcount+=1;
    }
}

/*****************************************************************************/
void DisplayResults(DialogPtr DialogName, double *theArray, short NumPatt, short NumVers)
{
    short       itemType,x,Identity,theWord,theVersion;
    double      Max;
    Handle      item;
    Rect        box;
    Str255      theText;
    Max=0.0;
    Identity=0;

    for (x=0; x<NumPatt;x++) /* Only a max. of upto 50 classes for the output neurons     */
    {
        if(Max<theArray[x])/* Find the biggest one this will be the identity of the word  */
        {
            Identity=x;
            Max=theArray[x];
        }
    }
    theWord   = (short) ( (double)Identity / (double)NumVers);
    theVersion  = Identity - NumVers*theWord;
    /* Note: We can use the item numbers of the words set in the dialog box to get a handle
    to the text to display the recognized one.                                             */

    /*  Display theWord name in the word identity box                                     */
    GetDItem(DialogName,theWord+5,&itemType,&item,&box);
    GetCTitle((ControlHandle)item,theText);
    GetDItem(DialogName,3,&itemType,&item,&box);
    SetIText(item,theText);

    /*  Display the Version name in the version identity box                              */
    GetDItem(DialogName,theVersion+15,&itemType,&item,&box);
    GetCTitle((ControlHandle)item,theText);
    GetDItem(DialogName,4,&itemType,&item,&box);
    SetIText(item,theText);
}

/****************************************************************************/
void SetTrainingPat (double  *TPatt, short NumberONurons)
{
    short x,y,Count,NumberPatts;
    Count=0;
    NumberPatts=NumberONurons;

    for (x=0; x<NumberPatts; x++)
    {
        for (y=0;y<NumberONurons; y++)
        {
            /* NumberPatts*x+y points to every neuron(over all patterns). NumberONurons*x+x
             point to the specific neuron in every output pattern that is set to one          */
            if ((NumberPatts*x+y)==(NumberONurons*x+x))
                TPatt[NumberPatts*x+y]=1.0;
        }
    }
}

/***************************************************************************/
void CalcForwardPass(double *InputArray, double *HiddenArray, double *OutputArray,
                     double *HBias, double *OBias, double *WIH, double *WHO,
                     short INurons, short HNurons, short ONurons, short PatNumber)
{
    short x,y;
    double   WeightedSum=0.0;

    /*    Calculation of HiddenNurons                                              */
    for(x=0;x<HNurons;x++)
    {
        for (y=0;y<INurons;y++)
            WeightedSum+=WIH[INurons*x+y]*InputArray[INurons*PatNumber+y];
        if (HBias!=NULL)
            WeightedSum+=HBias[x];
        HiddenArray[x]=1.0/(1.0+exp(-WeightedSum));
        WeightedSum=0.0;
    }

    /*    Calculation of OutputNurons                                              */
    for (x=0; x<ONurons; x++)
    {
        for (y=0; y<HNurons; y++)
            WeightedSum+=WHO[HNurons*x+y]*HiddenArray[y];
        if (OBias!=NULL)
            WeightedSum+=OBias[x];
        OutputArray[x]=1.0/(1.0+exp(-WeightedSum));
        WeightedSum=0.0;
    }
```

```
}

/*****************************************************************************/
void SetCurrentValues       (DialogPtr DialogName,Boolean Bias, float Momtum,float StopVal,
                            float LearnRt,Boolean RandomWts)
{
    char    theText[255];
    short   itemType;
    Handle  itemA,itemB;
    Rect    box;

    /* Biases                                                              */
    GetDItem(DialogName, 14,&itemType,&itemA,&box);
    SetCtlValue((ControlHandle) itemA,Bias);

    /* Random Weight selection                                             */
        GetDItem(DialogName, 26,&itemType,&itemA,&box);
        SetCtlValue((ControlHandle)itemA,RandomWts);

    /*  Learning Rate                                                      */
            GetDItem(DialogName, 11,&itemType,&itemA,&box);
            sprintf (theText, "%f ", LearnRt);
            SetIText(itemA,theText);

    /*  Momentum                                                           */
            GetDItem(DialogName, 12,&itemType,&itemA,&box);
            sprintf (theText, "%f ", Momtum);
            SetIText(itemA,theText);

    /*  Stopping Value                                                     */
            GetDItem(DialogName, 13,&itemType,&itemA,&box);
            sprintf (theText, "%f ", StopVal);
            SetIText(itemA,theText);
}
```

```
/*************************************************************************************/
                              SERIAL ROUTINES
                    Written by Chris D. Love / June 1990-January 3,1991
/*************************************************************************************/
#include  <string.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <SerialDvr.h>

            /*      Prototypes of serial communication functions                    */

void  AscToFloat (char *AscFloat,double *FloatptArray);
void  GetLPCData          (double *FloatptArray);
void PlaybackWord         (short WhichWord);

/**************************************************************************/
void GetLPCData(double *FloatptArray)
{
    char        sendGO[4],receiveOK[4],BytesToSend[4],*WriteBuffer,*ReadBuffer;
    int         ret,ix,HeaderData,InputBufSize;
    long        HeaderSize,DataLength;
    Ptr         InputBuffer;
    SerShk      flag;
    OSErr       theError;

    ix              =       0;
    HeaderSize      =       2L;
    InputBuffer     =       NULL;
    InputBufSize    =       2048;
    WriteBuffer     =       (char *) calloc(128, sizeof(char));
    ReadBuffer      =       (char *) calloc(1024, sizeof(char));

    /* Request to Send =GO            Clear to Send =OK  DataHeader=BytesToSend        */
    strcpy(sendGO, "GO");
    strcpy(receiveOK, "OK");
    strcpy(BytesToSend, '\0');

    /* Channel Communication Parameters                                                */
    flag.fXOn = TRUE;
flag.fCTS = FALSE;
flag.xOn  = 0x11;
flag.xOff = 0x03;
flag.errs = parityErr + hwOverrunErr + framingErr;
flag.evts = breakEvent;
flag.fInX = TRUE;
flag.fDTR = FALSE;

RAMSDOpen(sPortA);              /* Open RAM Serial Driver                    */

/* Initialize Input and Output Ports                                         */
SerReset(AinRefNum,  baud9600 + data8 + stop10 + noParity);
SerReset(AoutRefNum,             baud9600 + data8 + stop10 + noParity);

/* Initialize Handshaking using XON/XOFF                                     */
SerHShake(AinRefNum, &flag);
SerHShake(AoutRefNum, &flag);

/* Set input Buffer size for LPC data to 2Kbytes                             */
SerSetBuf(AinRefNum,InputBuffer,InputBufSize);
InputBuffer   = (char *) calloc(InputBufSize, sizeof(char));

 /***********************************************/
 /*                      Receive Routine                                     */
 /***********************************************/

/* Send GO message to start operation                                        */
     theError = FSWrite(AoutRefNum, &HeaderSize, (Ptr) sendGO);
do
     {
     /* Waiting for ReceiveOK message from other terminal                    */
         theError = FSRead(AinRefNum, &HeaderSize, (Ptr) ReadBuffer);
         ix++;
      }while( ((ret=strcmp(receiveOK, ReadBuffer)) != 0) &&  (ix < 2000));

if (ret == 0)
{
      /* Read the LPC ASCII data & convert to float pt.                      */
     DataLength=300L;
```

```
        theError = FSRead(AinRefNum, &DataLength, (Ptr) ReadBuffer);
        AscToFloat(ReadBuffer,FloatptArray);
    }

    RAMSDClose(sPortA);             /* Close the RAM Serial Port                          */

    /*                     DeAllocate Memory                                              */
    free(ReadBuffer);
    free(WriteBuffer);
    free(InputBuffer);
}

/**************************************************************************/
void PlaybackWord(short WhichWord)
{
    char        sendGO[4],receiveOK[4],BytesToSend[4],*WriteBuffer,*ReadBuffer;
    long        HeaderSize,DataLength;
    SerShk      flag;
    OSErr       theError;
    int         ret,ix,HeaderData,InputBufSize,OutputBufSize;
    Ptr         InputBuffer,OutputBuffer;

    ix          =       0;
    HeaderSize  =       2L;
    InputBuffer =       NULL;
    InputBufSize =      128;
    OutputBufSize =     128;
    WriteBuffer =       (char *) calloc(128, sizeof(char));
    ReadBuffer  =       (char *) calloc(128, sizeof(char));

    /* Request to Send =GO          Clear to Send =OK       DataHeader=BytesToSend  */
    strcpy(sendGO, "GO");
    strcpy(receiveOK, "OK");
    strcpy(BytesToSend, '\0');

    /* Channel Communication Parameters                                               */
    flag.fXOn = TRUE;
    flag.fCTS = FALSE;
    flag.xOn  = 0x11;
    flag.xOff = 0x03;
flag.errs = parityErr + hwOverrunErr + framingErr;
flag.evts = breakEvent;
flag.fInX = TRUE;
flag.fDTR = FALSE;

RAMSDOpen(sPortA);                    /* Open RAM Serial Driver                      */

/* Initialize Input and Output Ports                                                 */
SerReset(AinRefNum, baud9600 + data8 + stop10 + noParity);
SerReset(AoutRefNum,            baud9600 + data8 + stop10 + noParity);

/* Initialize Handshaking using XON/XOFF                                             */
SerHShake(AinRefNum, &flag);
SerHShake(AoutRefNum, &flag);

SerSetBuf(AinRefNum,InputBuffer,InputBufSize);
InputBuffer   =(char *) calloc(InputBufSize, sizeof(char));          /* 128 bytes    */
OutputBuffer =(char *) calloc(OutputBufSize, sizeof(char));          /* 128 bytes    */

/*******************************************************/
/*                         Transmit Routine                                          */
/*******************************************************/

/* Send GO message to start operation                                                */
     theError = FSWrite(AoutRefNum, &HeaderSize, (Ptr) sendGO);
do
     {
     /* Waiting for ReceiveOK message from other terminal                            */
        theError = FSRead(AinRefNum, &HeaderSize, (Ptr) ReadBuffer);
        ix++;
     }while( ((ret=strcmp(receiveOK, ReadBuffer)) != 0) &&  (ix < 20));

if (ret == 0)
     /* Send Utterance Number                                                        */
     theError = FSWrite(AoutRefNum, &HeaderSize, (Ptr) (&WhichWord));

RAMSDClose(sPortA);              /* Close the RAM Serial Port                        */
/*            DeAllocate Memory                                                      */
free(ReadBuffer);
```

```
    free(WriteBuffer);
    free(InputBuffer);
    free (OutputBuffer);
 }

/***********************************************************************************/
                SERIAL ROUTINES FOR THE IBM AND LPC PROCESSING
    Project : Speech Encoder/Decoder
    Programmers: Armein Langi, Chris Love (After [Pars86] Appendix A)
    Purpose : To test the algorithms of the VOCODER
    Start   : June 29, 1990*/
/***********************************************************************************/
#include <stdio.h>
#include <stddef.h>
#include <math.h>
#define  COUNT1  8000
#define FRAME   200
#define COM1    0
signed  short int  BuffData[COUNT1];
short int    *AddrBuffData;
char    *NameFile;
float   ospeech[FRAME]; /*original speech                                    */
float   trcelp[30];  /*transmission CELP                                    */
float   old_trcelp[30];
float   x[FRAME]; /*working speech data for LP                              */
float   r[11]; /*correlation coefficients                                   */
float   sum;  /*general purpose variable                                    */
int     i, j, k, l, go, count;
int     auto_err; /*error flag for the autocorr..                           */
int     pre_err; /*error flag for the predictor ...                         */
char    result[350];

main()
{     char    c;
    result[0] = '\0';
    go = 1;
    set_baud();
    printf("\nGo Ahead ...!");
    read_command();
    while(go == 1)
```

```
        {
            lp();
            store_lp();
            printf("\nGo Ahead ...!");
            read_command();
            if(go == 0)
            {
               printf("\nFinish, Thanks for using this program ...");
            }
        }
}
lp()
{
    printf("\nPress any key to start acquiring data ...!");
    lin2asc();
    l = 0;
    count = 2000;
    get_lp_speech();
    printf("\nGot Them\n     Now Analyzing Speech ... ");
    lpc();
    l = 10;
    printf("\nGot %d LPC ",l);
    count = 4000;
    get_lp_speech();
    lpc();
    l = 20;
    printf("\nGot %d LPC",l);
    count = 6000;
    get_lp_speech();
    lpc();
    printf("\nGot all the LPC ... Now sending to Macintosh ...!");
}
store_lp()
{
    char   res_buff[11];
        for (i = 0 ;i < 30;i++)
        {
           sprintf(res_buff,"%+f ",trcelp[i]);
           strcat(result,res_buff);
```

```
            /*printf("%+f ", trcelp[i]);                                        */
        }
        send_data();
        result[0] = '\0';
}
get_lp_speech()
{
    for (i = 0 ;i < FRAME ;i++)
    {
        ospeech[i] = 5 * (float) BuffData[count+i] /32768;
        /*printf("%d %f\n",i,ospeech[i]);                                       */
    }
}
lin2asc()
{
    AddrBuffData = &BuffData[0];
    _asm{
    push   es
    mov    cx,COUNT1
    mov    dx,0d000h
    mov    es,dx
    mov    di,AddrBuffData
    mov    dx,304h
    mov    al,0011b  ; enable int
    out    dx,al
tunggu:
    mov    dx,306h;
    in     al,dx;
    and    al,4;
    jz     tunggu
    mov    al,001b  ; ask for the EB memory access
    mov    dx,304h
    out    dx,al
    mov    al,es:[1401h]
    mov    [di],al
    inc    di
    mov    al,es:[1402h]
    mov    [di],al
    inc    di
    mov    dx,304h
```

```
        mov    al,01111b  ; interrupt the EB
        out    dx,al
        mov    al,0011b  ; release the of interrupt the EB
        out    dx,al
        dec    cx;
        jnz    tunggu;
        pop    es
        }
}
lpc()
{
    printf("\n             LPC ");
    hamming();
    autocorr();
    predict();
}
hamming()
{
    for(i = 0; i < FRAME ; i++)
    {
        x[i] = ospeech[i] + ospeech[i] * 0.84 * cos(6.28 * (i-120)/240);
    }
}
autocorr()
{
    for (i = 0; i<= 10; i++)
    {
        sum = 0;
        for (k = 0; k< FRAME - i; k++)
        {
            sum = sum + x[k] * x[k + i];
            /*printf("Sum = %f x = %f ", sum, x[k]);                              */
        }
        r[i] = sum;
    }
    if (r[0] == 0)
    {
        auto_err = 1;
        printf("\n Correlation error ");
    }
    else
    {
        /*for (i = 0; i <= 10; i++)
           printf("\n%f ",r[i]);                                          */
    };
}
predict()
{
    float  a[11]; /*direct coefficients                                  */
    float  rc[11]; /*reflection coefficients                             */
    float  pe;
    float  akk, ai, aj, ra;
    pe = r[0];
    a[0] = 1;
    for (k = 1; k <= 10; k++)
    {
        sum = 0;
        for (i = 1; i <= k; i++)
        {
           sum = sum - a[k - i] * r[i];
        }
        akk = sum/pe;
        rc[k] = akk;
        a[k] = akk;
        for (i = 1; i <= k/2; i++)
        {
           ai = a[i];
           aj = a[k-i];
           a[i] = ai + akk * aj;
           a[k - i] = aj + akk * ai;
        }
        pe = pe * (1 - akk * akk);
        /*printf("\npe = %f", pe);                                        */
        if (pe <= 0)
        {
           pre_err = 1;
        }
    }
```

```
        if (pre_err == 1)
        {
            printf("\n predictor error ...");
        }
        else
        {
            /* for(i = 0; i <= 10; i++)
            {
                printf("\ni=%d  a= %f  rc = %f", i, a[i], rc[i]);
            }*/
            for(i = 0; i <10; i++)
            {
                trcelp[i+l] = a[i+1];
            }
        }
}
set_baud()
{
        _asm{
        mov     ah,00
        mov     dx,COM1                 /* 0 means COM1                         */
        mov     al,011110011b           /*9600-N-1-8                            */
        int     14h
        }
}
read_command()
{
_asm{
stat:       mov     ah,02h
            mov     dx,COM1
            int     14h
            and     ah,08h
            jnz     stat
            and     ah,0fh
            jnz     error
            cmp     al,'G'
            jnz     ostat
            mov     ah,02h
            mov     dl,al
            int     21h
```

```
stat2:  mov     ah,02h
        mov     dx,COM1
        int     14h
        and     ah,08h
        jnz     stat2
        and     ah,0fh
        jnz     error
        cmp     al,'O'
error:  jnz     stat
        mov     ah,02h
        mov     dl,al
        int     21h
        jmp     ok
ostat:  cmp     al,'O'
        jnz     stat
        mov     ah,02h
        mov     dx,COM1
        int     14h
        and     ah,08h
        jnz     ostat
        and     ah,0fh
        jnz     error
        cmp     al,'K'
        jnz     stat
        mov     ax,0
        mov     go,ax
        jmp     ok2
ok      mov     dx,03f8h        /*Now the host transmits.              */
        mov     al,'O'
        out     dx,al
        mov     ah,01h
        mov     al,'K'
        mov     dx,0
        int     14h
        mov     ah,02h
        mov     dl,0ah
        int     21h
        mov     ah,02h
        mov     dl,0dh
        int     21h
ok2:    nop
    }
}
send_data()
{
char    *y=&result[0];
    _asm{
        push    si
        push    dx
        push     cx
        mov     cx,300
        mov     si,y
        mov     al,[si]
        mov     dx,03f8h
        out     dx,al
        dec     cx
        inc     si
lagi:
        mov     'ah,01h
        mov     al,[si]
        mov     dx,COM1
        int     14h
        inc     si
        dec     cx
        jnz     lagi
        pop      cx
        pop     dx
        pop     si
        }
}
```