**5. DESIGN OF SPEECH RECOGNITION SYSTEM**

**5.1. Overview**

In this chapter the design of speech recognition system have been performed. For feature extraction of speech Spectrogram, Linear Predictive Coding (LPC) and Mel-Frequency Cepstral Coefficient (MFCC) techniques were used. The basic structure of speech recognition system is given. The flowcharts of the designed program have been described. For classification of the speech neural networks has been applied.

**5.2. General structure of the system**

The general structure of the speech recognition program is shown in figure 5.1. The input of the system is the speech signal. After receiving speech signal the preprocessing of the speech is started. The preprocessing includes Denoising and End points detection operations. After preprocessing the signal is sent to the feature extraction block. In this thesis three methods- LPC, MFCC, and Spectrogram are used for feature extraction. These features are used as an input to the neural network in the next block for classification. Finally the decision is taken if there is matching or no in the last block.



Figure 5.1: General structure of the program.

**5.3. Flowcharts of features extraction methods**

Three methods (LPC, MFCC, and Spectrogram) were used in this thesis for feature extraction, each of them was explained in chapter 3, and the flowcharts of these three methods were given in this section. Feature extraction phase is started after Denoising and End points of the speech signal are detected as mentioned above. Starting with LPC method that the flowchart of it is illustrated in figure 5.2, the flowchart shows the steps that followed to get LPC coefficients in this thesis.

The signal is framed with 240 samples for every frame, the frames are saved in a matrix as columns so the number of columns represents the number of frames, after that windowing is applied on every column of the matrix, and then LPC function of 12 order is applied on every column also, now every column has 12 coefficients, and finally the coefficients are rearranged in a one column to use them as input to the neural network in the classification phase. The number of coefficients in this thesis was 420 for LPC method.

The flowchart of MFCC method is shown in figure 5.3. The steps of MFCC method are started with framing and windowing as in LPC method but after windowing cepstral analysis are started instead of predicting analysis.

The number of coefficients that were obtained in this thesis was 613 coefficients for this method and these coefficients were used as input to the neural network in classification phase.

The last method used in this thesis was Spectrogram. The steps of this method are like the previous one, but the cepstral analysis is not applied on the signal after FFT, and the mapping of the spectral amplitude to a grey level is applied, and then Spectrogram coefficients are gotten. Figure 5.4 shows the flowchart of spectrogram method.



Figure 5.2: Flowchart of LPC method.



Figure 5.3: Flowchart of MFCC method.



Figure 5.4: Flowchart of spectrogram method.

**5.4. Implementation of features extraction methods**

As mentioned before feature extraction signal processing is carried out. Signal preprocessing includes Denoising and end points detection. Zero Crossing based Endpoint Detection algorithm was used in this thesis. Figure 5.5 depicts source signal and results of end points detection operation.

  

 (a) (b)

Figure 5.5: Operation of end points detection (a) Source signal, (b) End points detected signal.

The process of features extraction of the speech signal starts with: framing the speech signal and windowing it. After that, one of the three methods (LPC, MFCC, and Spectrogram) will be applied to extract the coefficients from the speech signal.

Framing is done because the speech signal is quasi-stationary (fairly stationary). In this thesis the spoken word has been framed to many frames and every frame is 30 msec. Because of the used sampling frequency 8 KHz, every frame includes 240 samples in this thesis. The figure 5.6 depicts the framing result for the word “one”.



Figure 5.6: A one frame of the word “one”.

After these operations the windowing is applied and every frame is multiplied with a window function $w(n)$ with length $N$, where $N$ is the number of samples in each frame (w(n)\*y(n), here y(n) is frame signal). Windowing is used to avoid problems due to truncation of the signal. Hamming window is usually used in speech recognition systems for windowing operation. Figure 5.7 shows the obtained signal from the operation of multiplying the frame signal by hamming window.

 $w\left(n\right)=0.54-0.46\cos(\left(\frac{2πn}{N-1}\right)) ,  0\leq n\leq N-1$ (5.1)



Figure 5.7: Frame signal after applying hamming window.

Now it is a time to estimate the coefficients from the obtained speech signal. Using feature extraction techniques (LPC, MFCC and Spectrogram methods) the coefficients are determined for each frame. The obtained coefficients from each frame are combined in order to get feature vector for the word. The above procedures are applied on all the spoken words.

**5.4.1. Linear Predictive Coding (LPC)**

Linear predictive coding is a technique used mostly in speech processing to estimate basic speech parameters like pitch formants and spectral envelope of the speech signal, in compressed form, using the information of linear predictive model. LPC is one of the most useful methods for encoding good quality speech at a low bit rate. The coefficients are generated by the linear combination of the past speech samples using the autocorrelation or auto variance method.

 $ \tilde{x}\_{(n)}=a\_{1}x\_{(n-1)}+a\_{2}x\_{(n-2)}+. . . . . .+a\_{p}x\_{(n-p)}$ (5.2)

Where $\tilde{x}\_{(n)}$ is the predicted of $x\_{(n)}$ based on the summation of past samples, $a\_{i}$ is the linear prediction coefficients, $p$ is the number of coefficients and $n$ is the samples.

LPC coefficients can be estimated by applying some procedures on the speech signal. These procedures started with applying autocorrelation on the windowed frames. Every windowed frame is autocorrelated by p`th order by applying the MATLAB code bellow:

p = 12;

x1 = x;

x2 = x;

N = length(x);

for k = 1: p+1

b = sum(x1.\*x2);

B(k,1) = b;

x1 = zeros(N+k,1);

for i = 1:N

x1(i) = x(i);

end

x2 = zeros(N+k,1);

j = 1;

for i = k+1:N+k

x2(i) = x(j);

j = j + 1;

end

end

where x is a data vector of a one frame, p is the order of the correlation coefficients and B is the correlation coefficients. The MATLAB code above is equal to the embedded MATLAB function:

B = xcorr(x);

B = B(N:(N+P));

The autocorrelation coefficients have been calculated and then Levinson-Durbin algorithm is applied to calculate the final LPC coefficients.

At the beginning, the first coefficient of the first column is calculated by applying equation (5.3):

 $A\_{(i,i)}={\left[\sum\_{k=0}^{i}\begin{matrix}B\_{k+2}&A\_{(i-k,i)}\end{matrix}\right]}/{E\_{i}}$ (5.3)

 *E(1) = B(1)*

where A is the matrix of the LPC coefficients, B is the vector of the autocorrelation coefficients, and E is the vector of the energy of the prediction error.

Then E(2) – E(p) is calculated by applying equation (5.4):

 *E(i+1) = (1-A(i,i)^2) \* E(i)* (5.4)

In the second stage the second coefficient of the second column is calculated according to equation (5.3) and the rest coefficients of the same (second) column are calculated according to equation (5.5)

 $A\_{(i,j)}=A\_{(i,j-1)}+A\_{(j,j)}\*A\_{(j-i,j-1)}$ (5.5)

And so on. These processes are repeated until all the coefficients are calculated.

The last column of the matrix A represents the coefficients of LPC for the one frame. The result of computing LPC coefficients are given in table below. The MATLAB code bellow is to apply the above equations to get the coefficients of the windowed frame:

A = zeros (p,p);

E = zeros (1,p+1);

E(1) = B(1);

for i = 1:p,

suma = 0;

for j = 1:(i-1),

suma = suma + A(j,i-1) \* B(i - j +1);

end

ki = - ( B(i+1) + suma ) / E(i);

A(i,i) = ki;

for j = 1:(i-1)

A(j,i) = A(j,i-1) + ki \* A(i-j,i-1);

end

E(i + 1) = (1 - ki ^ 2) \* E(i);

end

A

E

After applying this code on a frame the following matrix appears:



The last column represents the LPC coefficients of a one frame of a spoken word. To estimate LPC coefficients also there is an already embedded function in MATLAB this function calculates the coefficients directly:

C = lpc(x,12);

C =

 1.0000

 -1.3288

 0.3577

 -0.2748

 0.1944

 -0.2104

 0.4498

 0.0073

 -0.2682

 0.1649

 0.1435

 -0.3697

 0.1965

As seen they are equal to the last column of matrix A. lpc function calculates p+1 coefficients and always the first one is equal to 1, so, the first coefficient is cancelled and not included to the LPC coefficient set. Figure 5.8 shows the LPC coefficients for a one frame.



Figure 5.8: LPC coefficients for a one frame.

These are the coefficients for a one frame. By combining all the coefficients for all the frames of a spoken word, the LPC coefficients of a one word will be derived. Figure 5.9 shows the LPC coefficients of word “one”.



Figure 5.9: LPC coefficients for a word “one”.

**5.4.2. Mel Frequency Cepstral Coefficients (MFCC)**

It is a method used to extract the features from the speech signal. MFCC is based on human hearing perceptions. It is observed that human ear acts as a filter, these filters are non-uniformly spaced on the frequency axis (More filters in the low frequency regions and Less no. of filters in high frequency regions). So MFCC has two types of filter which are spaced linearly at low frequency below 1000 Hz and logarithmic spacing above 1000Hz.

Framing and windowing are done as explained previous section. After that the windowed frames are converted from time to frequency domain by applying FFT on them. FFT is an embedded function in MATLAB. Result of FFT of above windowed frame example is shown in figure 5.10.



Figure 5.10: Spectrum of a one windowed frame.

Then the obtained signal is filtered by a filter bank of mel filters. These filters are uniformly spaced to simulate the human auditory system. Mel filters bank are given in figure 5.11.



Figure 5.11: Mel filters bank.

The result of FFT is multiplied to Mel filter bank. After discrete cosine transform (DCT) is applied for the logged mel frames to return the signal to the time domain. Then the coefficients of MFCC are obtained. The plot of the coefficients is given in figure 5.12.



Figure 5.12: MFCC coefficients for a one frame.

And the final coefficients for all the frames are combined to represent the coefficients of a word as in figure 5.13.



Figure 5.13: MFCC coefficients for the word “one”.

**5.4.3. Spectrogram**

A spectrogram is a visual representation of the spectrum of frequencies in a sound. Spectrograms can be used to identify spoken words phonetically. They are used extensively in the development of the fields of music, sonar, radar, and speech processing, etc. The horizontal dimension corresponds to time (reading from left to right), and the vertical dimension corresponds to frequency. Sound having high amplitude has high red colour.

Spectrogram of a speech signal can be derived by taking a Fast Fourier Transform (FFT) for each frame of the speech signal. Then the rotation of plot diagram implemented to fix vertical axis as frequency and horizontal axis as amplitude. Result of rotation operation is given in figure 5.14.



Figure 5.14: Obtained signal after rotating spectrum of the windowed frame.

And then represent the amplitude using grey scale values from 0 to 255. Where 0 represents black colour and 255 represents white colour. High amplitude correspond to dark region, low amplitude correspond to white region. The vertical axis represents the frequency, the horizontal axis represents the time (number of frames for the speech signal), and the intensity of the colour represents the amplitude. The amplitude values are taken as feature values of the windowed frame. The above operations are repeated for all frames. Resulting feature vector is obtained by combining feature values of all windowed frames. Figure 5.15 shows the spectrogram of the word “one”. MATLAB has an already function that calculate spectrogram of the signal

S = spectrogram(x,w,noverlap,nfft,fs);



Figure 5.15: Spectrogram of the word “one”.

**5.5. The design of speech recognition program**

The system was developed for speech recognition based on three methods for feature extraction (LPC, MFCC, and Spectrogram) and one algorithm for pattern matching (ANN). Audionic AH-112 headphone set was used as a tool to record spoken words (were chosen randomly and listed in table 5.1) that were stored as .WAV files (by the author voice) in a special folder as a memory for the system. Sony (Vaio, CORE i7 CPU 2.2 GHz, and Windows 8 64-bit operating system) laptop was used as a device to apply the system on it by using MATLAB (R2012a) program.

|  |  |
| --- | --- |
| Seq. | Words used in Speech Recognition system |
| 1 | Blue |
| 2 | Class |
| 3 | Do |
| 4 | Down |
| 5 | Get |
| 6 | Give |
| 7 | Go |
| 8 | Hello |
| 9 | Last |
| 10 | Left |
| 11 | Name |
| 12 | Number |
| 13 | Out |
| 14 | Red |
| 15 | Round |
| 16 | Run |
| 17 | Stop |
| 18 | Test |
| 19 | Up |
| 20 | Voice |
| 21  | White |
| 22 | One |
| 23 | Two |
| 24 | Three |
| 25 | Four |
| 26 | Five |
| 27 | Six |
| 28 | Seven |
| 29 | Eight |
| 30 | Nine |

Table 5.1: Words that used in speech recognition system.

The instruction that was used for recording the words in MATLAB was:

Y = wavrecord (N, Fs);

Where *N* is the number of samples per word and *Fs* is the sampling frequency, 8 KHz sampling frequency was used in this thesis, and to store these words in the memory this instruction was used:

Wavwrite (Y, “filename”);

Then to read a wav file this instruction was used:

[Y, Fs] = wavread (“filename”);

The developed program has a Graphical User Interface (GUI), illustrated in figure 5.16. The program is .m MATLAB file that consists of one function to create the main displaying window, and one main function that call many functions to complete the recognizing.



Figure 5.16: Speech recognition system.

The MATLAB code for making this display interface is:

function Speech Recognition system

f = figure('Visible','on','Position',[60,150,800,585]);

choose\_method = uicontrol('Style','text','String','Choose the method of

recognizing','Position',[520,520,240,20],...

'BackgroundColor',[0.8,0.8,0.8],'FontSize',12,...

'ForegroundColor','k');

method = uicontrol('Style','popupmenu',...

'String',{'Select one','LPC& ANN','MFCC &ANN',...

'Spectrogram & ANN'},'Position',[550,410,200,100],...

'BackgroundColor',[0.9,0.9,0.9],'FontSize',12,...

'ForegroundColor','k','Callback',{@method\_Callback});

trnng = uicontrol('Style','pushbutton','FontSize',12,'String',...

'Train the Network','Position',[550,340,200,70],...

'Callback',{@train\_Callback});

test = uicontrol('Style','pushbutton','FontSize',12,...

'String','Test the Network','Position',[550,220,200,70],...

'Callback',{@test\_Callback});

qquit = uicontrol('Style','pushbutton','String','Quit','FontSize',12,...

'Position',[550,100,200,70],'Callback',{@quit\_Callback});

noise = uicontrol('Visible','off','Style','text','String','S/N Ratio = ',

'Position',[90,525,160,20],'BackgroundColor',[0.8,0.8,0.8],

'FontSize',12,'ForegroundColor','k');

noise\_ratio = uicontrol('Visible','off','Style','popupmenu',...

'Position',[220,530,90,20],'String',{'No Noise',...

 '30 dB','25 dB','20 dB','15 dB','10 dB','5 dB'},...

'BackgroundColor',[0.9,0.9,0.9],...

'ForegroundColor','k','FontSize',12,...

'Callback',{@comput\_noise\_Callback});

OK = uicontrol('Visible','off','Style','pushbutton','String','Ok',...

'FontSize',12,'Position',[320,523,40,25],...

'Callback',{@ok\_Callback});

fig = axes('Units','points','Position',[50,50,320,300]);

set([choose\_method,method,trnng,test,qquit,fig,noise,noise\_ratio,OK],'Units','normalized');

% Assign the GUI a name to appear in the window title.

set(f,'Name','Speech Recognition program')

% Move the GUI to the center of the screen.

movegui(f,'center')

end

**5.5.1. The selection of the method for feature extraction**

As seen in figure 5.16, there are many buttons, starting with “choose the method of recognizing”, and as mentioned previously three methods are there in the popup menu button and the user must choose the method that he want from this button when he starts with the program. Training button and testing button are not work if the user didn’t selects a method from the popup menu button, and an error will appear to inform the user that he didn’t chooses a method.

The MATLAB code for popup menu button is:

functionmethod\_Callback(source,~)

 current\_method = 0;

str = get(source, 'String');

val = get(source,'Value');

% current method.

switchstr{val};

case'Select one'

 current\_method = 0;

case'LPC & ANN'

 current\_method = 1;

case'MFCC & ANN'

 current\_method = 2;

case'Spectrogram & ANN'

 current\_method = 3;

end

end

**5.5.2. Training the system**

The second button is “Train the Network”. After choosing the method and clicking on this button the system starts to extract the features of all the spoken words that stored in the memory of the system (folder of the recorded spoken words), saves these features in a matrix to be the input of the neural network, and starts to train the network with these features to give every word a unique pattern, so, when the user chooses a word for testing, the network gives this word the same pattern that has been given to the same word stored in the system.

As mentioned previously, the first process in this system is End Points Detection. In this thesis Zero\_Crossing algorithm has been used for End Points Detection and the MATALB code for this algorithm is:

function [mag,pts] = locatespeech(sig,N,step,fs)

% 1) Remove DC offset

sig\_no\_dc = filter([1, -0.97], 1, sig);

% 2) Compute Avg. Mag and Zero-X rate of sig

m = avgmag(sig\_no\_dc,N,step);

z = zero\_crossing(sig\_no\_dc,N,step);

% 3) Compute mag and zero-crossing of noise (first 100 msec of sig) - already computed,

% just cut it out of m and z above

hundredmsec\_rel = round((fs\*.2)/step); % # samples that equals 100ms

% Ends of these may be corrupted due to zero padding -- chop off N/step

% samples from each side

chop = ceil((N/2-step)/step); % round up for safety

noise\_m = m(2+chop:hundredmsec\_rel-chop);

noise\_z = z(2+chop:hundredmsec\_rel-chop);

% Compute means and st. deviations of each, so we can develop thresholds.

noise\_m\_mean = mean(noise\_m);

noise\_m\_std =std(noise\_m);

noise\_z\_mean = mean(noise\_z);

noise\_z\_std =std(noise\_z);

% Set lower thresholds

fudge = 5;

ITL = noise\_m\_mean + fudge\*noise\_m\_std;

IZCT = noise\_z\_mean + fudge\*noise\_z\_std;

% Define upper threshold for avg mag.

ITU = 3.2\*noise\_m\_mean; % since std<< mean, twice the mean should cover it

% find place where sig consistently tops ITU.

start = 3; % since window goes back two spots, start at 3rd sample

avg\_last3pts = 0; % ITU won't be topped in the first 3 pts, so initialize to 0

while avg\_last3pts < ITU,

start = start + 1;

 avg\_last3pts = (m(start) + m(start-1) + m(start-2))/3;

end

% move backwards to find where we first go under ITL

while m(start) > ITL,

start = start - 1;

end

% See if need to move start back due to zero-crossing

below\_izct\_count = 0;

first\_below = -999;

if start > 25

fori = start:-1:start-25,

if z(i) < IZCT

below\_izct\_count = below\_izct\_count + 1;

iffirst\_below == -999

first\_below = i;

end

end

end

ifbelow\_izct\_count>= 3

start = first\_below;

end

end

% Now do the same process backwards for the end;

endpt = length(m)-2;

avg\_last3pts = 0; % threshhold won't occur in first 3 pts, so initialize to 0

while avg\_last3pts < ITU,

endpt = endpt - 1;

 avg\_last3pts = (m(endpt) + m(endpt+1) + m(endpt+2))/3;

end

% move forwards to find where we first go under ITL

while m(start) > ITL,

endpt = endpt + 1;

end

% See if need to move start back due to zero-crossing

below\_izct\_count = 0;

first\_below = -999;

if (endpt-length(z)) > 25

fori = endpt:1:endpt+25,

if z(i) < IZCT

below\_izct\_count = below\_izct\_count + 1;

iffirst\_below == -999

first\_below = i;

end

end

end

ifbelow\_izct\_count>= 3

endpt = first\_below;

end

end

% Return values (multiply endpoints by step so that it is scaled

% appropriately for the actual signal

mag = m(start:endpt);

pts = [start\*step endpt\*step];

end

**5.5.3. Speech recognition using LPC**

When the user chooses LPC & ANN as a method, this means that current\_method = 1 as shown in the code of the popup menu button above. The first step to extract features is framing. Every frame consists of 240 samples and 50% overlapping between every two adjacent frames. The MATLAB codethat was used for framing is:

% Framing.

 l = length(xpre); % xpre is the speech signal after applying End Point

% Detection algorithm.

 n = 240; % frame size.

 m = 120; % overlapping (50%)

frames = floor((l-n)/m) + 1;

for I = 1:n

for J = 1:frames

M(I, J) = xpre(((J - 1) \* m) + I);

end

end

This code is used to divide the speech signal into frames, every frame is 240 samples, this mean every frame is 30 msec (sampling frequency is 8 KHz), and then putting these frames in an array (M as shown above) every column in it represents a frame, so, the number of the columns represents the number of the frames.After that windowing must be done on the frames, and every frame is multiplied with 240 samples of hamming window as in the code below:

% Hamming window for every frame.

 w = hamming(n);

fori = 1:frames

xw(:,i) = w.\* M(:,i);

end

Figure 5.17 shows a one frame of the word "one" after applying hamming window on the speech signal, and figure 5.18 shows all the frames of the word "one".



Figure 5.17: Hamming window for a one frame of the word “one”.



Figure 5.18: Hamming window of the word “one”.

After that,LPC coefficients are computed for every frame to use them as input to the neural network. The MATLAB code below was used to compute LPC coefficients:

 p = 12;

fori = 1:frames

lpc\_coef(:,i) = lpc(xw(:,i),p);

end

Where *p* is the order of coefficients for every frame, usually it is between 10 and 20. Figure 5.19 shows LPC coefficients for all frames forthe word “one”.



Figure 5.19: LPC coefficients for “one”.

Now the LPC coefficients are ready to enter to the network but before that, the process of the features extraction is repeated for all the recorded spoken words, and these features are stored in a one array, every column in it represents one word. This array is used to train the neural network.

**5.5.4. Speech recognition using MFCC**

First, framing and windowing processes are done same as in the previous method, and then Fast Fourier Transform (FFT) is applied on the windowed frames to get the spectrum for the speech signal. The MATLAB code below was used to apply the FFT on the signal that resulted after applying hamming window:

fori = 1:frames

M2(:,i) = fft(xw(:, i));

end

After that, the signal enters to a train of triangular filters of Mel scale, so the samples that have frequency less than 1000 Hz will be in the linear scale and samples that have frequency more than 1000 Hz will be in the logarithmic scale. This process was applied on the output of the FFT process by using the MATLAB function below:

mfcc = melfunc(M2);

Then cepstrum analysis is applied to get the Cepstral Coefficients of the Mel Frequency, so the MFCC are ready now to enter the neural network for training. The MATLAB code below was used to get the cepstrum of the signal:

% take the Cepsetral for the coeffetionts.

rr = dct(log(mfcc));

**5.5.5. Speech recognition using Spectrogram**

When the current\_method = 3 this means that the selected method is Spectrogram, spectrogram in MATLAB is a ready function:

 S = spectrogram(x,w,noverlap,nfft,fs);

Where x is the speech signal, w is Hamming window of 240 samples, noverlap is the number of samples that each segment overlaps (here is 50% overlapping), nfft is the FFT length (240), and fs is the sampling frequency. This function divides speech signal to number of segments depending on the number of samples in Hamming window and overlapping.

The MATLAB code below was used to compute the spectrogram of the stored spoken words as a method for feature extraction:

 w = hamming(240);

 S = spectrogram(xpre,w,120,240,fs);

**5.5.6. Neural network training**

After extract features by one of the previous three methods, now it is a time to enter these features to the neural network, and as mentioned earlier, the feature extraction process are applied on all the 30 words, and are saved in a one array, where every word is represented in a one column, this array enters to the neural network, and the network gives every column a unique pattern, in this thesis the neural network has 30 outputs (for every pattern one output). Output one is “1” logic and other outputs is “0” logic for the first pattern (first stored word), output two is “1” logic and other outputs is “0” logic for the second pattern (second stored word), and so on. Figure 5.20 shows how the network trains on the stored words:



Figure 5.20: How features enter to the neural network and how output is gotten.

Where *n* is the number of input nodes this depending on the number of extracted features from every word, and must be noted that all the words must have the same number of features, *N* is the number of nodes in hidden layer.In this thesis equation (5.6) was used to compute the number of nodes in hidden layer.The output layer nodes represent the number of trained words.



(5.6)

Where *H* is the number of hidden neurons, *N* is the number of input neurons, *M* is the number of output neurons, and *T* is the number of input data (number of input neurons multiplied by number of trained words).

The function below contains the MATLAB code that was used in this thesis to create previous neural network.

function out = trnng\_files(coeffdata,T,n\_sounds)

% create a neural network to recognize a word.

 [row,col] = size(coeffdata);

hidd = floor((row.\*col)/(5.\*(col + n\_sounds))) + 1;

net = newff(minmax(coeffdata),[hiddn\_sounds],{'logsig'

'logsig'},'traingdx');

net.performFcn = 'sse';

net.trainParam.goal = 0.1;

 net.trainParam.lr = 0.0001;

net.trainParam.show = 20;

net.trainParam.epochs = 1000;

 net.trainParam.mc = 0.025;

out = train(net,coeffdata,T);

end

The inputs variables are coeffdata, T, and n\_sounds. coeffdata is the matrix that contains the features of all the 30 words every column represent a word so 30 columns are there in it, T is the target of the network that the outputs of the network must achieve it; T is a 30 by 30 array and the values in it as explained above represent the patterns for the trained words, and n\_sounds represents the number of trained words. The output of the network is an array contains the values of the patterns and these values are nearly equal to those in the target.

The instruction (newff) is used to create the neural network, and the other instructions are used to initialize the parameters of the network. Finally the output values are stored in an array named “out” as seen above to use them in the next step to compare them with the got values from tested word to see if there is matching. After applying the function above, a window appears to show neural network and how many neurons are used in every layer, and other results related with number of epochs, time of training, performance of the network, and other results as shown in figure 5.21 below.



Figure 5.21: Neural network applied window.

**5.5.7. Testing the system**

When the user click on this button, the system asks him to select which type of recognizing he want, because the recognition in this system is divided into two types; the first one is to recognize a not trained word that mean the user chooses a word not stored in the memory of the system, and the second type is to recognize a trained word, but in this type the user has been asked to add a noise to the speech signal from the “SNR” button found on the main displaying interface window, and the explaining of every type was described in the coming paragraphs. In the beginning when the user opens the program the “SNR” button is not activated and not appeared on the main displaying interface as shown in figure 5.22, because of the first type of recognition does not use noise with the speech signal and the user just chooses the word that he wants to recognize it and then the result appears there is matching or no, but the “SNR” button and related buttons will be activated when the user chooses the second type of recognition and the plotting of the speech signal will also appears as explained later.



Figure 5.22: First display of displaying interface window.

As mentioned above when the user click on the “Test the Network” button, the command window of MATLAB program appears asking the user to enter a No. to choose any type of recognition he wants as shown in figure 5.23.



Figure 5.23: Select a type of recognizing window.

The MATLAB code below was used to ask the user about what type of recognizing he wants:

choice = input('Enter a No. to recognize a word, No. 1 for not trained words:

 No. 2 for trained words:');

if (choice == 1)

 [filename, pathname] = uigetfile('C:\Users\Ghaith\Desktop\final

 database\\*.wav','Select the sound file that you want to recognize it');

sp = fullfile(pathname, filename);

machindex = dir('C:\Users\Ghaith\Desktop\final database\\*.wav');

When the user enter No. 1 this means that he wants to recognize a not trained word, and a new window appears to ask him to choose a word to test it and see if there is matching with a trained word or no, figure 5.24 shows the window that appears to ask the user to choose a not trained word.



Figure 5.24: Selecting a not trained word window.

After selecting a word the system starts to extract features from the speech signal depending on the method that the user chose it in the beginning, and the result appears. If there is matching with another trained word, the name of the matched word appears in the command window of the MATLAB program as in figure 5.25, and a plotting of the speech signal appears in the main displaying interface window as in figure 5.26.



Figure 5.25: Decision window for a trained word.



Figure 5.26: Plotting of the matched trained word.

And if there is no matching with any other trained word a message box appears to inform the user that there is no matching with any other word.

These were all processes related with first type of recognition in this thesis (recognize a not trained word), the next explaining is related with the second type of recognition in this thesis (recognize a trained word with or without noise added to the word).

After clicking on the “Test the Network” button, and the window of asking the user to enter a No. for choosing the type of recognition as in figure 5.12, and the user enters No. 2, this means that the user wants to recognize a trained word with noise added with it, a new window appears to ask him to choose a word to test it and see if there is matching with a trained word or no as in figure 5.27.



Figure 5.27: Selecting a not trained word window.

After choosing a word, new buttons are activated on the main displaying interface window and these buttons are “SNR popup menu” button, and “Ok” button, and the plotting of the speech signal of the word appears also as in figure 5.28.



Figure 5.28: Displaying new buttons in main window.

After that the user selects a value from SNR popup menu, there is many values of SNR listed in the popup menu started with “No Noise” , “30 dB”, “25 dB”, “20 dB”, “15 dB”, “10 dB”, and “5 dB”. The MATLAB function for adding noise to the speech signal is:

 y = awgn(x,SNR,'measured');

Where *x* is the speech signal, *SNR* is the signal to noise ratio value in dB, and *measured* mean that the function measures the power of the speech signal before adding noise. And the values of SNR are determined and added to the program by this MATLAB code function:

functioncomput\_noise\_Callback(source,~)

str = get(source, 'String');

val = get(source,'Value');

% Signal to Noise Ratio in dB.

switchstr{val};

case'No Noise'

 SNR = 100;

case'30 dB'

 SNR = 30;

case'25 dB'

 SNR = 25;

case'20 dB'

 SNR = 20;

case'15 dB'

 SNR = 15;

case'10 dB'

 SNR = 10;

case'5 dB'

 SNR = 5;

end

end

After selecting SNR value the noise will be added in real time and the noise that added to the speech signal will be seen in the main displaying interface as shown in figure 5.29.



Figure 5.29: Speech signal after adding noise.

Then the user must click on “Ok” button to complete the recognizing, and the same processes of the first type will be done if there is matching or no matching with other words, and “SNR” and “Ok” buttons will deactivated again to allow the user to choose again another word for recognizing.

**5.5.8. Quit button**

When the user click on this button the speech recognition program will be closed with all figures related with the program, and the MATLAB code below was used in this program for this purpose:

functionquit\_Callback(~,~)

closeall;

end