Speaker Identification
using Multi-Layer Perceptron Neural Networks

A Research Project by

Annliza Nina dL. Andrada
Diane Felice N. Lerma
Christine Pamela dG. Obedencio
Rochelle S. Ticlao

Submitted to the Department of Computer Science
College of Engineering
University of the Philippines

In Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science in Computer Science

College of Engineering
University of the Philippines
Diliman, Quezon City

March 2003

Table of Contents

List of Tables

List of Figures

Abstract

Open communication systems and computer networks are now prevalent. Thus, there exists the need for access control and authentication. Passwords are the basic method for system authentication but they pose a number of problems. Simple passwords are almost always readily associated with the owner and are more prone to easy guessing while a complicated password is more prone to disremembering. Writing the password on a piece of paper so as not to forget it allows for more complications. What this paper suggests then is an alternative – use a person's voice to identify him or her and prevent unauthorized use and access to facilities and confidential data. Voice, which is inherent to every individual, can be a reliable source of security.

In this paper, a system for identifying a person through his voice is proposed. This system can be divided into two phases: the enrollment/training phase and the testing phase. Each phase is further divided into three stages, the first two of which are common to both phases. The first stage is Digital Signal Acquisition where voice samples are acquired and converted to data that can be interpreted by the system. The second stage is Speech Analysis where Feature Extraction is done. This is where information inherent to the voice is extracted and the features used are its Mel-Frequency Cepstral Coefficients (MFCC). The third stage is the Classifier where supervised learning is done through a Multi-Layer Perceptron (MLP) Neural Network. Inputs to the MLP are the MFCCs obtained from the second stage. In the enrollment phase, the MFCCs obtained from all the speakers in the speaker set are presented to the network. In the testing phase, the network is simulated and the extracted features from a recent voice sample are used to identify the speaker.

Acknowledgement

The researchers would like to thank Gamby, the spider in NEC 414 for accompanying us during our sleepless nights as we work on our project.

Chapter 1

Introduction

1.1 Statement of the Problem

The goal of speech communication is to convey messages. From this, it follows that the message is the most important information embedded in the speech signal. But it is not the only information. Other kinds of information embedded in the speech signal include the identity of the speaker, the language spoken, the presence and type of speech pathologies, and the physical and emotional state of the speaker. Of these other kinds of information, the identity of the speaker is chosen to be the focus of this paper. This is because speaker characteristics impose striking and comprehensive effects on spoken utterances and because there are many practical applications awaiting practical and effective methods of automatically extracting speaker identity information from speech signals.

The objective of speaker identification is to determine the identity of a speaker among a set of speakers, based on the individual's voice. This is contrasted to speaker verification where the objective is to verify the person's claimed identity. In speaker identification, matching is one-to-many while in speaker verification, matching is one-to-one. Speaker identification and speaker verification fall under the general category

of speaker recognition.

The problem, therefore, is to make the machine recognize a person's voice to be able to identify him/her. In order for humans to recognize voices, the voices must be familiar. So too, with machines. The machine must undergo a process of "getting to know" the speakers which is called the training phase. This process consists of collecting data from utterances of people to be identified. The second component of speaker identification is testing: namely the task of comparing an unidentified utterance to the training data and making the identification.

1.2 Significance of this Study

The basic access-control method is knowledge-based and the most popular form of which is the use passwords. Using passwords though have a number of flaws. Passwords can only be either simple or complicated. Simple passwords are prone to easy guessing while complicated passwords are frequently forgotten. Some would write down their passwords on a piece of paper for instant recall but this is prone to misplacement and accidental exposure. Passwords alone as security measure is therefore unreliable.

The proposed method, speaker identification, is probably the most natural and economical method for solving the problem of unauthorized use of computer and communications systems and multilevel access control.

Speaker identification is a biometric performance system. Biometric systems automatically recognize a person by using distinguishing traits, in this case the voice. Speaker identification is a performance biometric because a person performs a task to be recognized. An individual's voice, like other biometrics, cannot be forgotten or

misplaced unlike knowledge-based or possession-based access-control methods.

The application of speaker identification is not limited to access-control and security measures however. Although security applications do abound since access to cars, buildings, bank accounts and other services may be voice controlled in the future. The potential for application of speaker identification systems exists any time speakers are unknown and their identities are important. In meetings, conferences, or conversations, the technology makes machine identification of participants possible. If used in conjunction with continuous speech recognizers, automatic transcriptions could be produced containing a record of who said what. This capability can serve as the basis for information retrieval technologies from the vast quantities of audio information produced daily. In law enforcement, speaker identification systems can be used to help identify suspects.

1.3 Scope of this Study

Speaker identification systems can be classified through the mode of operation used – text dependent or text independent. Text dependent speaker identification systems require that the target speaker utter a specific phrase or a given password for both the training and testing phases. Text independent speaker identification systems identify the target speaker regardless of his utterance. He is recognized mainly through his voice features.

Speaker identification systems can also be closed set or open set. Closed set speaker identification refers to the case where the speaker is known a priori to be a member of a set of N speakers and the output is given as the closest match. Open set speaker identification includes the additional possibility where the speaker may

not be included in the N speaker set. It requires an additional thresholding step to determine if the speaker is "out of set" and therefore rejected.

For security purposes, an open set system is better than a closed-set so as not to allow access to impostors. Statistics show that text-dependent systems provide a higher performance rate than text-independent systems. Thus, the research project focuses on an open set text dependent speaker identification system.

To accomplish this, the researchers used Mel-Frequency Cepstral Coefficients to uniquely describe each person's voice and a Multi-Layer Perceptron Network to learn the differences between the features of each person's voice and be able to identify the owner during the simulation of the network in the testing phase.

For the simulation, digit utterances from a number of speakers were collected. The first speaker set consisted of 3 speakers, the second speaker set – 4, and the third speaker set – 15. The word training set consisted of the digits "one", "two", "eight", "nine", and "ten". This was done so that the speakers would use their normal speaking voice. For the training part alone, 20 voice samples for each utterance was required of the speakers.

Chapter 2

Theoretical Framework

2.1 Review of Related Literature

2.1.1 Improved Voice Identification Using a Nearest-Neighbor Distance Measure[6]

A comparison of the baseline algorithm with a new modified algorithm. The modifications are segment-based scoring, limiting likelihood ratio estimates for robustness and estimating biases associated with reference files. These algorithms compare the underlying probability distributions of speech utterances using a method that is free of assumptions regarding the form of the distributions (e.g. Gaussian, etc.).

The baseline algorithm is based on a "nearest neighbor" method, and is applied to a set of M test messages simultaneously. It uses K reference files for each of T talkers. Other changes to the baseline algorithm are that the nearest-neighbor distance used is assymetrical and does not include a self-entropy correction.

After frame selection, each of the T test utterance file is subdivided into groups of 50 frames. Each such group is called a segment. The nearest neighbor distance can then be applied to find a distance from each segment of each test message to each file for each reference talker. It has been found that a major source of error

in the baseline algorithm is related to biases associated with each reference file, and that these biases can be estimated by examining the nearest-neighbor distances to reference files across the test utterance messages. The biases β_{kl} are calculated in three steps, yielding successive values $\beta^{[1]}$, $\beta^{[2]}$ and $\beta^{[3]}$. After estimating the reference file biases in this way, the segment-to-reference file distances are recalculated a final time. To convert the $\lambda_{im,kt}$ to approximate negative log likelihood ratios, they are next normalized over reference files by subtracting the minimum of those values over all reference files.

In a 24 talker speaker identification test of 97 trials using Switchboard speech data, the modified algorithm achieves error-free performance for sixty second test utterance tests.

2.1.2 Speaker Identification Based on the Time-Delay Hierarchical Mixture of Experts[5]

The Hierarchical Mixture of Experts (HME) is based on the principle of divide-and-conquer in which a large, hard to solve problem is adaptively broken up into many, smaller, easier to solve problems. The HME architecture is a tree in which the gating nets sit at the nonterminals of the tree. These nets receive the vector x as input and produce scalar outputs that are a partition of unity at each point in the input space. The expert nets sit at the leaves of the tree. Each expert produces an output vector for each input vector. These output vectors proceed up the tree, being blended by the gating net outputs.

Consider a special multiway classification problem in which the outputs is a binary vector with a single non-zero component. A generalized Bernolli density is defined as follows

$$P(y_1, y_2, \ldots, y_K) = \prod_{k=1}^{K} p_k^{y_k} (1 - p_k)^{1-y_k} \tag{2.1.1}$$

For training the time-delay HME, the Expectation-Maximization algorithm is adopted. The algorithm consists of calculating posterior probabilities in E-step and solving several *Iteratively Reweighted Least Squares* (IRLS) problems in M-step.

The speaker identification system based on the time-delay HME with EM algorithm based on the proposed *generalized Bernolli density* can achieve the satisfactory performance. It requires much fewer epochs than conventional neural networks during the training.

2.1.3 CD-HMM Algorithm Performance for Speaker Identification on an Italian Database[2]

In this work the performance evaluation of a closed-set, text dependent, automatic speaker identification algorithm applied to an Italian Database has been carried out. This recognition system consists of an LPC or LPC-Cepstral feature extractor and a CD- HMM classifier. The Italian Database SIVA the MUSER consists of 360 phone calls made by 20 different male speakers coming from various Italian regions. In particular, the system has been tested in various working conditions such as different training set, different spoken words and a variable number of states of the CD-HMM classifier for both the LPC and LPC-Cepstral features.

Speaker recognition systems are basically characterized by using sequences of feature vectors representative of the input signal. In the training phase, given a speaker and his training set of T utterances, the model associated to the speaker is created. In the testing phase, given the unknown speaker utterance, the identification is achieved by associatiing to the unknown speaker the identity of model K that maximize the

probability that the utterance sequences of the training set are generated by that model.

The classifier considered in this work differs from the schemes presented in other works for two peculiarities. The first one concerns the number of mixtures of each state. The second peculiarity regards the initial estimates of the CD-HMM parameters. It has been chosen to consider a uniform segmentation of observation sequences into the states instead of the random one usually proposed into the literature. This choice is justified by some trials which have shown that a random initialization may result in a very unfair observation distribution among the states.

2.1.4 Speaker Identification Using Dynamic Time Warping With Stress Compensation Technique[8]

This is an algorithm for an isolated word text-dependent speaker identification under normal and four stressful styles : shout, slow, loud and soft. Comparing DTW combined with cepstral stress compensation technique with DTW without the technique, the recognition rate has improved to some extent with a little increase in the computations. Dynamic Time Warping Algorithm : The time-normalized distance can be calculated between two speech patterns as the minimum distance between them. Every utterance is brought into time registration with the other utterances to find the distance (Euclidean distance) between every two utterances that are represented by points.

The theory of the DTW can be found in many books. A comparison is performed to the templates in normal condition based on the following. 1. The distance among different utterances within the same speaker are very close to each other, but the distances among different speakers are not close; on the other hand, the distances

between each speaker and other speakers are higher than that within each speaker with different utterances. 2. To select the optimum utterance for each speaker, the distance should be minimum within the speaker and maximum to the other speakers.

One solution to improve recognition performance under stressful conditions is to use Hidden Markov Model or Neural Network.

2.1.5 Speaker identification using hybrid LVQ-SLP networks[4]

Hybrid LVQ-S networks have the potential to provide improved recognition accuracy with greatly reduced training time. The basic idea is to construct a 2-layer network by concatenating an LVQ network with a single layer perceptron (SLP) network to train 2 layers separately. With this comes a layer called the connection layer. The SLP network is a standard feedforward network. Each output node in the SLP network corresponds to an individual class, it computes the weighted sum of its inputs and then passes the sum through a sigmoid function. The functions of a node in the connection layer are: 1. finding out the minimum output from its code vectors and 2. applying the Gaussian function *ti* the distance value and then passing the result to an input node of the SLP network.

An LVQ-SLP has better classification performance than LVQ hence the borders of decision patterns provided by an LVQ network are piecewise linear formed by midplanes between code vectors of neighboring classes, whereas more complex decision regions can be formed by an LVQ-SLP network due to the additional SLP layer and the nonlinear functions.

2.1.6 Text-Dependent Speaker Recognition Using the Information in the Higher Frequency Band[3]

Many studies on speaker recognition have been carried out and in most cases, the high frequency region, with low energy, has not been regarded as having rich information, except in the case of fricative sounds.

This paper presents the results of speaker identification experiments performed in a text-dependent mode using various frequency bands.

The particulars of the experiment. An unconstrained-endpoint dynamic time warping is used in speaker recognition system. A relatively wide adjustment window is used in order to investigate the speaker recognition rates. The selective linear prediction is used in implementing feature extraction. Cepstral coefficients are calculated by the selective LP analysis with a frame period of 16 ms and a frame length of 32 ms.

Experiments of text-dependent speaker recognition using the information in the higher frequency band were carried out. Based on the results, it is concluded that a rich amount of speaker individual information is contained in the higher frequency band, and it is useful for speaker recognition. However, additional work is needed to examine the cases of female speakers in text-independent mode.

2.2 Theories that have a Bearing on the Problem

2.2.1 Mel Frequency Scale

Many studies have shown that human perception of the frequency content of sounds does not follow a linear scale. This can be either for pure tones or for speech signals and it has led to the idea of defining subjective speech of pure tones. For each tone

with an actual frequency measured in Hertz, a subjective speech is measured on a scale called the **Mel scale**. In this scale, the pitch of a 1000 Hz tone is defined as 1000 mels. This is 40 dB above the perceptual hearing threshold. Other values are obtained by adjusting the frequency of a tone such that the human perceived frequency is half or twice the perceived frequency of a reference tone with a known mel frequency. Subjective speech is constant with the logarithmic frequency below 1000 Hz and linear with the logarithmic frequency beyond 1000 Hz.

Another important subjective criterion of the frequency content of a signal is the critical band. The critical band refers to the bandwidth within which subjective responses such as loudness remain constant until the noise bandwidth exceeds the width of the critical band. Upon exceeding the critical band width, increased loudness is perceived. Similarly, a subcritical bandwidth complex sound of constant intensity is about as loud as an equally intense pure tone of a frequency lying at the center of a band, regardless of the overall frequency separation of multiple tones. When the separation exceeds the critical bandwidth, the complex sound is perceived as becoming louder.

The subjective, nonlinear perception of frequency has led to an objective computational model that provides a mechanism to convert a physically measured spectrum of a given sound into a psychological subjective spectrum. In this model, each frequency component of the spectrum is replaced by a specific loudness level according to an empirical power law over a range of tonalness units. A unit of tonalness corresponds to a critical band and is called a **Bark**.

To combine the ideas of the mel-scale and the critical band, calculation of the spectral distance measure is modified. This modification entails warping the frequency

scale to the mel-scale or Bark scale. The mel-scale frequency is then simulated using a filter bank spaced uniformly on a mel-scale, where the output energy from each filter band approximates the modified spectrum. Let the output energy of the k^{th} filter be denoted by $\widetilde{S_k}$, the mel-warped cepstrum $c_{mel}(n)$ is obtained by taking the shifted discrete cosine transform (DCT) of the mel-scale spectrum as

$$c_{mel}(n) = \sum_{k-1}^{K} log(\widetilde{S_k}) cos(n(k-0.5)\frac{\pi}{K}) \tag{2.2.1}$$

2.2.2 Backpropagation Algorithm

In supervised learning, the **Backpropagation algorithm** is used. The key in implementing backpropagation is the use of a smooth monotonic nonlinearity with a continuous first derivative, in this project's case a **Hyperbolic Tangent Sigmoid**. The desired output is a pattern class or a functional relationship. For each exemplar, the actual neural network output is compared to the desired output and a **Mean-Squared Difference** (MSE) is calculated. This MSE is minimized with a Gradient descent algorithm. Backpropagation minimizes MSE by adjusting the weights proportional to the error gradient. The adjustment to each neural network weight is proportional to the gradient of the performance measure.

Chapter 3

Methodology

The typical structure of an automatic speaker identification system is presented in Figs. 3.1 and 3.2. Both the two phases - enrollment and testing - consist of a speech analysis section and a classifier. The purpose of the speech analysis section is to convert the speech signal into a sequence of feature vectors that contain relevant speaker information. In the classifier section, the speech features are used in the enrollment (or training) phase to create or improve models that represent speaker identity. In the testing phase, on the other hand, the speech features are used to select the model that best describes the most likely speaker identity or to determine if the test speaker is not in the set of known speakers.

3.1 Digital Signal Acquisition

The utterance of a person is an acoustic sound pressure wave. Before a digital machine, such as a computer, can process speech signal, it must be first represented in digital form.

The most common type of digital audio recording is called pulse code modulation (PCM). Pulse code modulation is what compact discs and what most WAV files use.

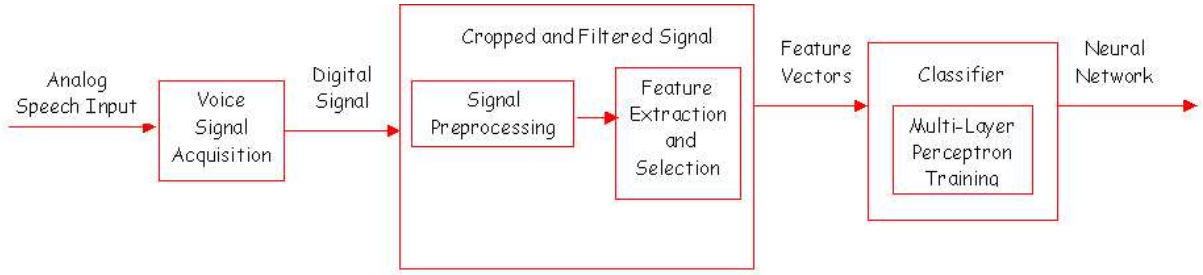

Figure 3.1: Enrollment Phase

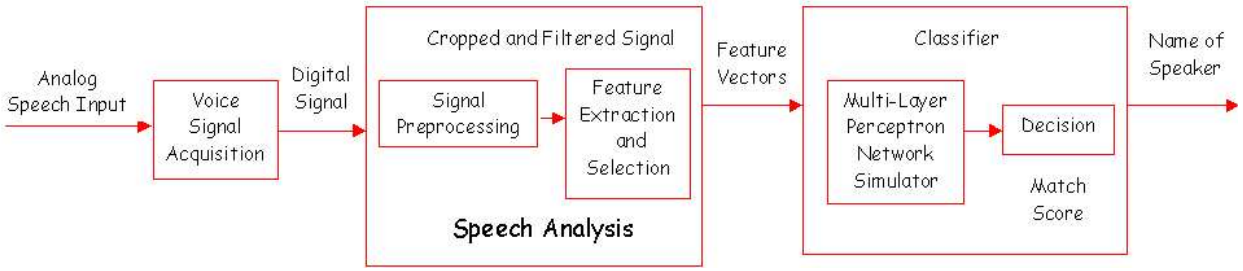

Figure 3.2: Testing Phase

In PCM recording hardware, a microphone converts a varying air pressure (sound waves) into a varying voltage. Then an analog-to-digital converter measures (samples) the voltage at regular intervals of time.

The software used for the purposes of the research is the Microsoft Sound Recorder and the format used for the recording is PCM, 16 bit Mono, 11,025 Hertz. The 11,025 Hertz is the sampling rate which means that in the recording, 11,025 samples are taken every second. Each sampled voltage gets converted into a 16-bit integer.

The output of this section is a representation of the digital signal in the time

domain. An example is shown in Fig. 3.3.

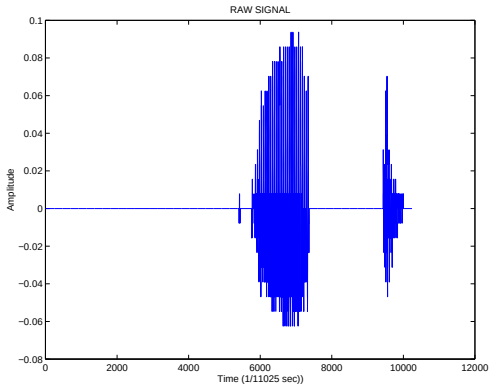

Figure 3.3: Speech Signal in the Time Domain

3.2 Speech Analysis Module

The main goal of this module is to reduce the speech signal information down to the distinct and important amount that is useful in the speaker recognition task. This module consists of: signal preprocessing and feature extraction.

3.2.1 Signal Preprocessing

The preprocessing of the speech signal includes the End Point Detection procedure which separates silence from speech and determines the boundaries of the utterance. This procedure also tries to eliminate as much noise from the signal.

End Point Detection Algorithm

Getting the starting point:

1. Get the highest variance data point among the data points within the first

three frames. (It is assumed that the first three frames does not contain any significant voice signal.)

2. Get the first peak and save its location or index.

3. Get the next one.

4. If the next peaks collected amounted to 80, this means that the saved location or index in 2 is already the starting point. Stop. Check if the distance of the second peak to the first peak is greater than 80. If it is, go back to two and restart the counting of the next peaks. Else, go to 5.

5. Get a peak. Go to 3.

Getting the ending point:

1. Get the highest variance data point among the data points within the last three frames. (It is assumed that the first three frames does not contain any significant voice signal.)

2. Starting at the end, get the first peak and save its location or index.

3. Get the next one.

4. If the next peaks collected amounted to 80, this means that the saved location or index in 2 is already the starting point. Stop. Check if the distance of the second peak to the first peak is greater than 80. If it is, go back to two and restart the counting of the next peaks. Else, go to 5.

5. Get a peak. Go to 3.

Figs. 3.4 and 3.5 illustrate how the End Point Detection algorithm works.

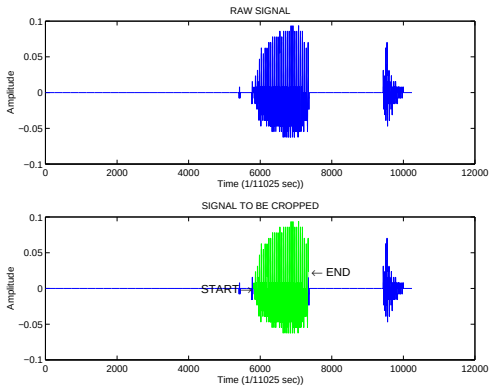

Figure 3.4: End Point Detection

3.2.2 Feature Extraction

After the speech signal is preprocessed, characteristic parameters called feature vectors are obtained from the signal. These feature vectors contain relevant speaker information that can sufficiently model a speaker and distinguish him/her from another. The extraction of these features is the key step in solving the recognition problem, as discussed in the next stage.

For speaker identification, the features extracted should be invariant with regard to the desired speaker (low intra-speaker variability) while exhibiting a large distance to other speakers (high inter-speaker variability). Commonly used features for speaker identification are intensity, pitch, short-time spectrum, prediction coefficients, formant frequencies and bandwidth, cepstral coefficients and spectral correlation. In this study , the feature used is the MFCC or Mel-Frequency Cepstral Coefficients (also called Mel-Warped Cepstrum).

In the feature extraction module, the following theories were used.

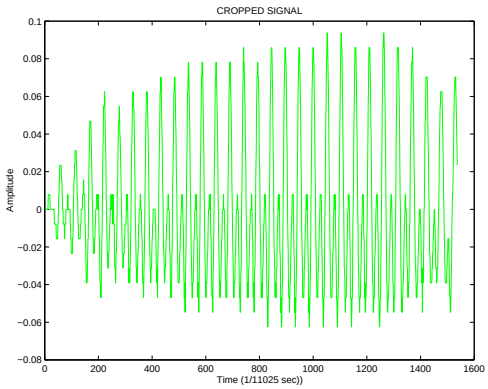

Figure 3.5: Cropped Signal

Data Windowing

Digital signals normally contain a wide mixture of frequencies. Only a few of them are exact harmonics. Spectral leakage is generally present which may lead to difficulties of interpretation. The original signal is thus tapered before transformation to reduce any discontinuities at its edges. This is obtained by multiplying the input signal with a suitable window function. Common window functions used are Rectangular, Hamming, Hanning, Blackman, Kaiser, and Bartlett. For the purposes of this research, the windowing function used is a **Hamming window** which is particularly valuable for its low sidelobe levels. These sidelobe levels substantially reduce distant spectral leakage. The Hamming window is defined as follows:

$$w[n] = 0.54 + 0.46cos(\frac{(2n - N + 1)\pi}{N}) \tag{3.2.1}$$

where N is the length of the window.

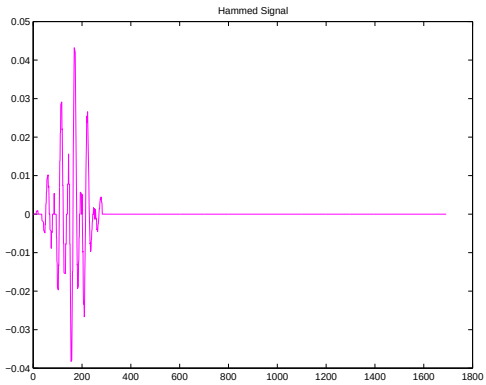

Figure 3.6: Hamming Window for 1 Signal Frame

Discrete Fast Fourier Transform Algorithm

The **discrete Fourier Transform** is an algorithm which converts a sampled complex-valued function of time into a sampled complex-valued function of frequency. In simpler words, it is used to convert a digital signal in the time domain into a set of points in the frequency domain. The input to the DFT algorithm is a set of N time values x_k; the algorithm then computes a set of N complex values X_k that represent the frequency-domain information or sinusoidal decomposition of the time signal. The DFT algorithm is, in general, a computationally intensive algorithm and may require a considerable amount of computer time if N is large. However, if the number of points is a power of two ($N = 2^M$), then a special algorithm called the **fast Fourier Transform** (FFT) can be used. The FFT algorithm greatly reduces the number of computations needed to convert the time signal into the frequency domain.

To have a rigorous mathematical understanding of the FFT, here is an equation which tells the exact relationship between the inputs and outputs of the FFT. In this equation, x_k is the kth complex-valued input (time domain) sample, y_p is the pth

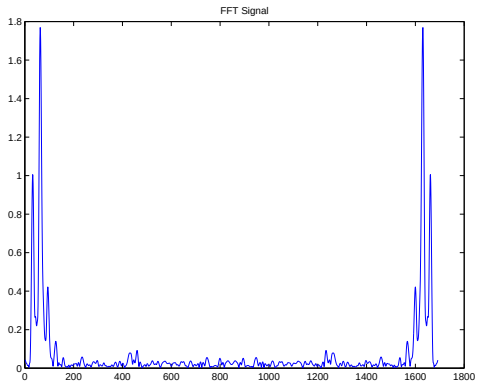

Figure 3.7: FFT of 1 Signal Frame

complex-valued output (frequency domain) sample, and $n = 2^N$ is the total number of samples. k and p are in the range $0 \ldots n-1$.

$$y_p = \sum_{k=0}^{n-1} x_k(cos(2\pi\frac{kp}{n}) + isin(2\pi\frac{kp}{n})) \tag{3.2.2}$$

Although this formula tells what the FFT is equivalent to, it is not how the FFT algorithm is implemented. This formula requires $O(n^2)$ operations, whereas the FFT itself is $O(nlog_2(n))$.

MFCC or Mel-Warped Cepstrum[7]

The mel-warped cepstrum is calculated using a filter-bank approach in which the set of filters are of equal bandwidth with respect to the mel-scale of frequencies. This is because human perception of the frequency content of sounds does not follow a linear scale. Instead they are approximately linear with logarithmic frequency beyond about 1000 Hz. In addition, the critical band is a constant with the logarithmic frequency below 1000 Hz and linear with respect to the logarithmic frequency beyond 1000 Hz. The critical band refers to the bandwidth within which subjective responses such as

loudness remain constant until the noise bandwidth exceeds the width of the critical band. The mel-scale is defined in such a way that the 1000 Hz in the linear frequency domain is 1000 mels, and the other values are obtained by adjusting the frequency of a tone such that the human perceived frequency is half or twice the perceived frequency of a reference point with a known mel frequency. The mel-scale frequency is simulated using a filter bank spaced uniformly on a mel-scale, where the output energy from each filter band approximates the modified spectrum. Let the output energy of the k^{th} filter be denoted by $\widetilde{S_k}$, the mel-warped cepstrum $c_{mel}(n)$ is obtained by taking the shifted discrete cosine transform (DCT) of the mel-scale spectrum as

$$c_{mel}(n) = \sum_{k-1}^{K} log(\widetilde{S_k}) cos(n(k-0.5)\frac{\pi}{K}) \tag{3.2.3}$$

Feature Extraction Algorithm

1. Divide the voice signal into frames.
2. Repeat 2 to 14 for every frame.
3. Get the hammed window for the frame and multiply it to the framed signal (see Fig. 3.6)
4. Get the absolute values of the FFT (see Fig. 3.7) of the hammed frame.
5. Square the first half of $< 4 >$. The result of doing this step on the signal frame in Fig. 3.7 is shown in Fig. 3.8.
6. Compute the number of linear channels:

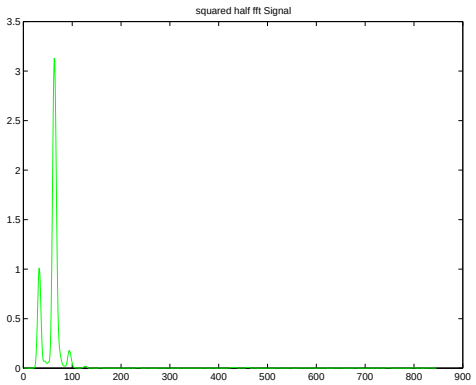

Figure 3.8: Square of the 1^{st} Half of the FFTed Signal

$$NumLinChan = \frac{\lfloor((1000 * 2 * length(frame\ from < 5 >)))\rfloor}{sampling\ frequency * 16} \tag{3.2.4}$$

7. Create 16 linear channels and concatenate all of them. Each linear channel is the Bartlett of the NumLinChan.

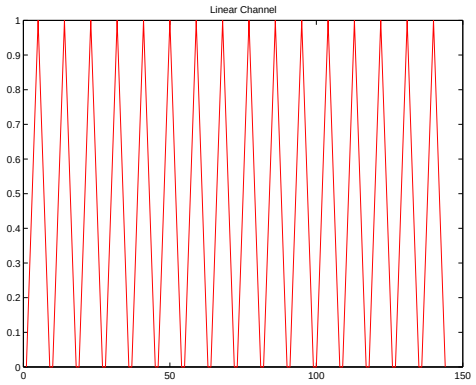

Figure 3.9: Linear Channels

8. Compute the number of logarithmic channels:

$$NumLogChan = \frac{\lfloor((length(frame\ from < 5 >) - (length(final\ linear\ channel))\rfloor}{15} \tag{3.2.5}$$

9. Create 4 logarithmic channels and concatenate all of them. Each logarithmic channel is the Bartlett of the NumLogChan.

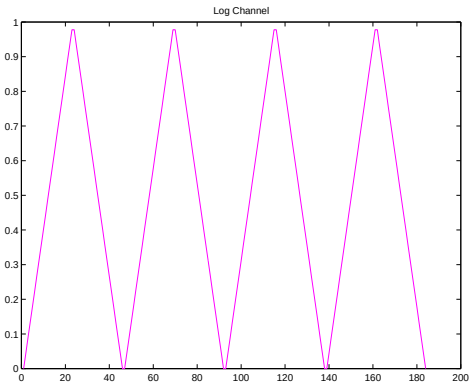

Figure 3.10: Logarithmic Channels

10. Concatenate the 16 linear channels and the 4 logarithmic channels created.

11. Pad the resulting channel from $< 10 >$ with zeros if its length is less than that of the frame for $< 5 >$

12. Multiply the resulting channel from $< 11 >$ and the final frame from $< 5 >$

13. Get the sum of the linear channels and the logarithmic channels.

14. Compute the MFCC formula Eq.3.2.3 where $n = 7$, $K = 20$, and $S = result\ in < 13 >$.

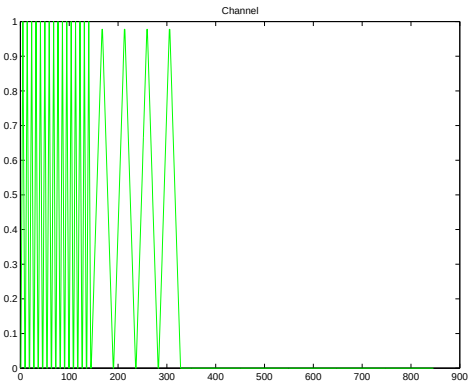

Figure 3.11: Concatenated Channels with Zero Padding

3.3 Classifier Module

This module is the most important one. This is where the training and the testing phases differ. The objective of the training phase is to create distinct models consistent with pattern representation, based on one or more known patterns of each individual class, for each utterance class to be recognized. The representation can be a pattern itself, which generally is called a template; or a rich signal model that characterizes the statistical variations of the speaker class. The function of the testing or recognition phase is to get the vector of features and compare them with templates(which are produced in the training phase). The distinction between input utterance and each voice template is computed. The best match is considered the target speaker. Broadly speaking, there are three approaches for the classifier module: a. the acoustic phonemic approach b. the artificial neural network approach, and c. the pattern recognition approach.

In this study, the artificial neural network approach is used.

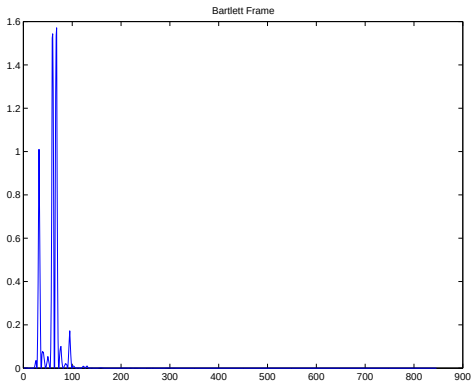

Figure 3.12: Bart Frame

3.3.1 Artificial Neural Network[1]

The artificial neural network attempts to mechanize how a person applies his or her intelligence. It visualizes, analyzes and finally makes a decision on the measured acoustic features. Neural networks (NNs) are information processing systems that have certain performance characteristics in common with biological neural networks. Neural networks have been developed by generalizing mathematical models of human cognitive or neural biology. A neural network is characterized by: 1. Its pattern of connections between the neurons (i.e. its architecture). 2. its method of determining the weights on the connections (i.e. training or learning algorithm), and 3. its activation functions.

In speech recognition, the most common NN architecture is the Multilayer Perceptron (MLPs). Typically, MLPs have a layered feed forward architecture with an input layer (consisting of the input variables), zero or more hidden (intermediate) layers and an output layer. Each layer computes a set of linear discriminate functions (via a weight matrix). This is followed by a nonlinear function, which is often a

sigmoid function

$$f(x) = \frac{1}{1 + exp(-x)} \tag{3.3.1}$$

This nonlinear function performs a different role for the hidden units than the output units. On the hidden units, it serves to generate high order moments of the inputs. (This can be done effectively by many nonlinear functions, not just the sigmoid function). On the output units, the nonlinearity can be viewed as a differentiable approximation to the decision threshold logic unit or perceptron (i.e. essentially to control error). For this purpose, the output nonlinearity should be a sigmoid or sigmoid- like function. In speech recognition, the NN trained by the known utterance will find the best weight vector(s) for all layers. This is done by applying the feature vector (FV) to the input layer and computing the best class for this segment (utterance). After training the MLP, the recognition process can be started by applying the test feature vector. The system identifies the owner of the test utterance.

3.3.2 The Multi-Layer Perceptron Network Training

A Feed-Forward backpropagating Multi-Layer perceptron is used having 400 input nodes, 800 and 300 hidden nodes respectively for its two hidden layers and 15 output nodes – one for each of the registered speakers. Transfer function of the three layers except for the input layer is the Hyperbolic Tangent Sigmoid function or "tansig" and the training function is "traingdx" or Gradient descent w/momentum and adaptive learning rate backpropagation. The backpropagation weight/bias learning function used is the default "learngdm" or the Gradient descent w/momentum weight/bias

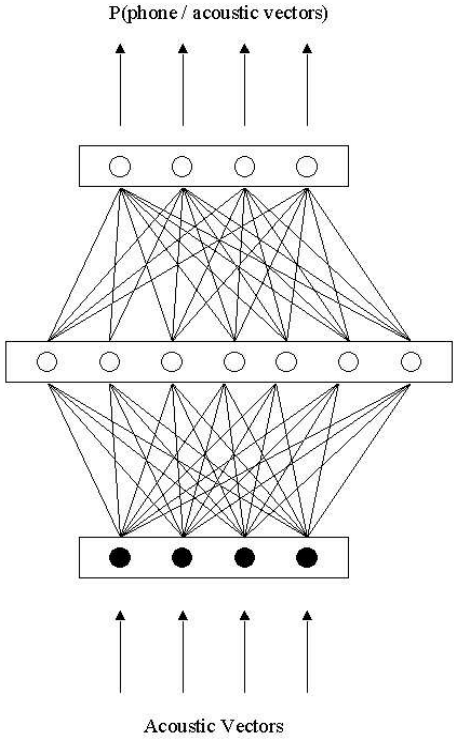

Figure 3.13: Multi-Layer Perceptron Network

learning function. MSE or the mean squared error is used as the performance function. Feed-forward networks consist of Nl layers using the DOTPROD weight function, NETSUM net input function, and the specified transfer functions.

The first layer has weights coming from the input. Each subsequent layer has a weight coming from the previous layer. All layers have biases. The last layer is the network output.

Each layer's weights and biases are initialized with INITNW.

Training is done with the specified training function. Performance is measured according to the specified performance function.

The transfer Function – Hyperbolic Tangent Sigmoid

Transfer functions calculate a layer's output from its net input. For backpropagating networks, it can be any of these three differentiable transfer functions: TANSIG, LOGSIG, or PURELIN. Of course, even at the start, the choice is cut down to logsig or tansig since Purelin is for "pure linearity" applications and is inappropriate for our purposes. Between the Logarithmic Sigmoid function (logsig) and tansig, tansig is chosen mainly because it takes an input matrix of net input (column) vectors and returns each element squashed between -1 and 1 while logsig returns it squashed between 0 and 1. (TANSIG is named after the hyperbolic tangent which has the same shape.) The purposes of choosing a wider range is to give the network much space for placing learned outputs– as compared with 0 to 1, -1 to 1 would make it easier for the network to divide substantially the range of its output to that of -1 to clearly unrecognized voices, 0 for unsure identities and for the identified voices. TANSIG(N) calculates its output according to:

$$n = \frac{2}{(1 + exp^{(-2*n)}) - 1} \tag{3.3.2}$$

This is mathematically equivalent to TANH(N). It differs in that it runs faster than the MATLAB implementation of TANH, but the results can have very small numerical differences. This function is a good trade off for neural networks, where speed is important and the exact shape of the transfer function is not.

The Training Function – Gradient descent w/momentum and adaptive learning rate backpropagation

The training function can be any of the backpropagation training functions such as
traingd - Gradient descent backpropagation,
traingdm - Gradient descent w/momentum backpropagation,
traingda - Gradient descent w/adaptive learning rate backpropagation,
traingdx - Gradient descent w/momentum and adaptive learning rate backpropagation,
trainlm - Levenberg-Marquardt backpropagation.

However, TRAINGDX is chosen since a network training function that updates weight and bias values according to gradient descent momentum and, additionally, an adaptive learning rate. This is good so as to improve generalization.

Training occurs according to the TRAINGDX's training parameters shown here:
net.trainParam.epochs 1500 Maximum number of epochs to train
net.trainParam.goal 0 Performance goal
net.trainParam.lr 0.01 Learning rate
net.trainParam.lrinc 1.05 Ratio to increase learning rate
net.trainParam.lrdec 0.7 Ratio to decrease learning rate
net.trainParam.maxfail 10 Maximum validation failures
net.trainParam.maxperfinc 1.04 Maximum performance increase
net.trainParam.mc 0.9 Momentum constant.
net.trainParam.mingrad 1e-1000 Minimum performance gradient
net.trainParam.show 25 Epochs between showing progress
net.trainParam.time inf Maximum time to train in seconds

Also, TRAINGDX can train any network as long as its weight, net input, and transfer functions have derivative functions.

Backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables X. Each variable is adjusted according to the gradient

descent with momentum.

$$dX = mc \times dXprev + lr \times mc \times dperf \div dX \tag{3.3.3}$$

where dXprev is the previous change to the weight or bias.

For each epoch, if performance decreases toward the goal, then the learning rate is increased by the factor lrinc. If performance increases by more than the factor maxperfinc, the learning rate is adjusted by the factor lrdec and the change, which increased the performance, is not made. Training stops when any of these conditions occur:

1) The maximum number of EPOCHS (repetitions) is reached.

2) The maximum amount of TIME has been exceeded.

3) Performance has been minimized to the GOAL.

4) The performance gradient falls below MINGRAD.

5) Validation performance has increase more than MAXFAIL times since the last time it decreased (when using validation).

The Learning Function – Gradient descent w/momentum weight/bias learning function

The learning function BLF can be either of the backpropagation learning functions such as LEARNGD, or LEARNGDM. Learngdm is chosen as opposed to learngd because has no momentum attribute. Learning occurs according to LEARNGDM's learning parameters, shown here with their default values.

LP.lr - 0.01 - Learning rate

LP.mc - 0.9 - Momentum constant

LEARNGDM calculates the weight change dW for a given neuron from the neuron's input P and error E, the weight (or bias) learning rate LR, and momentum constant MC, according to gradient descent with momentum:

$$dW = mc \times dWprev + (1 - mc) \times lr \times gW \tag{3.3.4}$$

The previous weight change dWprev is stored and read from the learning state LS.

The Performance Function – Mean Squared Error

MSE is a network performance function. It measures the network's performance according to the mean of squared errors.

The Weight Function - Dot product weight function

Weight functions apply weights to an input to get weighted inputs.

The Input Functions – Sum Net Input Function

Net input functions calculate a layer's net input by combining its weighted inputs and biases.

Network Initialization – Nguyen-Widrow layer initialization function

INITNW is a layer initialization function which initializes a layer's weights and biases according to the Nguyen-Widrow initialization algorithm. This algorithm chooses values in order to distribute the active region of each neuron in the layer evenly across the layer's input space.

The Nguyen-Widrow method generates initial weight and bias values for a layer so that the active regions of the layer's neurons will be distributed roughly evenly

over the input space. Advantages over purely random weights and biases are:

(1) Few neurons are wasted (since all the neurons are in the input space).

(2) Training works faster (since each area of the input space has neurons).

3.3.3 Testing/Simulation

For the testing phase, the network is simulated – the test voice features are input to the network and the network returns a vector of size 15 whose elements range from -1 to 1. The speaker whose associated node has lighted up (i.e. has the highest value) is identified as the one who owns the test input voice only if this highest value is greater than or equal to the threshold .6. The system then outputs the speaker name, otherwise, it flashes an error message for an unrecognized speaker which means that the speaker is not part of the registered set.

Chapter 4

Experiments

4.1 Environment

Our system runs under the Windows Operating System and uses the Matlab Graphics for its own user interface. The environment under which simulation is executed is below:

Hardware : PC Celeron MMX (Physical memory 64M , 266MHz)
Software : Windows 98 Operating System
User Interface : Matlab 5.0 Interface
Swap Area : Swap File system 16M
Language : Matlab 5.0
Compiler : Visual C++
Debugger : Mideva

4.2 Specifications

The system is trained to recognize 15 speakers all of whom are required to record 20 voice samples for each of the 5 words in the training set consisting of "one", "two", "eight", "nine", and "ten". The voice files have to be recorded in these specifications:

Sampling rate : 11,025 Hz

Format : Pulse Code Modulation (PCM) 16-bit mono

The system computed the Mel-Frequency coefficients of 1500 voice files and these coefficients in turn were the inputs to the training of the neural network. The network was trained for 1500 epochs for a minimum gradient of 1e - 1000 and a goal of 0. Also, the voice files for the testing phase are required to be in the same format as of the training voice samples. The system then computes the coefficients of this input file and then simulates the network with these coefficients as input. It will then display the full name (as upon entered) of the recognized speaker (or an "unrecognized speaker" error message, otherwise).

4.3 Experimental Results

Four speaker identification tests were performed each with a different number of speakers included in the speaker set (except for the second and third). The first speaker set consisted of 3 people namely Joel, Lala, and Shiela and the training was done in 3000 epochs. Over-all recognition rate was 92%. The results for the first test were as follows.

	One	Two	Eight	Nine	Ten	Total	Rating
Joel	$\frac{8}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{48}{50}$	96%
Lala	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{46}{50}$	92%
Shiela	$\frac{7}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{44}{50}$	88%

Table 4.1: Results of 150 Simulations of the 1^{st} Network

The second and third speaker identification tests used the same speaker set which

consisted of 4 speakers: Joel, Lala, Francis, and Shiela. Training for the second test was done in 1500 epochs. Training for the third test was done twice, both in 1000 epochs. But the main difference between the tests was in the manner by which the neural network was trained. For the first and second trainings, the inputs to the neural network were sequential. In the first training, the inputs were always Joel-Lala-Shiela, Joel-Lala-Shiela, and so on for each utterance sample. In the second training, the inputs were in the sequence Joel-Lala-Francis-Shiela. For the third training, the researchers tried a different technique. The inputs to the network were given in random order – meaning the two trainings for the third network had different sequences. The first time, the sequence was Joel-Lala-Francis-Shiela. The second time, the input sequence was Shiela-Lala-Francis-Joel. The results, however, were the same. Both the second and third tests yielded 87% over-all performance and the results are in the table below.

	One	Two	Eight	Nine	Ten	Total	Rating
Joel	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{10}{10}$	$\frac{48}{50}$	96%
Lala	$\frac{7}{10}$	$\frac{5}{10}$	$\frac{10}{10}$	$\frac{6}{10}$	$\frac{7}{10}$	$\frac{35}{50}$	70%
Francis	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{10}{10}$	$\frac{50}{50}$	100%
Shiela	$\frac{8}{10}$	$\frac{6}{10}$	$\frac{8}{10}$	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{41}{50}$	82%

Table 4.2: Results of 200 Simulations of the 2^{nd} and\or 3^{rd} Network

Since there was no significant output difference for the second and and third tests, the researchers decided to stick with the sequential inputs since it is easier and faster to implement.

For the fourth and final test, the speaker set used consisted of 15 speakers. Training was done in 1500 epochs only. This was done to lessen the amount of time for

training. For the 4 speakers alone in the second and third tests, each training took 12 hours to finish. For the fourth test, training took 150 hours or 6.25 days. This action, however, diminished the identification performance of the network to only 59%.

	One	Two	Eight	Nine	Ten	Total	Rating
Lala	$\frac{8}{10}$	$\frac{3}{10}$	$\frac{10}{10}$	$\frac{5}{10}$	$\frac{6}{10}$	$\frac{32}{50}$	64%
Joel	$\frac{3}{10}$	$\frac{6}{10}$	$\frac{4}{10}$	$\frac{8}{10}$	$\frac{10}{10}$	$\frac{31}{50}$	62%
Shiela	$\frac{2}{10}$	$\frac{3}{10}$	$\frac{4}{10}$	$\frac{5}{10}$	$\frac{6}{10}$	$\frac{20}{50}$	40%
Francis	$\frac{8}{10}$	$\frac{8}{10}$	$\frac{8}{10}$	$\frac{6}{10}$	$\frac{9}{10}$	$\frac{39}{50}$	78%
Didi	$\frac{10}{10}$	$\frac{7}{10}$	$\frac{6}{10}$	$\frac{10}{10}$	$\frac{7}{10}$	$\frac{40}{50}$	80%
Vince	$\frac{3}{10}$	$\frac{5}{10}$	$\frac{3}{10}$	$\frac{8}{10}$	$\frac{6}{10}$	$\frac{25}{50}$	50%
Elaine	$\frac{8}{10}$	$\frac{8}{10}$	$\frac{9}{10}$	$\frac{10}{10}$	$\frac{5}{10}$	$\frac{40}{50}$	80%
Marlon	$\frac{10}{10}$	$\frac{4}{10}$	$\frac{5}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{29}{50}$	58%
Vanie	$\frac{5}{10}$	$\frac{2}{10}$	$\frac{3}{10}$	$\frac{5}{10}$	$\frac{8}{10}$	$\frac{23}{50}$	46%
Rico	$\frac{5}{10}$	$\frac{5}{10}$	$\frac{6}{10}$	$\frac{8}{10}$	$\frac{9}{10}$	$\frac{33}{50}$	66%
Laiza	$\frac{7}{10}$	$\frac{7}{10}$	$\frac{5}{10}$	$\frac{7}{10}$	$\frac{4}{10}$	$\frac{30}{50}$	60%
Ej	$\frac{3}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{4}{10}$	$\frac{12}{50}$	24%
Jie	$\frac{7}{10}$	$\frac{3}{10}$	$\frac{5}{10}$	$\frac{9}{10}$	$\frac{8}{10}$	$\frac{32}{50}$	64%
Lawrence	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{3}{10}$	$\frac{4}{10}$	$\frac{3}{10}$	$\frac{12}{50}$	24%
Aleth	$\frac{8}{10}$	$\frac{9}{10}$	$\frac{8}{10}$	$\frac{10}{10}$	$\frac{9}{10}$	$\frac{44}{50}$	88%

Table 4.3: Results of 750 Simulations of the 4^{th} Network

Chapter 5

Summary and Conclusion

Speaker identification is the use of a machine to identify a particular person from a spoken phrase. The theories applying to speaker identification have been covered and the features and algorithms used were presented. The scope of this work is limited to digit utterances ("one", "two", "eight", "nine", and "ten") collected from a total of 15 cooperative people and without adverse microphone or channel impairments.

In this study, a new speaker identification system was presented that uses the Mel-Warped Cepstrum, also known as Mel-Frequency Cepstral Coefficients (MFCC) to discriminate between speakers and the MultiLayer Perceptron Network (MLP) to recognize the speakers. Four speaker identification tests were performed. The first test with 3 speakers yielded 92% correct open-set speaker identification. The second and third test, both with 4 speakers, both yielded 87% correct and the third test with 15 speakers yielded 59% correct identification. The tests were done in 3000, 1500, 1000, and 1500 epochs, respectively. From this, the researchers conclude that as the number of enrollees in the training increase, so too must the number of epochs implemented for the training increase. An average human being needs more time and learning to "get-to-know" more people. So does a machine.

Appendix A

Description of the Environment and Source Codes

A.1 Environment

The program runs in the Matlab environment only. The researchers used Matlab particularly because it contains 2 toolboxes which are very essential to this project. These toolboxes are the Digital Signal Processing toolbox and the Neural Network toolbox. As the name implies, the Digital Signal Processing toolbox contains the functions the researchers primarily used in the Speech Analysis module – functions like the fft, hamming, and bartlett. Similarly, the Neural Network toolbox contains the functions used for constructing the MultiLayer Perceptron network implemented in this project; these functions are train, traingdx, sim, tansig, newff, and learngdm.

A.2 Interface Modules

A.2.1 Source Codes

The following modules of the program serve as the interface of the software.

Splash

Displays the splash screen.

Main Menu

Displays the main menu which consists of 3 choices "Train", "Test", and "Exit". When the user presses "Train", the program proceeds with the module Trainparam. When the user presses "Test", the program proceeds with another interface module, Note12. When the user presses "Exit", the program, as well as the Matlab environment is terminated.

Trainparam

Called upon choosing "Train" in the Main Menu. Gets the training parameters such as the number of enrollees, number of words in the training set, and the number of samples per word per enrollee from the user.

Get-Params

Stores the parameters given as input in Trainparam. Stores the parameters in 2 text files: the number of enrollees in numEnrollees.txt and the number of words and the number of samples per word per person in numWords.txt. Also calls the other modules which gets the other parameters from the user such as the names of the enrollees and words to be included in the word training set.

Get-Name

Called n times by the module Get-Params depending on the number of enrollees. Gets the names and log-in names of the enrollees then calls the module Store-Name to store them.

Store-Name

Also called n times depending on the number of enrollees specified. Stores the names and log-in names given as input in Get-Name. Stores the inputs in 2 text files: each name in name<enrollee number>.txt and all the log-in names in log-in.txt.

Get-Word

Called n times by the module Get-Params depending on the number of words specified. Gets each word to be included in the word training set then calls the module Store-Word to store the input.

Store-Word

Also called n times depending on the number of words. Stores the words given as input in Get-Word. Creates a text file called word-training-set.txt for the first word then simply appends the other input words to the same text file.

Note8

Displays a note to the user of what software to use for recording his voice, how to name the .WAV files to enable the program to use them, where to save the .WAV files and the format by which he should record his voice before the training. Also displays the word training set.

note12

Called when the user chooses "Train" in the Main Menu. Same as Note8 except it does not instruct the user on how to name the .WAV file generated because it is not necessary to the program anymore.

Decision

Displays the decision of the testing stage. If accept, displays the name of the speaker. If reject, displays "Sorry, but your voice does not match any of our voice models".

A.3 Speech Analysis Module

A.3.1 Source Code

Comput-Mfcc

Called n ($n = number\ of\ enrollees * nnumber\ of\ words * number\ of\ samples$) times by the Training module and called once in the Testing module. One of the most important modules because it reads the actual .WAV file, performs End Point Detection on the speech signal and computes the Mel-Frequency Cepstral Coefficients.

A.3.2 Matlab Functions Used

Wavread

function [y,Fs,bits]=wavread(file,ext)

WAVREAD Read Microsoft WAVE (".wav") sound file.

[Y,FS,BITS]=WAVREAD(FILE) returns the sample rate (FS) in Hertz and the number of bits per sample (BITS) used to encode the data in the file.

Supports multi-channel data, with up to 16 bits per sample.

NOTE: This file reader only supports Microsoft PCM data format. It also does not support wave-list data.

Hamming

function w = hamming(n-est,sflag)

HAMMING Hamming window.

W = HAMMING(N) returns the N-point symmetric Hamming window in a column vector.

FFT

FFT Discrete Fourier transform.

FFT(X) is the discrete Fourier transform (DFT) of vector X. If the length of X is a power of two, a fast radix-2 fast-Fourier transform algorithm is used. If the length of X is not a power of two, a slower non-power-of-two algorithm is employed. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension.

Bartlett

function w = bartlett(n-est)

BARTLETT Bartlett window.

W = BARTLETT(N) returns the N-point Bartlett window.

A.4 Classifier Modules

A.4.1 Source Codes

The following modules implement the actual training of the neural network and the simulation for the testing.

Training

Calls the Speech Analysis Module n times first before executing. Concatenates all generated coefficients in one vector. Performs the actual training of the MultiLayer Perceptron Network. Informs the user of successful completion when finished.

Testing

Gets the .WAV input file first then calls Comput-mfcc to compute the Mel-Frequency Cepstral Coefficients. Performs the simulation of the MultiLayer Perceptron Network for the testing then passes the result to the Decision module.

A.4.2 Matlab Functions Used

Newff

function net = newff(pr,s,tf,btf,blf,pf)

NEWFF Create a feed-forward backpropagation network.

Syntax

net = newff(PR,[S1 S2...SNl],TF1 TF2...TFNl,BTF,BLF,PF)

Description

NEWFF(PR,[S1 S2...SNl],TF1 TF2...TFNl,BTF,BLF,PF) takes,

PR - Rx2 matrix of min and max values for R input elements.

Si - Size of ith layer, for Nl layers.

TFi - Transfer function of ith layer, default = 'tansig'.

BTF - Backprop network training function, default = 'trainlm'.

BLF - Backprop weight/bias learning function, default = 'learngdm'.

PF - Performance function, default = 'mse'.

and returns an N layer feed-forward backprop network.

The transfer functions TFi can be any differentiable transfer function such as TANSIG, LOGSIG, or PURELIN.

The training function BTF can be any of the backprop training functions such as TRAINLM, TRAINBFG, TRAINRP, TRAINGD, etc.

The learning function BLF can be either of the backpropagation learning functions such as LEARNGD, or LEARNGDM.

The performance function can be any of the differentiable performance functions such as MSE or MSEREG.

Algorithm

Feed-forward networks consist of Nl layers using the DOTPROD weight function, NETSUM net input function, and the specified transfer functions.

The first layer has weights coming from the input. Each subsequent layer has a weight coming from the previous layer. All layers have biases. The last layer is the network output.

Each layer's weights and biases are initialized with INITNW.

Adaption is done with ADAPTWB which updates weights with the specified learning function. Training is done with the specified training function. Performance is measured according to the specified performance function.

Tansig

function a = tansig(n,b)

TANSIG Hyperbolic tangent sigmoid transfer function.

Syntax

A = tansig(N)

info = tansig(code)

Description

TANSIG is a transfer function. Transfer functions calculate a layer's output from its net input.

Algorithm

TANSIG(N) calculates its output according to:

n = 2/(1+exp(-2*n))-1

This is mathematically equivalent to TANH(N). It differs in that it runs faster than the MATLAB implementation of TANH, but the results can have very small numerical differences. This function is a good trade off for neural networks, where speed is important and the exact shape of the transfer function is not.

Traingdx

function [net,tr] = traingdx(net,Pd,Tl,Ai,Q,TS,VV,TV)

TRAINGDX Gradient descent w/momentum and adaptive lr backpropagation.

Syntax

[net,tr] = traingdx(net,Pd,Tl,Ai,Q,TS,VV) info = traingdx(code)

Description

TRAINGDX is a network training function that updates weight and bias values according to gradient descent momentum and an adaptive learning rate.

Training occurs according to the TRAINGDX's training parameters shown here with their default values: net.trainParam.epochs 10 Maximum number of epochs to train

net.trainParam.goal 0 Performance goal

net.trainParam.lr 0.01 Learning rate

net.trainParam.lr-inc 1.05 Ratio to increase learning rate

net.trainParam.lr-dec 0.7 Ratio to decrease learning rate

net.trainParam.max-fail 5 Maximum validation failures

net.trainParam.max-perf-inc 1.04 Maximum performance increase

net.trainParam.mc 0.9 Momentum constant.

net.trainParam.min-grad 1e-10 Minimum performance gradient

net.trainParam.show 25 Epochs between showing progress

net.trainParam.time inf Maximum time to train in seconds

Dimensions for these variables are:

Pd - NoxNixTS cell array, each element Pi,j,ts is a DijxQ matrix.

Tl - NlxTS cell array, each element Pi,ts is an VixQ matrix.

Ai - NlxLD cell array, each element Aii,k is an SixQ matrix.

Where

Ni = net.numInputs

Nl = net.numLayers

LD = net.numLayerDelays

Ri = net.inputsi.size

Si = net.layersi.size

Vi = net.targetsi.size

Dij = Ri * length(net.inputWeightsi,j.delays)

If VV is not [], it must be a structure of validation vectors,

VV.PD - Validation delayed inputs.

VV.Tl - Validation layer targets.

VV.Ai - Validation initial input conditions.

VV.Q - Validation batch size.

VV.TS - Validation time steps.

which is used to stop training early if the network performance on the validation vectors fails to improve or remains the same for MAX-FAIL epochs in a row.

Algorithm

TRAINGDX can train any network as long as its weight, net input, and transfer functions have derivative functions.

Backpropagation is used to calculate derivatives of performance PERF with respect to the weight and bias variables X. Each variable is adjusted according to the gradient descent with momentum.

dX = mc*dXprev + lr*mc*dperf/dX

where dXprev is the previous change to the weight or bias.

For each epoch, if performance decreases toward the goal, then the learning rate is increased by the factor lr-inc. If performance increases by more than the factor max-perf-inc, the learning rate is adjusted by the factor lr-dec and the change, which increased the performance, is not made.

Training stops when any of these conditions occur:

1) The maximum number of EPOCHS (repetitions) is reached.

2) The maximum amount of TIME has been exceeded.

3) Performance has been minimized to the GOAL.

4) The performance gradient falls below MINGRAD.

5) Validation performance has increase more than MAX-FAIL times since the last time it decreased (when using validation).

Train

function [net,tr]=train(net,P,T,Pi,Ai,VV,TV)

TRAIN Train a neural network.

Syntax

[net,tr] = train(NET,P,T,Pi,Ai)

[net,tr] = train(NET,P,T,Pi,Ai,VV,TV)

Description

TRAIN trains a network NET according to NET.trainFcn and NET.trainParam.

TRAIN(NET,P,T,Pi,Ai) takes,

NET - Network.

P - Network inputs.

T - Network targets, default = zeros.

Pi - Initial input delay conditions, default = zeros.

Ai - Initial layer delay conditions, default = zeros.

and returns,

NET - New network.

TR - Training record (epoch and perf).

TRAIN(NET,P,T,Pi,Ai,VV,TV) takes optional structures of validation and test vectors,

VV.P, TV.P - Validation/test inputs.

VV.T, VV.T - Validation/test targets, default = zeros.

VV.Pi, VV.Pi - Validation/test initial input delay conditions, default = zeros.

VV.Ai, VV.Ai - Validation/test layer delay conditions, default = zeros.

The validation vectors are used to stop training early if further training on the primary vectors will hurt generalization to the validation vectors. Test vector performance can be used to measure how well the network generalizes beyond primary and validation vectors. If VV.T, VV.Pi, or VV.Ai are set to an empty matrix or cell array, default values will be used. The same is true for VT.T, VT.Pi, VT.Ai.

Algorithm

TRAIN calls the function indicated by NET.trainFcn, using the adaption parameter values indicated by NET.trainParam.

Typically one epoch of training is defined as a single presentation of all input vectors to the network. The network is then updated according to the results of all those presentations.

Training occurs until a maximum number of epochs occurs, the performance goal is met, or any other stopping condition of the function NET.trainFcn occurs.

Some training functions depart from this norm by presenting only one input vector (or sequence) each epoch. An input vector (or sequence) is chosen randomly each epoch from concurrent input vectors (or sequences). NEWC and NEWSOM return networks that use TRAINWB1, a training function that does this.

Sim

function [Y,Pf,Af]=sim(net,P,Pi,Ai)

SIM Simulate a neural network.

Syntax

[Y,Pf,Af] = sim(net,P,Pi,Ai)

[Y,Pf,Af] = sim(net,Q TS,Pi,Ai)

[Y,Pf,Af] = sim(net,Q,Pi,Ai)

Description

SIM simulates neural networks.

[Y,Pf,Af] = SIM(net,P,Pi,Ai) takes,

NET - Network.

P - Network inputs.

Pi - Initial input delay conditions, default = zeros.

Ai - Initial layer delay conditions, default = zeros.

and returns:

Y - Network outputs.

Pf - Final input delay conditions.

Af - Final layer delay conditions.

[Y,Pf,Af] = SIM(net,Q TS,Pi,Ai)

is used for networks which do not have an input, such as Hopfield networks when cell array notation is used.

[Y,Pf,Af] = SIM(net,Q,Pi,Ai) is used for networks which do not have an input, such as Hopfield networks when matrix notation is used.

Algorithm

SIM uses these properties to simulate a network NET.

NET.numInputs, NET.numLayers

NET.outputConnect, NET.biasConnect

NET.inputConnect, NET.layerConnect

These properties determine the network's weight and bias values, and the number of delays associated with each weight:

NET.inputWeightsi,j.value

NET.layerWeightsi,j.value

NET.layersi.value

NET.inputWeightsi,j.delays

NET.layerWeightsi,j.delays

These function properties indicate how SIM applies weight and bias values to inputs to get each layer's output:

NET.inputWeightsi,j.weightFcn

NET.layerWeightsi,j.weightFcn

NET.layersi.netInputFcn

NET.layersi.transferFcn

Learngdm

function [dw,ls] = learngdm(w,p,z,n,a,t,e,gW,gA,d,lp,ls)

LEARNGDM Gradient descent w/momentum weight/bias learning function.

Syntax

[dW,LS] = learngdm(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

[db,LS] = learngdm(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)

info = learngdm(code)

Description

LEARNGDM is the gradient descent with momentum weight/bias learning function.

LEARNGDM(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

takes several inputs,

W - SxR weight matrix (or Sx1 bias vector).

P - RxQ input vectors (or ones(1,Q)).

Z - SxQ weighted input vectors.

N - SxQ net input vectors.

A - SxQ output vectors.

T - SxQ layer target vectors.

E - SxQ layer error vectors.

gW - SxR gradient with respect to performance.

gA - SxQ output gradient with respect to performance.

D - SxS neuron distances.

LP - Learning parameters, none, LP = [].

LS - Learning state, initially should be = [].

and returns,

dW - SxR weight (or bias) change matrix.

LS - New learning state.

Learning occurs according to LEARNGDM's learning parameters, shown here with their default values.

LP.lr - 0.01 - Learning rate

LP.mc - 0.9 - Momentum constant

LEARNGDM(CODE) returns useful information for each CODE string:

'pnames' - Returns names of learning parameters.

'pdefaults' - Returns default learning parameters.

'needg' - Returns 1 if this function uses gW or gA.

Algorithm

LEARNGDM calculates the weight change dW for a given neuron from the neuron's input P and error E, the weight (or bias) learning rate LR, and momentum constant MC, according to gradient descent with momentum:

dW = mc*dWprev + (1-mc)*lr*gW

The previous weight change dWprev is stored and read from the learning state LS.

Bibliography

[1] Yousef A. Alotaibi and M.M. Shahsavari. Speech recognition. *IEEE*, Feb '98:23–28, 1998.

[2] Paola Salmi Carlo Caini and Alessandro Vanelli Coralli. Cd-hmm algorithm performance for speaker identification on an italian database. *IEEE*, Sept '97:9–12, 1997.

[3] Shaji Hayakawa and Fumitada Itakura. Text-dependent speaker recognition using the information in the higher frequency band. *IEEE*, Feb '94:137–140, 1994.

[4] Li Liu Jialong He and Gunther Palm. Speaker identification using hybrid lvq-slp networks. *IEEE*, Apr '95:2052–2055, 1995.

[5] Dahong Xie Ke Chen and Huisheng Chi. Speaker identification based on the time-delay hierarchical mixture of experts. *IEEE*, Apr '95:2062–2066, 1995.

[6] J.E. Porter L.G. Bahler and A.L. Higgins. Improved voice identification using a nearest-neighbor distance measure. *IEEE*, Feb '94:321–323, 1994.

[7] Richard J. Mammone. Robust speaker recognition. *IEEE*, Sept '96:58–71, 1996.

[8] Ismail Shahin and Nazeih Botros. Speaker identification using dynamic time warping with stress compensation technique. *IEEE*, Dec '98:65–68, 1998.