SPEECH RECOGNITION USING ARTIFICIAL NEURAL NETWORKS

MINI PROJECT REPORT
(February-June 2004)
Project Code- RV04CS6P21

Submitted in the partial fulfillment of requirements for VI Semester, B.E.,
Computer Science and Engineering,
Visveswariah Technological University, Belgaum
by

Anushruthi Rai (1RV01CS011)
Arjun Jain (1RV01CS013)
Nalini V (1RV01CS059)

under the guidance of

Dr. Vishwanath,
Mr. Nagesha,
Mrs. Rohini Patil,
and
Mr. Pujar

Department of Computer Science and Engineering
R.V. College of Engineering, Bangalore

R.V. COLLEGE OF ENGINEERING

Department of Computer Science and Engineering

Bangalore - 59

CERTIFICATE

This is to certify that mini project titled **Speech Recognition using Neural Networks** has been successfully completed by

- Arjun Jain (1RV01CS013)

in partial fulfillment of VI Semester B.E, (Computer Science and Engineering), during the period February-June 2004 as prescribed by the Visveswariah Technological University, Belgaum.

Signature of

Staff Incharge **Prof. B.I. Khodanpur, HOD** **Examiner 1** **Examiner 2**

Contents

List of Figures

Acknowledgment

I wish to place on record, my grateful thanks to **Prof. B.I.Kodhanpur**, head of department of computer science, RVCE for providing all of us encouragement and guidance throughout our work.
I also convey my sincere regards to **Mr Vishwanath** ,**Mr Nagesha**, **Mrs Rohini Patil** and **Mr Pujar**, in charge, Mini Project lab RVCE for providing invaluable suggestions and guidance at all stages of development of this package.
I would also like to place my special thanks to all the staff of computer science department.
I thank all my friends for their help and encouragement, which has helped a lot in the completion of this project.

Chapter 1

Abstract

It is a well-known fact that building speech recognition systems is one of the hottest areas of research .Speech recognition has been an important part of the civilization for many centuries. We depend on intelligent and recognizable sounds for common communications.

In this project, spoken commands are recognized by the system and executed. The project consists of two phases. The first part deals with recognizing the formant frequencies associated with the input voice and the second phase involves pattern classification using Artificial Neural Networks.

Chapter 2

Literature Survey/Back Ground

The Process of speech recognition can be divided into the process of **1-Feature Extraction** and **2-Pattern Classification**. The various techniques for feature extraction commonly user are **1-Fourier transformations**, **2-Discrete Fourier transformations** and **3-Linear Predictive Coding**. The most popular techniques for pattern analysis and classification for the features thus obtained are **1-Template matching**, **2-Dynamic Warping Method** and **3-Artificial Neural Network Approach**.

We now explain briefly the various techniques and the reason for us having chosen a particular technique.

2.1 Feature Extraction Methods

2.1.1 The Fourier Transform

Since frequency is one of the important pieces of information necessary to accurately recognize sound, it is necessary to have a transformation that allows one to break a signal into its frequency components. The most common way to do this is the Fourier transform. The Fourier transform of a signal is the representation of the frequency and amplitude of that signal. The Fourier transform of a signal can obtained mathematically by the equation 2.1, where i is the imaginary number and the $|| \ ||$ represents magnitude.

$$X(f) = \left\| \int_{\infty}^{\infty} x(t) e^{-2\pi f t} dt \right\| \tag{2.1}$$

The Fourier transform of the signal in Figure 2.1 would then be given as figure 2.2

The Fourier Transform has its difficulties in accurately recognizing voice data, and the Short Term Fourier Transform has problems as well. However, each methodology is very good with a different aspect of the problem. The Fourier Transform resulted in excellent frequency resolution, while the Short Term Fourier Transform provided good time resolution. The obvious solution would be to take the best of both worlds.

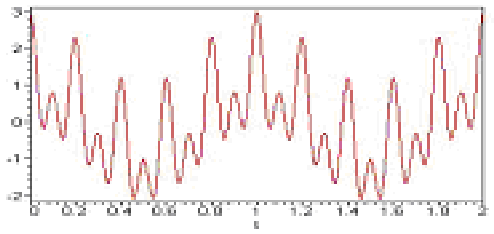


Figure 2.1: cos(2pt) + cos(10pt) + cos(20pt)

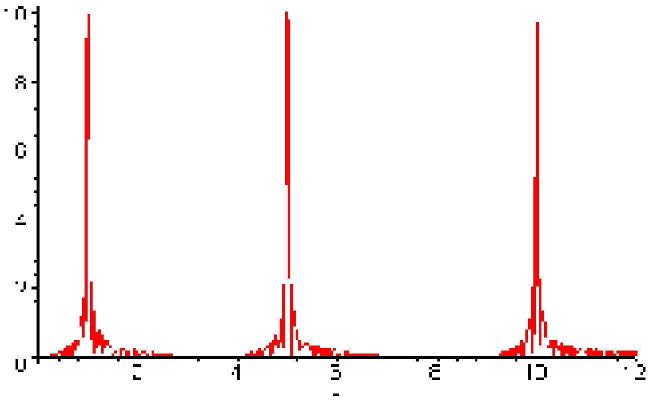


Figure 2.2: Fourier Transform, X(f), of x(t)

In most practical signals, low frequencies are stationary over the length of the signal. High frequencies however, tend to come and go over short intervals of time. Therefore, low frequencies should be analyzed over a longer time interval, and higher frequencies should be checked over a short interval. The result is a multi-resolution analysis, which is essentially the wavelet solution.

2.1.2 Discrete Fourier Transform

Take the signal in figure 2.3 as an example. This signal is discrete, so the Discrete Fourier Transform, shown in equation 2.2, is used to produce figure 2.4. The Discrete Fourier Transform is symmetric, so the first half of the data is really all that is interesting and so that is all that is shown. Ignore the x-axis and observe the location of the spikes. There are two spikes in the low frequency range, and one spike in the high frequency range. As seen in the DFT thus obtained, the high frequencies are attenuated. Thus, the method is rejected.

$$X(k) = \sum_{n=0}^{N-1} x[n](t)e^{\frac{-2\pi ikn}{N}}dt \tag{2.2}$$

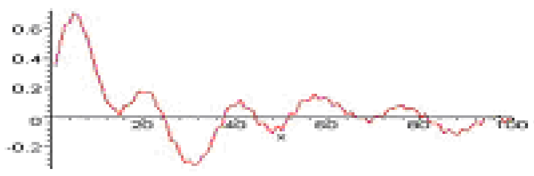


Figure 2.3: Discrete Signal

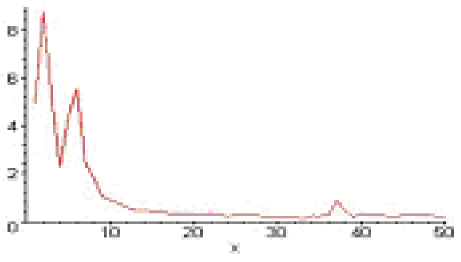


Figure 2.4: Discrete Fourier Transform, X(f), of x(t)

2.1.3 Linear Predictive Coding (The method implemented)

LPC is a modification of DFT. LPC starts with the assumption that the speech signal is produced by a buzzer at the end of a tube. The glottis (the space between the vocal cords) produces the buzz, which is characterized by its intensity (loudness) and frequency (pitch). The vocal tract (the throat and mouth) forms the tube, which is characterized by its resonances , which are called formants.

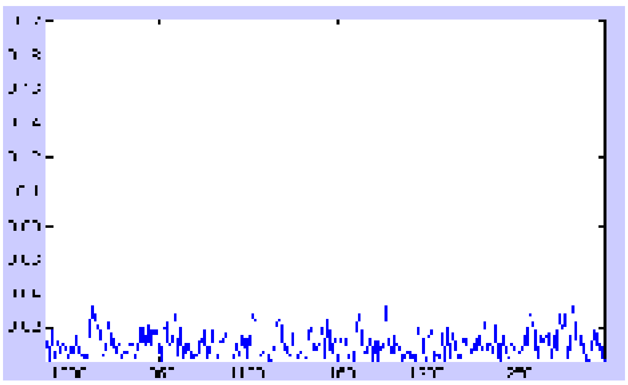


Figure 2.5: LPC of letter *a*

2.2 Pattern Classification Methods

Once we have extracted the features of the speech signal we go for pattern classification methods.

2.2.1 Template Matching

This is one of the simplest methods to measure similarity. Let t_i be the i^{th} lpc value of the test sample. And o_i be the i^{th} lpc value of the training sample. The test sample is compared with each of the training sets and the one with the best match is the one with the least Euclidean distance. Euclidean distance is given by:

$$E = \sqrt{\sum (t_j - o_j)^2} \tag{2.3}$$

t_i is the i^{th} lpc value of the training sample. o_i is the i^{th} lpc value of the test sample.

2.2.2 Dynamic Warping Method

This is the oldest method that has been used to identify speech. In this method speech is divided into frames of 30ms at every 15 ms intervals (allowing overlap). The lpc features of each frame are extracted.

A frame of the test sample is compared with the corresponding frame in the training sample by applying Euclidean formula. x_i- i^{th} lpc value of the x^{th} frame of the test sample. y_i- i^{th} lpc value of the y^{th} frame of the training sample. The dynamic equation is given by:

$$c(x,y) = min(c(x-1,y), c(x,y-1), c(x-1,y-1)) + ed(x,y) \tag{2.4}$$

Where $c(x,y)$ measures the dissimilarity between the test sample (up to frame x) and training sample (Upton frame y). The test sample is compared with all the trained samples, and one with the least $c(x,y)$ gives the best match.

2.2.3 Neural Network Approach(The method implemented)

A neural network is composed of a number of interconnected units (artificial neurons). Each unit has an input/output(I/O) characteristics and implements a local computation or function. The output of any unit is determined by the I/O characteristics, its interconnection to other units and (possibly) the external inputs.

Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANN as well. Learning typically occurs by example through training, or exposure to a truth-ed set of input/output data where the training algorithm iteratively adjusts the connection weights (synapses). These connection weights store the knowledge necessary to solve specific problems.

Chapter 3

Requirements Document

3.1 Purpose

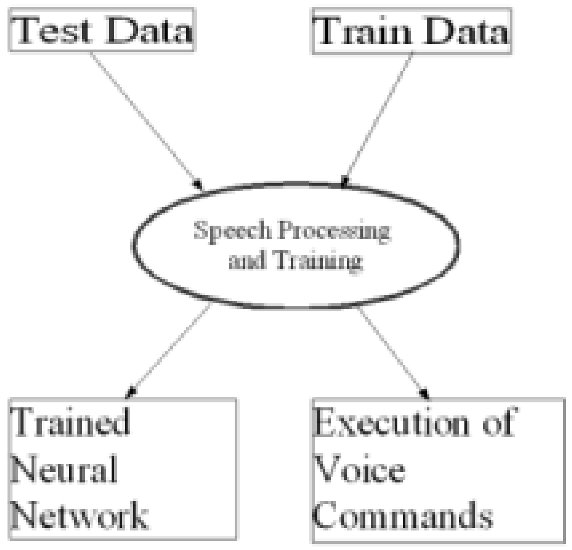
Speech recognition systems are widely used in security systems. Speech analog signal consumes a lot of space. It can be converted to text, transferred along any communication channel, reconverted back to speech on the receiver side. This saves considerable space. Speech recognition systems are good tools for those who are not fast at typing or who do not know to type. In this project we demonstrate speech recognition through voice commands.

3.2 Glossary

Linear predictive coding: LPC analyzes the speech signal by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency
Artificial Neural Networks: Also referred to as connectionist architectures, parallel distributed processing and neuromorphic systems, an artificial neural network (ANN) is an information processing paradigm inspired by the way the densely interconnected, parallel structure of the mammalian brain process information. Artificial neural networks are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning.

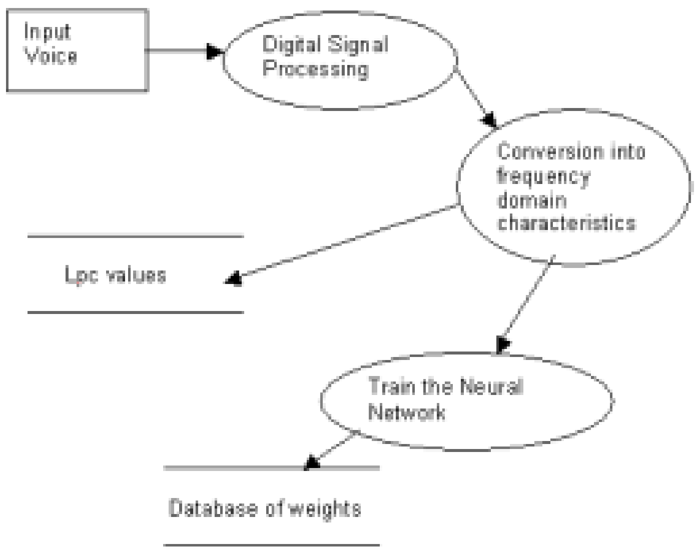
3.3 System Model: Data Flow Model

3.3.1 0-Level DFD

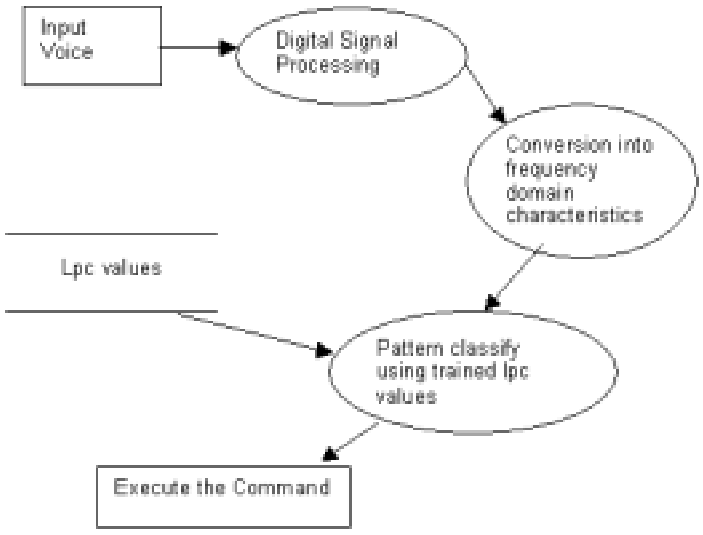


0-Level DFD

3.3.2 1-Level DFD


Input Voice
Digital Signal Processing
Conversion into frequency domain characteristics
Lpc values
Train the Neural Network
Database of weights

Training Process


Input Voice
Digital Signal Processing
Conversion into frequency domain characteristics
Lpc values
Pattern classify using trained lpc values
Execute the Command

Training with new patterns

3.4 Functional Requirements

The voice command is accepted through the microphone The command is reflected on the screen and executed.

3.5 Non functional requirements

The response time should be less than 10 seconds.

3.6 Hardware Requirements

High resolution sound card, high quality microphone and preferably a fast CPU. Also a system able to run the bellow mentioned softwares.

3.7 Software Requirements

Linux(Red Hat 9.0), Octave for mathematic/electronic operations, Gcc compilers, editors(gedit,kwrite) and the Troltech's Qt library for the interface design.

3.8 System Evolution

The system is strictly user dependent. To make it more accurate, the user has to train more.

3.9 Requirement Specification

The recognition system should consist of 2 parts.
Training session: Where the user trains the network with the desired commands.
Testing Session: The user says a command and the command is executed. ïż£

Chapter 4

Methodology(Design Document)

4.1 Architecture

It consists of a microphone, digital signal processor, a wave analyzer , a pattern classifier and display.

4.2 Abstract Specification

The voice command is accepted through the microphone The command is reflected on the screen and executed.

4.3 Data Structure Design

The following floating point arrays build up the various components of the Artificial Neural Network:

1. double **input- Then input layer of the network.
2. double *hidden- The hidden layer.
3. double **output- The output layer of the neural network.
4. double **target- The expected target output.
5. double *bias- The bias to each neuron.
6. double **weight_i_h- The weight matrix from the input to the hidden layer in the network.
7. double **weight_h_o- The weight matrix from the hidden to the output layer in the network.

8. double *errorsignal_hidden- The error propagated from the hidden layer.

9. double *errorsignal_output- The error propagated from the output layer.

4.4 Algorithm Design

4.4.1 Phase I: Speech recording and extraction of features

In This phase, the voice of the user is recorded. Then the octave library function lpc() is used for the extraction of the features in the speech. These lpc values are written to a text file.

```
record();
lpc();
FILE *fp=lpc.txt
```

4.4.2 Phase II: Pattern Classification

In this phase, the extracted lpc values are inputed to the neural network of classification and the neural network return the probabilities of possible patterns. The various functions used for the above mentioned purpose are:

1. `void initialize_net ()`- This is the method with initializes and allocates runtime heap memory for the various data structures.

2. `int compare_output_to_target ()`- This module is responsible for the comparison of the output of the network to the expected output in the learning phase. This function is called by the learn function.

3. `void load_data ()`- This function loads the data from the text files into the various data structures like the weight matrices.

4. `void save_data (char *argres)`- This function saves the data into the text files.

5. `void forward_pass (int pattern)`- The forward pass is the function which is responsible for the propagation of the input from the input to the hidden and the hidden to the output layers.

6. `void backward_pass (int pattern)`- This function propagates the output pattern to the hidden layer and the hidden layer to the input layer. It is used to propagate the error and readjusts the weight matrices.

7. `void compute_output_pattern ()`- This is the function which finally computes the output pattern. This function generates the probability of output for each neuron(possible output) of the output layer.

8. `float bedlam (long *idum)`- This is a function implemented to generate the initial random weight matrices. The inbuilt random function (srand())is not used because for optimum learning there are rules for constructing the initial weight matrices.

9. `void learn ()`- This module does the complete learning. It propagates the input using (`void forward_pass (int pattern)`), compares it with the desired output by calling (`int compare_output_to_target ()`) and finally readjusts the weight matrices by propagating the error back (`void backward_pass (int pattern)`).

10. `void test ()`- This function computes the probability of the test input pattern to the possible outputs.

11. `void print_data ()`- This is a function which can be used to display the final probabilities of the various outputs to the either console or to a file. It calls void `print_data_to_screen ()` or `void print_data_to_file ()`.

12. `void change_learning_rate ()`- This is a simple function which is used to change the learning rate of the network. The initial default learning rate alpha=0.05.

4.4.2.1 Structure of a neuron.

Our "artificial" neuron will have inputs (all N of them) and one output:

4.4.2.2 A Neural Network

A single neuron by itself is not a very useful pattern recognition tool. The real power of neural networks comes when we combine neurons into the multilayer structures, called neural networks. The following figure represents a simple neural net:

As you can see, the neuron has:
Set of nodes that connect it to inputs, output, or other neurons, also called synapses.
A Linear Combiner, which is a function that takes all inputs and produces a single value. A simple way of doing it is by adding together the Input multiplied by the Synaptic Weight:

```
for(int i = 0; i < NumOfInputs; i++)
    Sum += Input[i] * Weight[i];
```

An Activation Function. It will take ANY input from minus infinity to plus infinity and squeeze it into the -1 to 1 or into 0 to 1 interval. Finally, we have a threshold. It defines the INTERNAL ACTIVITY of a neuron should be when there is no input. In general, for the neuron to fire ,the sum should be greater than threshold. For simplicity, we will

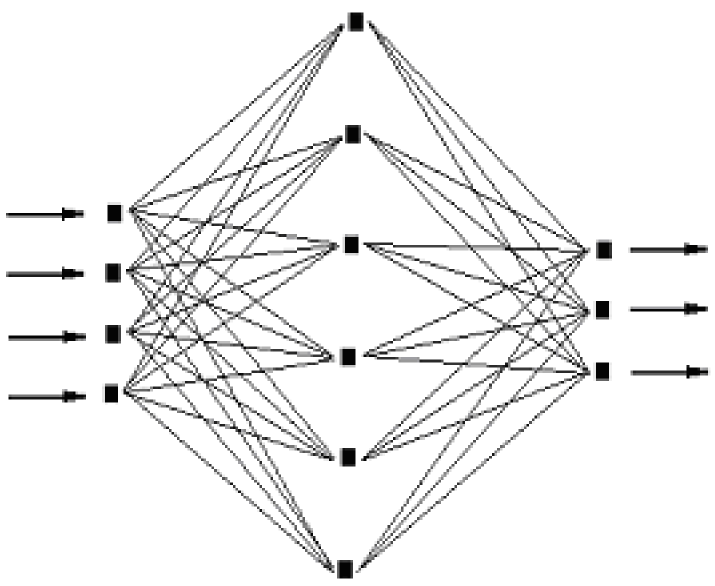


Figure 4.1: A 3-Layer Neural Network

replace the threshold with an EXTRA input, with weight that can change during the learning process and the input fixed and always equal (-1).

The first layer is known as the input layer, the middle layer is known as hidden layer and the last layer is the output layer.

4.4.2.3 The Back propagation Algorithm

The primary objective of this session is to explain how to use the back propagation training functions in the to train feed forward neural networks to solve speaker dependent speech recognition problems. There are generally four steps in the training process:

1. Assemble the training data
2. Create the network object
3. Train the network
4. Simulate the network response to new inputs

4.4.2.4 Feed forward Dynamics

When a BackProp network is cycled, the activations of the input units are propagated forward to the output layer through the connecting weights.

$$net_j = \sum w_j a_i \tag{4.1}$$

where a_i is the input activation from unit i and w_{ji} is the weight connecting unit i to unit j. However, instead of calculating a binary output, the net input is added to the unit's bias and the resulting value is passed through a sigmoid function:

$$F(net_j) = \frac{1}{1 + e^{-net_j + j}} \tag{4.2}$$

The sigmoid function is sometimes called a "squashing" function because it maps its inputs onto a fixed range.

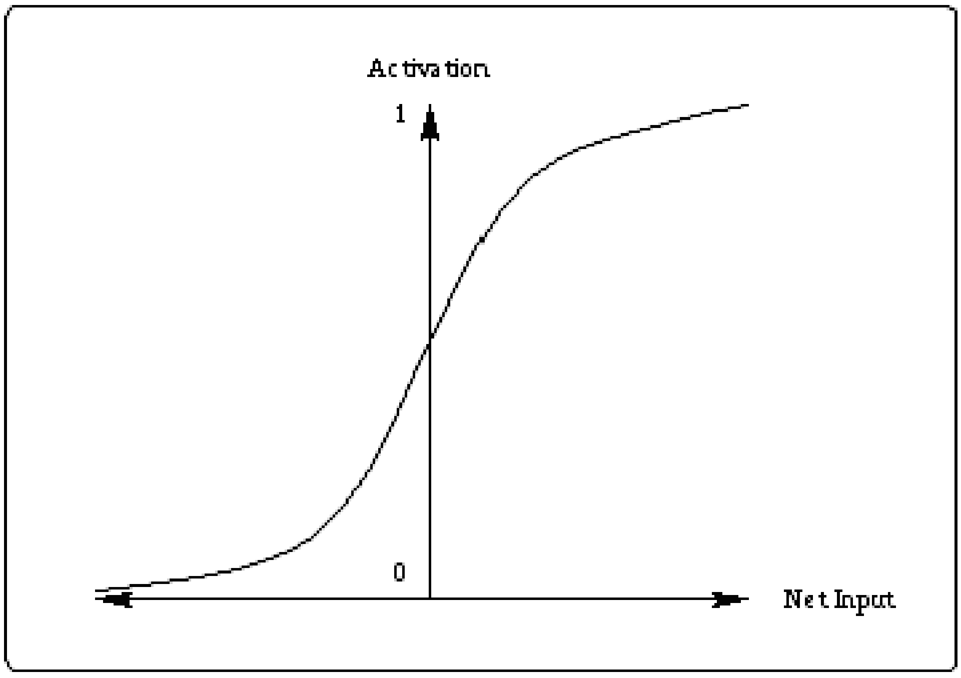


Figure 4.2: Sigmoid Activation Function

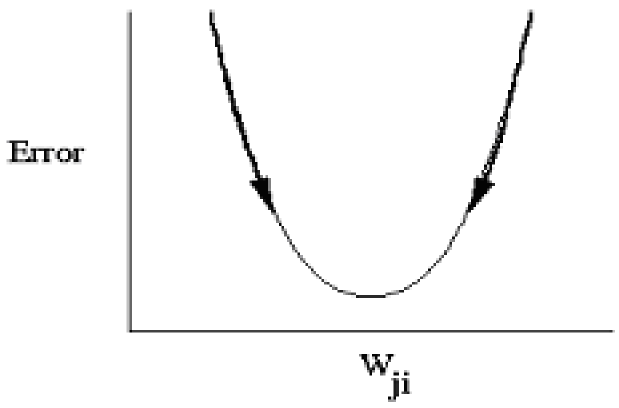


Figure 4.3: Gradient Descent

4.4.2.5 Training the neural network

The lpc values of each of the training sets is fed as input to the neural network. The target o/p is made 1 at the corresponding neuron the training set alphabet corresponds to. The are 100 inputs and x o/p s (x-corresponds to the no. of voice commands). The training is iterated and the weights are adjusted for each training sample.

Let e represent the input layer and a represent the output layer. Let e be the set of inputs. e represents the first 100 lpc values (formant frequencies) and b be the set of outputs. There are x outputs. ne represents the no. of neurons in the input layer and na represents the no. of neurons in the o/p layer. $w1$ represents the weights between the input and hidden layer, $w2$ represents the weight between the hidden layer and the output layer.

Initializing random weights to the matrices

```
for(i=1;i<=nhid;i++)
for(j=1;j<=ne;j++)
   w1[i][j]=rand()small weights;



for(i=1;i<=na;i++)
for(j=1;j<=nhid;j++)
   w2[i][j]=rand()small weights;
```

The training step

The lpc values of each of the training sets is fed as input to the neural network. The target o/p is made 1 at the corresponding neuron the training set voice command corresponds to. There are 100 inputs. At least 4 training samples for each command is required to get good results. The training is iterated and the weights are adjusted for each training sample.

```
for(i=0;i<=nhid;i++)
   {
      r=0;
      for(j=1;j<=ne;j++)
          r+=w1[i][j]*e[j];
      h[i]=1.0/(1+exp(-r));
   }
```

The output at each hidden layer is given by:
$h_j = f(net_j)$, which can be obtained by:

```
for(i=1;i<=na;i++)
```

```
    {
       r=0;
       for(j=1;j<=nhid;j++)
       r+=w2[i][j]*h[j];
       b[i]=1.0/(1+exp(-r));
    }
```

The output at the hidden layer acts as the input to the next layer . $b_j = f(net_j)$, where net_j represents the hidden layer.

Backpropagate the error to adjust w2.

```
 for(i=1;i<=na;i++)
    {
       err=a[i]-b[i];
       error[i]=err;
       tt=alpha*err*b[I]*(1-b[I]);
       for(j=1;j<=nhid;j++)
          w2[i][j]=w2[i][j]+tt*h[j];
    }
```

E is the Euclidean distance or the error. t_i is the target o/p and o_j is the obtained o/p.

$$E = \sum_j (t_j - o_j)^2 \tag{4.3}$$

$$w_{ji} = \alpha * d_j * i_j \tag{4.4}$$

Where α is the learning rate.
The Back propagation algorithm developed in this chapter only requires that the weight changes be proportional to the derivative of the error. The larger the learning rate alpha the larger the weight changes on each epoch, and the quicker the network learns. However, the size of the learning rate can also influence whether the network achieves a stable solution. If the learning rate gets too large, then the weight changes no longer approximate a gradient descent procedure. (True gradient descent requires infinitesimal steps). Oscillation of the weights is often the result.

Chapter 5

Implimentaion

```
//Backpropagation, binary sigmoid function network

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <float.h>
#include<string.h>
char person[30];
char infile[30];
char outfile[30];
double **input,
  *hidden,
  **output,
  **target,
  *bias, **weight_i_h, **weight_h_o, *errorsignal_hidden, *errorsignal_output;
int aaa = 4;
int trialn;
int input_array_size = 100,
  hidden_array_size = 100,
  output_array_size = 10,
  max_patterns,
  bias_array_size = 110,
  gaset = (unsigned long) &aaa,
  number_of_input_patterns = 0, pattern, file_loaded = 0, ytemp = 0, ztemp =
  0;
```

```
double learning_rate = 0.7, max_error_tollerance = 0.1;
char filename[128];
#define IA   16807
#define IM   2147483647
#define AM   (1.0 / IM)
#define IQ   127773
#define IR   2836
#define NTAB 32
#define NDIV (1+(IM-1) / NTAB)
#define EPS  1.2e-7
#define RNMX (1.0 - EPS)
int compare_output_to_target ();
void load_data ();
void save_data (char *argres);
void forward_pass (int pattern);
void backward_pass (int pattern);
void custom ();
void compute_output_pattern ();
void get_file_name ();
float bedlam (long *idum);
void learn ();
void make ();
void test ();
void print_data ();
void print_data_to_screen ();
void print_data_to_file ();
void output_to_screen ();
int getnumber ();
void change_learning_rate ();
void initialize_net ();
void clear_memory ();

main (int argc, char **argv)
{
  int x, y, z;

  char choice;

  cout << "\nEnter your name\n";
  cin >> person;
  strcpy (infile, person);
  strcpy (outfile, person);
  strcat (infile, "in");
  strcat (outfile, "out");


  ifstream in (person);
  if (!in)
    {
      cout << "new user!!!!\n";
      ofstream out, out1;
      out.open (person);

      out1.open (infile);
      out1 << (int) 0 << "                      " << endl;

      for (x = 0; x < bias_array_size; x++)
out << (1.0 - (2.0 * bedlam ((long *) (gaset)))) << ' ';
      out << endl << endl;
      for (x = 0; x < input_array_size; x++)
{
  for (y = 0; y < hidden_array_size; y++)
    out << (1.0 - (2.0 * bedlam ((long *) (gaset)))) << ' ';
}
      out << endl << endl;
      for (x = 0; x < hidden_array_size; x++)
{
  for (y = 0; y < output_array_size; y++)
    out << (1.0 - (2.0 * bedlam ((long *) (gaset)))) << ' ';
}
      out.close ();

    }
  in.close ();


  if (!strcmp (argv[1], "train"))
    choice = '1';
  else
    choice = '2';
  switch (choice)
    {
    case ’1’:


      make ();

      load_data ();
      learn ();

      clear_memory ();
      break;
    case ’2’:
      load_data ();
      compute_output_pattern ();
      clear_memory ();
      break;

    case ’3’:
      return 0;

    };
}

void
initialize_net ()
{
  int x;
  input = new double *[number_of_input_patterns];
  if (!input)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  for (x = 0; x < number_of_input_patterns; x++)
    {
      input[x] = new double[input_array_size];
      if (!input[x])
```

```
{
  cout << endl << "memory problem!";
  exit (1);
}
    }
  hidden = new double[hidden_array_size];
  if (!hidden)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  output = new double *[number_of_input_patterns];
  if (!output)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  for (x = 0; x < number_of_input_patterns; x++)
    {
      output[x] = new double[output_array_size];
      if (!output[x])
{
  cout << endl << "memory problem!";
  exit (1);
}
    }
  target = new double *[number_of_input_patterns];
  if (!target)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  for (x = 0; x < number_of_input_patterns; x++)
    {
      target[x] = new double[output_array_size];
      if (!target[x])
{
  cout << endl << "memory problem!";
  exit (1);
}
```

```
    }
  bias = new double[bias_array_size];
  if (!bias)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  weight_i_h = new double *[input_array_size];
  if (!weight_i_h)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  for (x = 0; x < input_array_size; x++)
    {
      weight_i_h[x] = new double[hidden_array_size];
      if (!weight_i_h[x])
{
  cout << endl << "memory problem!";
  exit (1);
}
    }
  weight_h_o = new double *[hidden_array_size];
  if (!weight_h_o)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  for (x = 0; x < hidden_array_size; x++)
    {
      weight_h_o[x] = new double[output_array_size];
      if (!weight_h_o[x])
{
  cout << endl << "memory problem!";
  exit (1);
}
    }
  errorsignal_hidden = new double[hidden_array_size];
  if (!errorsignal_hidden)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  errorsignal_output = new double[output_array_size];
  if (!errorsignal_output)
    {
      cout << endl << "memory problem!";
      exit (1);
    }
  return;
}

void
learn ()
{
  int x, y;
  double inpx;

  cout << "\n learning....";

  while (1)
    {
      for (y = 0; y < number_of_input_patterns; y++)
{
  forward_pass (y);
  backward_pass (y);
}
      if (compare_output_to_target ())
{
  cout << endl << "learning successful" << endl;
  ofstream out;
  out.open (person);
  if (!out)

    cout << "ha mem problem";



  for (x = 0; x < bias_array_size; x++)
    {
      out << bias[x] << ' ';
    }
  out << endl << endl;
  for (x = 0; x < input_array_size; x++)

    for (y = 0; y < hidden_array_size; y++)
      {
out << weight_i_h[x][y] << ' ';
      }

  out << endl << endl;
  for (x = 0; x < hidden_array_size; x++)

    for (y = 0; y < output_array_size; y++)
      {

out << weight_h_o[x][y] << ' ';
      }
  out << endl << endl;

  return;
}

    }
  cout << endl << "learning not successful yet" << endl;
  return;
}

void
load_data ()
{
  int x, y;
  ifstream in, in1;



  in1.open (infile);
  in1 >> number_of_input_patterns;

  bias_array_size = hidden_array_size + output_array_size;
  initialize_net ();
  in.open (person);
  for (x = 0; x < bias_array_size; x++)
    in >> bias[x];
  for (x = 0; x < input_array_size; x++)
    {
      for (y = 0; y < hidden_array_size; y++)
in >> weight_i_h[x][y];
    }
  for (x = 0; x < hidden_array_size; x++)
    {
      for (y = 0; y < output_array_size; y++)
in >> weight_h_o[x][y];
    }



  in.close ();



  for (x = 0; x < number_of_input_patterns; x++)
    {
      for (y = 0; y < input_array_size; y++)
in1 >> input[x][y];
    }
  in1.close ();

  in.open (outfile);

  for (x = 0; x < number_of_input_patterns; x++)
    {
      for (y = 0; y < output_array_size; y++)
in >> target[x][y];
    }
  in.close ();

  return;
}
```

```
void
forward_pass (int pattern)
{
  register double temp = 0;
  register int x, y;

// INPUT -> HIDDEN
  for (y = 0; y < hidden_array_size; y++)
    {
      for (x = 0; x < input_array_size; x++)
{
  temp += (input[pattern][x] * weight_i_h[x][y]);
}
      hidden[y] = (1.0 / (1.0 + exp (-1.0 * (temp + bias[y]))));
      temp = 0;
    }

// HIDDEN -> OUTPUT
  for (y = 0; y < output_array_size; y++)
    {
      for (x = 0; x < hidden_array_size; x++)
{
  temp += (hidden[x] * weight_h_o[x][y]);
}
      output[pattern][y] =
(1.0 / (1.0 + exp (-1.0 * (temp + bias[y + hidden_array_size]))));
      temp = 0;
    }
  return;
}




void
backward_pass (int pattern)
{
  register int x, y;
  register double temp = 0;

// COMPUTE ERRORSIGNAL FOR OUTPUT UNITS
```

```
  for (x = 0; x < output_array_size; x++)
    {
      errorsignal_output[x] = (target[pattern][x] - output[pattern][x]);
    }

// COMPUTE ERRORSIGNAL FOR HIDDEN UNITS
  for (x = 0; x < hidden_array_size; x++)
    {
      for (y = 0; y < output_array_size; y++)
{
  temp += (errorsignal_output[y] * weight_h_o[x][y]);
}
      errorsignal_hidden[x] = hidden[x] * (1 - hidden[x]) * temp;
      temp = 0.0;
    }

// ADJUST WEIGHTS OF CONNECTIONS FROM HIDDEN TO OUTPUT UNITS
  double length = 0.0;
  for (x = 0; x < hidden_array_size; x++)
    {
      length += hidden[x] * hidden[x];
    }
  if (length <= 0.1)
    length = 0.1;
  for (x = 0; x < hidden_array_size; x++)
    {
      for (y = 0; y < output_array_size; y++)
{
  weight_h_o[x][y] += (learning_rate * errorsignal_output[y] *
       hidden[x] / length);
}
    }

// ADJUST BIASES OF HIDDEN UNITS
  for (x = hidden_array_size; x < bias_array_size; x++)
    {
      bias[x] +=
(learning_rate * errorsignal_output[x - hidden_array_size] / length);
    }
```

```
// ADJUST WEIGHTS OF CONNECTIONS FROM INPUT TO HIDDEN UNITS
  length = 0.0;
  for (x = 0; x < input_array_size; x++)
    {
      length += input[pattern][x] * input[pattern][x];
    }
  if (length <= 0.1)
    length = 0.1;
  for (x = 0; x < input_array_size; x++)
    {
      for (y = 0; y < hidden_array_size; y++)
{
  weight_i_h[x][y] += (learning_rate * errorsignal_hidden[y] *
       input[pattern][x] / length);
}
    }

// ADJUST BIASES FOR OUTPUT UNITS
  for (x = 0; x < hidden_array_size; x++)
    {
      bias[x] += (learning_rate * errorsignal_hidden[x] / length);
    }
  return;
}

int
compare_output_to_target ()
{
  register int y, z;
  register double temp, error = 0.0;
  temp = target[ytemp][ztemp] - output[ytemp][ztemp];
  if (temp < 0)
    error -= temp;
  else
    error += temp;
  if (error > max_error_tollerance)
    return 0;
  error = 0.0;
  for (y = 0; y < number_of_input_patterns; y++)
    {
      for (z = 0; z < output_array_size; z++)
{
  temp = target[y][z] - output[y][z];
  if (temp < 0)
    error -= temp;
  else
    error += temp;
  if (error > max_error_tollerance)
    {
      ytemp = y;
      ztemp = z;
      return 0;
    }
  error = 0.0;
}
    }
  return 1;
}

void
save_data (char *argres)
{
  int x, y;
  ofstream out;
  out.open (argres);
  if (!out)
    {
      cout << endl << "failed to save file" << endl;
      return;
    }
  out << input_array_size << endl;
  out << hidden_array_size << endl;
  out << output_array_size << endl;
  out << learning_rate << endl;
  out << number_of_input_patterns << endl << endl;
  for (x = 0; x < bias_array_size; x++)
    out << bias[x] << ' ';
  out << endl << endl;
  for (x = 0; x < input_array_size; x++)
    {
      for (y = 0; y < hidden_array_size; y++)
out << weight_i_h[x][y] << ' ';
    }
  out << endl << endl;
  for (x = 0; x < hidden_array_size; x++)
    {
      for (y = 0; y < output_array_size; y++)
out << weight_h_o[x][y] << ' ';
    }
  out << endl << endl;
  for (x = 0; x < number_of_input_patterns; x++)
    {
      for (y = 0; y < input_array_size; y++)
out << input[x][y] << ' ';
      out << endl;
    }
  out << endl;
  for (x = 0; x < number_of_input_patterns; x++)
    {
      for (y = 0; y < output_array_size; y++)
out << target[x][y] << ' ';
      out << endl;
    }
  out.close ();
  cout << endl << "data saved" << endl;
  return;
}

void
make ()
{
  int x, y, z;
  double inpx;



  ofstream out;
  ifstream in;
  cout << "Enter the no.of patterns u wanna train : ";
  cin >> z;









  FILE *inf = fopen (infile, "r");


  fscanf (inf, "%d", &trialn);
  fclose (inf);


  inf = fopen (infile, "r+");
  fprintf (inf, "%d", trialn + z);
  fclose (inf);


  inf = fopen (infile, "a");

  FILE *ouf = fopen (outfile, "a");

  for (x = 0; x < z; x++)
    {
      char ent;
      cout <<
"1.ls \n2.date \n3.pwd \n4.mail \n5.pinky \n6.dmesg \n7.who am i \n8.cal \n9.ps";
      cout << "\nenter choice ";
      int ch;
      cin >> ch;
      cin.get (ent);
      int i;
      for (i = 0; i < output_array_size; i++)
{
  if (ch - 1 == i)
    fprintf (ouf, "%d ", 1);
  else
```

```
    fprintf (ouf, "%d ", 0);
}



      fprintf (ouf, "\n\n");
      cout << "press enter and speak\n";
      cin.get (ent);
      system ("octave nn.m>dummy.txt");
      in.open ("data.txt");
      for (y = 0; y < input_array_size; y++)
{
  in >> inpx;

  fprintf (inf, "%f ", inpx);

}
      fprintf (inf, "\n\n");
      in.close ();

    }

  fclose (inf);
  fclose (ouf);







  return;
}

float
bedlam (long *idum)
{
  int xj;
  long xk;
  static long iy = 0;
  static long iv[NTAB];
  float temp;
  if (*idum <= 0 || !iy)
    {
      if (-(*idum) < 1)
{
  *idum = 1 + *idum;
}
      else
{
  *idum = -(*idum);
}
      for (xj = NTAB + 7; xj >= 0; xj--)
{
  xk = (*idum) / IQ;
  *idum = IA * (*idum - xk * IQ) - IR * xk;
  if (*idum < 0)
    {
      *idum += IM;
    }
  if (xj < NTAB)
    {
      iv[xj] = *idum;
    }
}
      iy = iv[0];
    }

  xk = (*idum) / IQ;
  *idum = IA * (*idum - xk * IQ) - IR * xk;
  if (*idum < 0)
    {
      *idum += IM;
    }
  xj = iy / NDIV;
  iy = iv[xj];
  iv[xj] = *idum;

  if ((temp = AM * iy) > RNMX)
    {
      return (RNMX);
    }
  else
    {
      return (temp);
    }
}

void
test ()
{
  pattern = 0;
  while (pattern == 0)
    {
      cout << endl << endl << "There are " << number_of_input_patterns
<< " input patterns in the file," << endl <<
"enter a number within this range: ";
      pattern = getnumber ();
    }
  pattern--;
  forward_pass (pattern);
  output_to_screen ();
  return;
}

void
output_to_screen ()
{
  int x;
  // cout << endl << "Output pattern:" << endl;
  for (x = 0; x < output_array_size; x++)
    {
      cout << endl << (x + 1) << ": " << output[pattern][x] << "    binary: ";
      if (output[pattern][x] >= 0.9)
cout << "1";
      else if (output[pattern][x] <= 0.1)
cout << "0";
      else
cout << "intermediate value";
    }
  cout << endl;
  return;
}

int
getnumber ()
{
  int a, b = 0;
  char c, d[5];
  while (b < 4)
    {
      do
{
  c = getchar ();
}
      while (c != '1' && c != '2' && c != '3' && c != '4' && c != '5'
     && c != '6' && c != '7' && c != '8' && c != '9' && c != '0'
     && toascii (c) != 13);
      if (toascii (c) == 13)
break;
      if (toascii (c) == 27)
return 0;
      d[b] = c;
      cout << c;
      b++;
    }
  d[b] = '\0';
  a = atoi (d);
  if (a < 0 || a > number_of_input_patterns)
    a = 0;
  return a;
}

void
compute_output_pattern ()
{
  custom ();



}

void
custom ()
{

  char filename[128];
  register double temp = 0;
  register int x, y;
  double *custom_input = new double[input_array_size];
  if (!custom_input)
    {
      cout << endl << "memory problem!";
      return;
    }
  double *custom_output = new double[output_array_size];
  if (!custom_output)
    {
      delete[]custom_input;
      cout << endl << "memory problem!";
      return;
    }

  char ent;
  cin.get (ent);
  for (;;)
    {
      cout << "press enter and speak";
      cin.get (ent);
      cout << endl;

      system ("octave nn.m>dummy.txt");
      ifstream in ("data.txt");
      temp = 0;
      if (!in)
```

```
{
  cout << endl << "failed to load data file" << endl;
  return;
}
      for (x = 0; x < input_array_size; x++)
{
  in >> custom_input[x];
}
      for (y = 0; y < hidden_array_size; y++)
{
  for (x = 0; x < input_array_size; x++)
    {
      //cout<<"wt="<<weight_i_h[x][y]<<endl;
      temp += (custom_input[x] * weight_i_h[x][y]);
    }
  hidden[y] = (1.0 / (1.0 + exp (-1.0 * (temp + bias[y]))));
  temp = 0;
}
      for (y = 0; y < output_array_size; y++)
{
  for (x = 0; x < hidden_array_size; x++)
    {
      temp += (hidden[x] * weight_h_o[x][y]);
    }
  custom_output[y] =
    (1.0 / (1.0 + exp (-1.0 * (temp + bias[y + hidden_array_size]))));
  temp = 0;
}


      int lett;
      double max = 0;



      for (x = 0; x < output_array_size; x++)
{



  if (custom_output[x] > max)
    {
      max = custom_output[x];
      lett = x;
    }





}



      switch (lett)
{
case 0:
  cout << endl << "ls" << endl;
  system ("ls");
  break;
case 1:
  cout << endl << "date" << endl;
  system ("date");
  break;
case 2:
  cout << endl << "pwd" << endl;
  system ("pwd");
  break;
case 3:
  cout << endl << "mail" << endl;
  system ("mail");
  break;
case 4:
  cout << endl << "pinky" << endl;
  system ("pinky");
  break;
case 5:
  cout << endl << "dmesg" << endl;
  system ("dmesg");
  break;
case 6:
  cout << endl << "who am i" << endl;
  system ("who am i");
  break;
case 7:
  cout << endl << "ls" << endl;
  system ("ls");
  break;
case 8:
  cout << endl << "cal" << endl;
  system ("cal");
  break;
case 9:
  cout << endl << "ps" << endl;
  system ("ps");
  break;

}



    in.close ();
    cout << endl;
  }
  delete[]custom_input;
  delete[]custom_output;
  return;
}

void
clear_memory ()
{
  int x;
  for (x = 0; x < number_of_input_patterns; x++)
    {
      delete[]input[x];
    }
  delete[]input;
  delete[]hidden;
  for (x = 0; x < number_of_input_patterns; x++)
    {
      delete[]output[x];
    }
  delete[]output;
  for (x = 0; x < number_of_input_patterns; x++)
    {
      delete[]target[x];
    }
  delete[]target;
  delete[]bias;
  for (x = 0; x < input_array_size; x++)
    {
      delete[]weight_i_h[x];
    }
  delete[]weight_i_h;
  for (x = 0; x < hidden_array_size; x++)
    {
      delete[]weight_h_o[x];
    }
  delete[]weight_h_o;
  delete[]errorsignal_hidden;
  delete[]errorsignal_output;
  file_loaded = 0;
  return;
}
```

Chapter 6

Experiment Analysis And Testing

6.1 Feeding Test Data

The test sample is fed to the Neural Network. Using the trained weights the o/p at each neuron is calculated. The o/p of each neuron at the o/p layer is found. The voice command corresponding to the neuron that gives the maximum o/p is the match required.

The greater the no. of training samples ,greater is the accuracy. As the number of commands increases more training samples have to be given.Lesser the allowable error, the greater is the accuracy. But these factors limit speed.

6.2 Unit Testing

Each module was tested independently.

6.2.1 lpc()

Initially 2000 lpc values were taken.It gave good results but took more time to learn. Hence it was reduced to 100 which gave faster results without considerably sacrificing accuracy.

6.2.2 learn()

The error tolerance was initially set to 0.1. This did not give good results.The outcome was better with 0.05 tolerance.

The other modules were tested and they gave no problems.

6.3 Interface Testing

After all the modules (feature extraction, recognition, front-end) were compiled and tested individually using unit testing, they were integrated to make the final product. After that, each test performed in the unit testing phase (refer section Unit Testing) was then performed on the product as a whole.

Many errors were then rectified like:

- The sting class of C++ used of the back-end were all changed to QString to make it compatible with the Qt[1] libraries used.
- All functions were explicitly prototyped for the Qt libraries to be able to call them from the private section of the classes.

[1]Qt is the library by Troltech inc used for the GUI design for the front-end

Chapter 7

Conclusion and future Enhancement

The program is capable of talking voice commands and carrying out the specified operation by that command. The percentage accuracy is about 85%. Also it take a considerable time in processing the voice input.
In future, we propose to enhance the system for:

- Increase the accuracy by using more differentiating features like mel frequencies instead of lpc values.
- Decreasing the processing time by using less no of input features for each command but using improved input methods.
- Provide for filtering of noise to the input signal and further improvise it by using hardware filters and dsp processors.
- Improvise it to work for continuous speech.
- Walk towards speaker independency by decreasing the threshold amount of traning required by the network for successful pattern classification.

Chapter 8

References

1. Digital Processing of Speech Signals: L. R. Rabiner and R. W. Schafer Prentice-Hall (Signal Processing Series), 1978

2. The Government Standard Linear Predictive Coding : LPC-10

3. Thomas E. Tremain. Speech Technology Magazine, April 1982, p. 40-49

4. Simon Haykin: Neural Networks, A Comprehensive Foundation, Second Edition

5. B. Yegnanarayana : Artificial Neural Networks

6. Robert J. Schalkoff : Artificial Neural Networks

7. Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Müller: Neural Networks: Tricks of the Trade

8. Paul J. Werbos, Backpropagation: basics and new developments, The handbook of brain theory and neural networks, MIT Press, Cambridge, MA, 1998