

WORD RECOGNITION USING NEURAL NETWORK

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

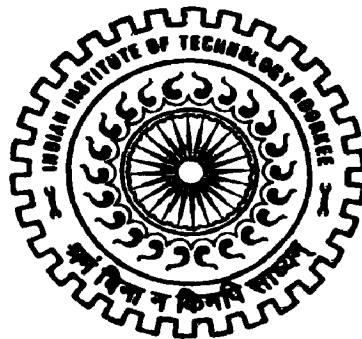
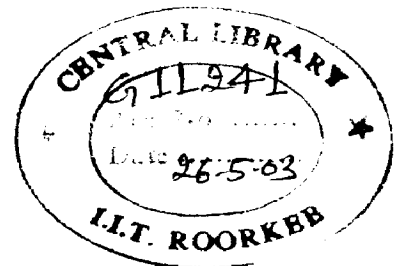
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

VAISHALI GOVINDRAO KALE



**ER & DCI
NOIDA**

**IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307**

FEBRUARY, 2003

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled **“WORD RECOGNITION USING NEURAL NETWORK”**, in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT, Roorkee – ER&DCI Campus, Noida**, is an authentic record of my own work carried out during the period from August, 2002 to February, 2003 under the guidance of **Mr. M. K. BHATTACHARYA** , Senior Project Manager , Electronics Research and Development Center of India, Noida.

The matter embodied in this dissertation has not been submitted by me for award of any other degree or diploma

Date: 26-02-03

Place: Noida



(Vaishali G. Kale)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 26-02-03

Place: Noida



(Mr. M. K. Bhattacharya)

Senior Project Manager
ER&DCI, Noida

ACKNOWLEDGEMENT

The successful completion of this project “ Word Recognition Using Neural Network”, is attributed to the great and indispensable help I have received from different people of my institute.

I take this opportunity to thank **Prof. Prem Vratt** ,Director, IIT-Roorkee and **Shree.R.K.Verma** ,Executive Director, ER&DCI, Noida, who were the back bones of this M.Tech (IT) course.

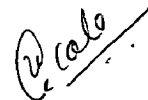
Special thanks to **Prof.A.K.Awasthi**, **Dean,PGS&R,IIT-Roorkee**, for his kind coordination and cooperation in running this course successfully.

I express my profound gratitude to **Prof.R.P.Agrawal** ,Course coordinator, M.Tech(IT) and **Mr.V.N.Shukla** Director, Special. Applications, ER&DCI, who contributed a lot to make this M.Tech course a grand success.

My sincere thanks to my project guide **Mr.M.K.Bhattacharya**, Senior Project Manager, ER&DCI, for his proper guidance, excellent spirit of coordination, his constructive criticisms and his help in formulation of this project.

I extend my thanks to **Mr.Munish Kumar** and **Dr.P.R.Gupta**, staff members, M.Tech(IT),ER&DCI for their proper guidance, help and support.

Above all, my heartfelt thanks to my family and friends,who offered moral support while suffering from my neglect for the past two years.



(Vaishali G. kale)

CONTENTS

CANDIDATE'S DECLARATION	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	1
1 INTRODUCTION	3
1.1 Background	3
1.2 Objective	4
1.3 Scope	4
1.4 Organization of Dissertation	5
2 LITERATURE SURVEY OF ARTIFICIAL NEURAL NETWORK	7
2.1 Artificial Neural Network	7
2.2 Historical Background	9
2.3 The Biological Model	9
2.4 Firing rules	12
2.5 Single and Multi-layer Perceptrons	14
3 BACKPROPAGATION NEURAL NETWORK	17
3.1 Introduction To Backpropagation Neural Network	17
3.2 Backpropagation Training Algorithm	19
3.2.1 Selection and Preparation of Training Data	20
3.2.2 Modification of the neuron connection weights	21
3.2.3 Repetition	24
3.2.4 Running	25
3.2.5 Hazards	25
3.3 Existing System	25
3.3.1 Template Matching	26

4 DESIGN AND IMPLEMENTATION	27
4.1 Design Flowchart	28
4.2 Implementation Strategy	29
4.3 Algorithm: Backpropagation	30
5 RESULTS AND DISCUSSION	33
6 CONCLUSION	41
REFERENCES	43
APPENDIX A : USER MANUAL	45
APPENDIX B : GLOSSARY	49

ABSTRACT

Word Recognition using Artificial Neural Networks, implemented the back-propagation neural network for developing word recognition software. The goal is to adapt the parameters of the network so that it performs well for patterns from outside the training set.

The word recognition software is made using Java2 and efficiently recognizes all alphanumeric characters which the user writes on the screen. The GUI for the software, developed using the Swing features provided in Java2, makes it highly ergonomic and purposeful.

The software can also be extended for developing a vehicle number-plate recognition system, Signature Verification system

INTRODUCTION

1.1 Background :

Visual pattern recognition such as reading characters or recognizing shapes is an easy task for human beings, but presents significant difficulty when programming an information processor to do the same thing. Artificial neural networks are a method of computation that try to achieve human-like performance in the field of image and character recognition. Artificial neural network models are composed of many non-linear computational elements operating in parallel and arranged in patterns mimicking biological neurons. These models are very good at interpreting vague, noisy and incomplete input. Artificial neural networks are trained from experience. In supervised learning, we try to adapt an artificial neural network so that its actual outputs come close to some target outputs for a training set which contains a predefined number of patterns. The goal is to adapt the parameters of the network so that it performs well for patterns from outside the training set. One of the main uses of supervised learning today lies in pattern recognition. Basic backpropagation is currently one of the most popular methods for performing the supervised learning task. A backpropagation network usually consists of three (or sometimes fewer) layers of neurons: the input layer, the hidden layer, and the output layer. The fundamental idea behind backpropagation is that the error is propagated backward to earlier layers so that a gradient search algorithm can be applied.

This thesis will discuss the basics behind the backpropagation neural networks. A practical implementation of backpropagation training algorithm to recognize handwritten characters and extended to recognize the word . The implementation is done using the Java programming language. Detailed design of the implementation is presented in chapter 5. [13]

1.2 Objective :

The project is to develop a system capable of recognizing handwritten characters or symbols, inputted by the means of a mouse. The system provides means for training the input characters first, then there is a classification option where the patterns or symbols that have already been trained should be fed, in order to recognize it. There are full options for the users, like

1. To load a default set of patterns which is already present in the system, which can either be trained or the default training file can be loaded. After which the recognition takes place.
2. To classify a line of text (to recognize individual words in a inputted line).
3. Options to either load a pattern file, trained or untrained & finally the option to create new patterns by the user to train & classify (added feature).

1.3 Scope :

Word recognition is the area in which Neural networks are providing solutions . Some of these solution are beyond simply academic curiosities, like neural network based product can be used to recognize hand written character through he scanner. This application can be used in credit card application form, where characters can be stored in the database after recognition.

Currently, the system is build to highlight characters below the certain percent probability of being right so that a user can manually fill in what the computer could not.

The word recognition system is developed for recognizing the handwritten character inputted by mean of mouse. The system is trained for the capital letters only but it can trained for lower case letter. The system can classify up to 13 character word. The system can efficiently recognize the patterns inputted by the user. Facility to load the created file and create a new pattern file is given. The user can train the network for newly created patterns in order to recognize it.

The program has been tested with two sets of training patterns – letters and numbers.

1.4 Organization of Dissertation :

Introductory chapter deals with the objective and scope of the dissertation. The chapter concludes with the organization of the project.

Chapter 2 deals with the review of the work done on the subject. It contains the overview of the Artificial Neural Network . It also shows the relationship with a biological neural network

Chapter 3 explains famous neural network topology called a multi-layer perceptron, to be trained with the Feed-forward Back-Propagation Network (BPN) algorithm. Finally it gives the existing solution to character recognition.

In chapter 4 design and implementation aspects for the model is explained.

In chapter 5 results and discussion are given.

Finally, the dissertation has been concluded with the chapter 6 which brings out the recognition of various alphanumeric patterns.

LITERATURE SURVEY OF ARTIFICIAL NEURAL NETWORK

2.1 Artificial Neural Network :

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest. Other advantages include:

1. **Adaptive learning:** An ability to learn how to do tasks based on the data given for training or initial experience.
2. **Self-Organization:** An ANN can create its own organization or representation of the information it receives during learning time.
3. **Real Time Operation:** ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. **Fault Tolerance via Redundant Information Coding:** Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage. Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable. On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault. Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.[14]

2.2 Historical Background :

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras. Many important advances have been boosted by the use of inexpensive computer emulation. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among Researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts. But the technology available at that time did not allow them to do too much.[14]

2.3 The Biological Model :

How the Human Brain Learns?

Much is still unknown about how the brain trains itself to process information, so theories abound. In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites. The neuron sends out spikes of electrical activity through a long, thin strand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing

the effectiveness of the synapses so that the influence of one neuron on another changes.[12]

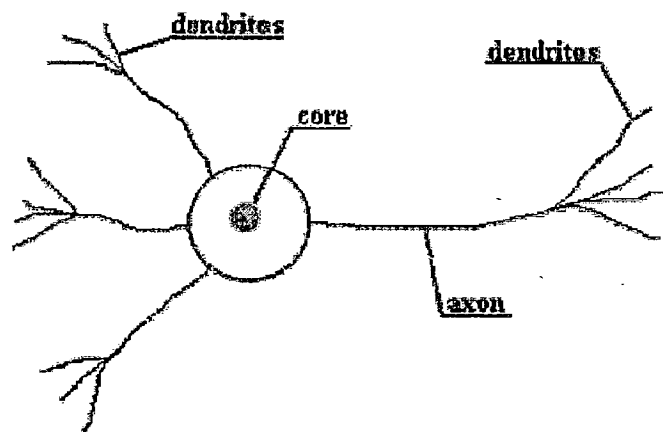


FIG 2.1 : Human Neuron

From Human Neurons to Artificial Neurons :

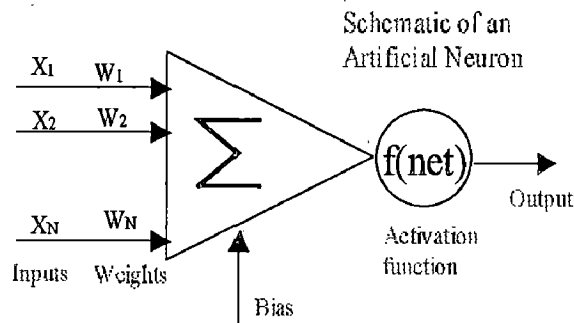


FIG 2.2 : The Neuron Model

Just as there is a basic biological neuron, there is basic artificial neuron. Each neuron has a certain number of inputs, each of which have a weight assigned to them. These are denoted by input signals x_i and set of real valued weights w_i in the figure. The weights simply are an indication of how 'important' the incoming signal for that input is. The net value of the neuron is then calculated - the net is simply the weighted sum, the sum of all the inputs multiplied by their specific weight. This is also denoted by the activation level, $\Sigma x_i w_i$. Each neuron has its own unique threshold value, and if the net is greater than the threshold, the neuron fires (or outputs a 1), otherwise it stays quiet (outputs a 0). The threshold function f is used to compute the value of the output. The output is then fed into all the neurons it is connected to.[13]

In calculating the output of the neuron, the activation function may be in the form of a threshold function, in which the output of the neuron is +1 if a threshold level is reached and 0 otherwise. Squashing functions limit the linear output between a maximum and minimum value. These linear functions, however, do not take advantage of multi-layer networks. Hyperbolic tangents and the sigmoid functions are similar to real neural responses; however, the hyperbolic tangent is unbounded and hard to implement in hardware. In this project, the Sigmoid function is used because of its ability to produce continuous non-linear functions, which can be implemented in hardware in future research areas.[12]

Sigmoid function is an exponential function which has as a most important characteristic the fact that, even if x assumes values next to the infinitely big or little, $f(x)$ will assume a value between 0 and 1. The learning algorithm will adjust the weights of the connections between units so that the function translates values of x to a binary value, typically: $f(x) > 0.9 : f(x) = 1$, $f(x) < 0.1 : f(x) = 0$.

Figure shows some commonly used activation functions.

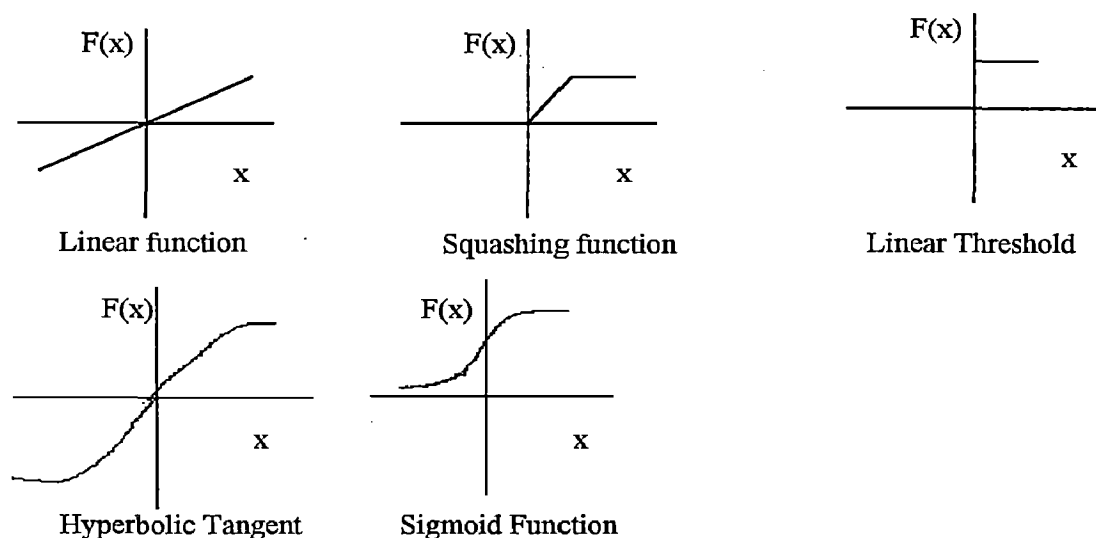


FIG 2.3 : various activation functions.

An alternative used in networks for the sigmoid function is the Threshold function t . The output assumes just two values: -1 or 1. Some threshold functions have a binary output: 0 or 1. This function is less complex to compute when a network is implemented on a digital computer than the

sigmoid function, but it is not useful in a backpropagation algorithm. An example of a network that uses a threshold function is the Boltzmann machine.

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.[12]

2.4 Firing Rules :

The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained. A simple firing rule can be implemented by using Hamming distance technique. The rule goes as follows: Take a collection of training patterns for a node, some of which cause it to fire (the 1-taught set of patterns) and others which prevent it from doing so (the 0-taught set). Then the patterns not in the collection cause the node to fire if, on comparison, they have more input elements in common with the 'nearest' pattern in the 1-taught set than with the 'nearest' pattern in the 0-taught set. If there is a tie, then the pattern remains in the undefined state.

For example, a 3-input neuron is taught to output 1 when the input (X1,X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001. Then, before applying the firing rule, the truth table is;

X1:	0	0	0	0	1	1	1	1
X2:	0	0	1	1	0	0	1	1
X3:	0	1	0	1	0	1	0	1
OUT:	0	0	0/1	0/1	0/1	1	0/1	1

TABLE 2.1 : Truth Table before applying firing rule

As an example of the way the firing rule is applied, take the pattern 010. It differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from 111 in 2 elements. Therefore, the 'nearest' pattern is 000 which belongs in the 0-taught set. Thus the firing rule requires that the neuron should not fire when the input is 001. On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1).

By applying the firing in every column the following truth table is obtained;

X1:	0	0	0	0	1	1	1	1
X2:	0	0	1	1	0	0	1	1
X3:	0	1	0	1	0	1	0	1
OUT:	0	0	0	0/1	0/1	1	1	1

TABLE 2.2 : Truth Table after applying firing rule

The difference between the two truth tables is called the **generalization** of the neuron. Therefore the firing rule gives the neuron a sense of similarity and enables it to respond

'sensibly' to patterns not seen during training. The single artificial neurons can now be interconnected in many different ways leading to a variety of neural networks with different architectures, learning rules and abilities. [4]

The most important ones are :

- Feedforward networks,
- Adaptive Resonance Theory (ART),
- Hopfield nets,
- Kohonen's self-organizing feature maps,
- Radial Basis Functions (RBF),
- Boltzmann-machines,

2.5 Single and Multi-layer Perceptrons :

A perceptron is a simple neural network model introduced by Frank Rosenblatt in 1958, and is perhaps the most widely used term in neural networks. A single layer perceptron is used to classify an input vector into several classes. In a single layer perceptron, the input values and activation level of the perceptron are either -1 or 1 ; weights are real-valued (between 0 and 1). The activation level is given by summing the weighted input values $\sum x_i w_i$. Perceptrons use a simple hard-limiting threshold function, where activation above a threshold results in an output value of 1, and -1 otherwise.

$$\begin{aligned} \text{Perceptron output} &= \text{sign}(\sum x_i w_i) \\ &= 1 \quad \text{if } \sum x_i w_i \geq t \\ &= -1 \quad \text{if } \sum x_i w_i < t \end{aligned}$$

The perceptron uses a simple form of supervised learning. The way a perceptron learns to distinguish patterns is through modifying its weights to reduce error. The adjustment for the weight Δw_i on the i^{th} component of the input vector is given by:

$$\Delta w_i = c x_i \delta$$

where c = learning rate

d = desired output

$$\delta = (\text{desired output}) - (\text{actual output}) = d - \text{sign}(\sum x_i w_i)$$

Single layer perceptrons can only solve problems where the solutions can be divided by a line (or hyperplane). The classes to be distinguished should be linearly separable. Therefore, a single layer perceptron cannot express non-linear decisions like the XOR problem.

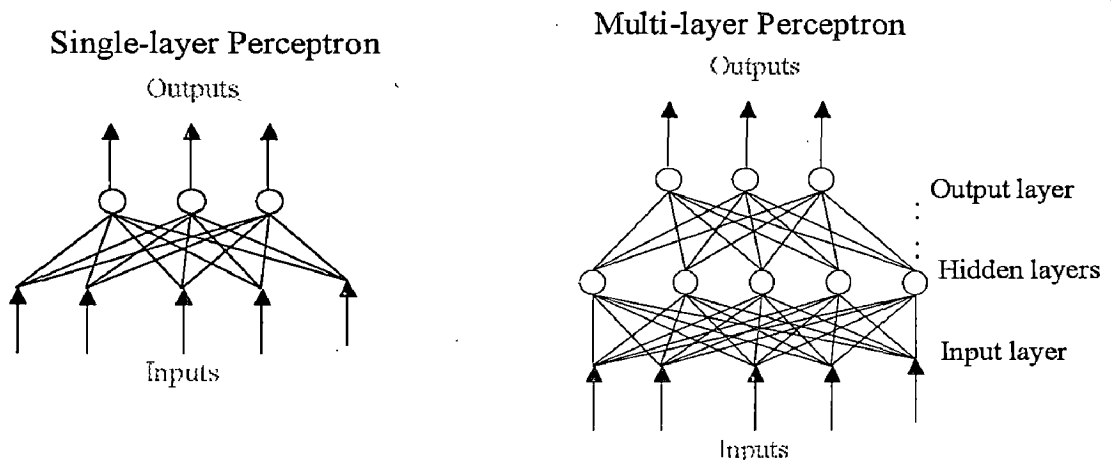


FIG 2.4 : Single-layer Perceptron and Multi-layer Perceptron

Multi-layer perceptrons are feed-forward nets with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the inputs and outputs. Multi-layer perceptrons overcome many of the limitations of the single layer perceptrons. The capabilities of multi-layer perceptrons stem from the nonlinearities used within nodes. In multi-layer networks, when adjusting a weight anywhere in the network, one has to be able to tell what effect this will have on the overall effect of the network. To do this, one has to look at the derivative of the error function with respect to that weight. The hard-limiter function for the single-layer perceptron is non-continuous, thus non-differentiable. The most popular continuous activation function used within backpropagation nets is the sigmoid function or the logistic function given by the equation:

$$f(\text{net}) = 1 / (1 + e^{-\lambda \cdot \text{net}}), \text{ where } \text{net} = \sum x_i w_i$$

As λ (called the squashing parameter) gets large, the sigmoid function approaches a linear threshold function over $\{0, 1\}$; as it gets closer to 1, it approaches a straight line. This activation function is non-linear, scaled and differentiable.

BACKPROPAGATION NEURAL NETWORK

3.1 Introduction To Backpropagation Neural Network :

The Backpropagation algorithm is perhaps the most widely used supervised training algorithm for multilayered feedforward networks. The backpropagation training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multilayer feedforward perceptron and the desired output. In the backpropagation algorithm, a feedforward phase is first done on an input pattern to calculate the net error. Then, the algorithm uses this computed output error to change the weight values in the backward direction. The error is slowly propagated backwards through the hidden layers - and hence its name.

The actual derivations for the different formulas used in the backpropagation algorithm come from the generalized delta rule. The delta rule is based on the idea of the error surface. The error surface represents cumulative error over a data set as a function of the network weights. Each possible network weight configuration is represented by a point on this error surface. By taking the partial derivative of the network error with respect to each weight we will learn a little about the direction the error of the network is moving. In fact, if we take the negative of this derivative (i.e. the rate change of the error as the value of the weight increases) and then proceed to add it to the weight, the error will decrease until it reaches a local minimum. The taking of these partial derivatives and then applying them to each of the weights, takes place starting from the output layer to hidden layer weights, then, from the hidden layer to input layer weights. A very simple way is to organize the neurons in several layers as shown in Figure 3.1 .

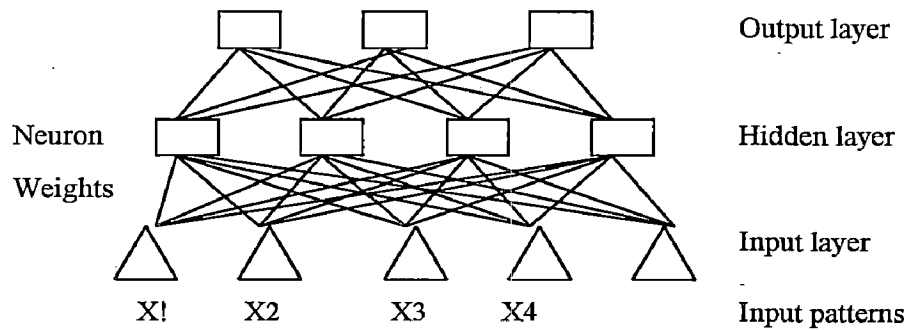


FIG 3.1 : Fully connected feedforward network with three layers

This architecture is called a feedforward net, since neurons of one layer are only connected with neurons of the succeeding layer, without any recurrent connections. Normally these nets consist of one input layer, one or two hidden layers (called hidden, since they don't have a direct connection to the outside world) and one output layer. With such a net, input data are mapped from the n dimensional input space to an m -dimensional output space. This net now has to learn to produce a certain desired output for each input pattern presented at the input layer.

The architecture shown in the fig below is called as the backpropagation net.

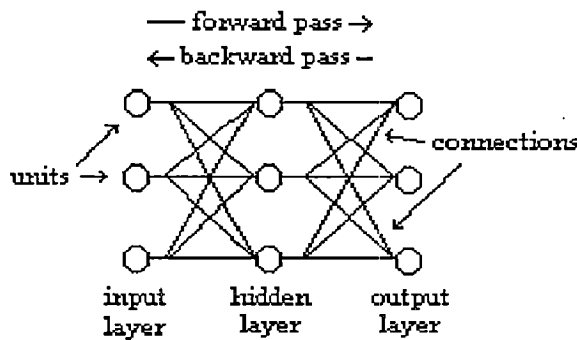


FIG 3.2 : Backpropagation Network

3.2 Backpropagation Training Algorithm :

The following description tends to assume a pattern classification problem, since that is where the BP network has its greatest strength. However, you can use back-propagation for many other problems as well, including compression, prediction and digital signal processing. When you present your network with data and find that the output is not as desired, what will you do? The answer is , we will modify some connection weights. Since the network weights are initially random, it is likely that the initial output value will be very far from the desired output. We wish to improve the behavior of the network. Which connection weights must be modified, and by how much, to achieve this objective? To put it another way, how do you know which connection is responsible for the greatest contribution to the error in the output? Clearly, we must use an algorithm which efficiently modifies the different connection weights to minimize the errors at the output. This is a common problem in engineering; it is known as optimization. The famous LMS algorithm was developed to solve a similar problem, however the neural network is a more generic system and requires a more complex algorithm to adjust the many network parameters. One algorithm which has hugely contributed to neural network fame is the back-propagation algorithm. The principal advantages of back-propagation are simplicity and reasonable speed . Back-propagation is well suited to pattern recognition problems. The training algorithm for a BPN consists of the following steps:

- Selection and Preparation of Training Data
- Modification of the neuron connection weights
- Repetition
- Running
- Hazards

3.2.1 Selection and Preparation of Training Data :

A neural network is useless if it only sees one example of a matching input/output pair. It cannot infer the characteristics of the input data for which you are looking for from only one example; rather, many examples are required. This is analogous to a child learning the difference between (say) different types of animals - the child will need to see several examples of each to be able to classify an arbitrary animal. If they are to successfully classify birds (as distinct from fish, reptiles etc.) they will need to see examples of sparrows, ducks, pelicans and others so that he or she can work out the common characteristics which distinguish a bird from other animals (such as feathers, beaks and so forth). It is also unlikely that a child would remember these differences after seeing them only once - many repetitions may be required until the information 'sinks in'. It is the same with neural networks. The best training procedure is to compile a wide range of examples (for more complex problems, more examples are required) which exhibit all the different characteristics you are interested in. It is important to select examples which do not have major dominant features which are of no interest to you, but are common to your input data anyway. One famous example is of the US Army 'Artificial Intelligence' tank classifier. It was shown examples of Soviet tanks from many different distances and angles on a bright sunny day, and examples of US tanks on a cloudy day. Needless to say it was great at classifying weather, but not so good at picking out enemy tanks. If possible, prior to training, add some noise or other randomness to your example (such as a random scaling factor). This helps to account for noise and natural variability in real data, and tends to produce a more reliable network. If you are using a standard unscaled sigmoid node transfer function, please note that the desired output must never be set to exactly 0 or 1! The reason is simple: whatever the inputs, the outputs of the nodes in the hidden layer are restricted to between 0 and 1 (these values are the asymptotes of the function. To approach these values would require enormous weights and/or input values, and most importantly, they cannot be exceeded. By contrast, setting a desired output of (say) 0.9 allows the network to approach and ultimately reach this value from either side, or indeed to overshoot. This allows the network to converge

relatively quickly. It is unlikely to ever converge if the desired outputs are set too high or too low. Once again, it cannot be overemphasized: a neural network is only as good as the training data! Poor training data inevitably leads to an unreliable and unpredictable network. Having selected an example, we then present it to the network and generate an output.

3.2.2 Modification of the neuron connection weights :

Consider the example in Figure designating (I1,I2), (H1,H2), and (O1, O2) as the inputs, hidden layer outputs and output-layer outputs respectively,

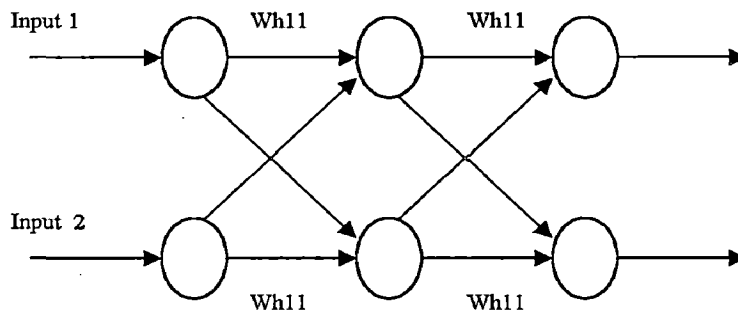


FIG 3.3 : example (2,2,2) BPN

the outputs of Hidden Node 1 and 2 are given by

$$H_1 = \text{sgm} \left(\sum_{l=1}^2 I_l W_{l1}^h \right) \text{-----} 1$$

$$H_2 = \text{sgm} \left(\sum_{l=1}^2 I_l W_{l2}^h \right) \text{-----} 2$$

Where

$$\text{sgm}(x) = \frac{1}{1 + e^{-x}} \text{-----} 3$$

The output-layer outputs are given by

$$O_1 = \text{sgm} \left(\sum_{m=1}^2 H_m W_{m1}^o \right) \text{-----} 4$$

$$O_2 = \text{sgm} \left(\sum_{m=1}^2 H_m W_{m2}^o \right) \text{-----} 5$$

Using 4 & 5

$$O_1 = \text{sgm} \left(\sum_{m=1}^2 \text{sgm} \left(\sum_{l=1}^2 I_{l1} W_{lm}^h \right) W_{m1}^o \right) \text{-----6}$$

$$O_2 = \text{sgm} \left(\sum_{m=1}^2 \text{sgm} \left(\sum_{l=1}^2 I_{l1} W_{lm}^h \right) W_{m2}^o \right) \text{-----7}$$

Now we can calculate the output given a particular set of inputs. This allows us to calculate the Mean Squared Error (MSE) between the actual output and the desired output for the given input in this training example. This is simply the average of the squares of the difference between what we want and what we got. Thus, our error function can be formally written as

$$E = \sum_{n=1}^2 (D_n - O_n)^2 \text{-----8}$$

or, using (6) and (7),

$$E = \sum_{n=1}^2 \left(D_n - \text{sgm} \left(\sum_{m=1}^2 \text{sgm} \left(\sum_{l=1}^2 I_{l1} W_{lm}^h \right) W_{mn}^o \right) \right)^2 \text{-----9}$$

where D_n is the n th desired output.

For example in the following example, suppose we have in the

output 0.75 and 0.05 and the desired outputs 0.9 and 0.1.

The (true) MSE is now ,

$$((0.9 - 0.75))^2 + (0.1 - 0.05)^2 / 2$$

which is equal to 0.0125. Clearly, for any given training example, this value is a function only of the weights of the network.

The gradient is fairly straightforward to calculate, due to the convenient fact that the derivative of the sigmoid function can be expressed in terms of the function itself:

$$\frac{d}{dx} \frac{1}{1 + e^{-x}} = \frac{-e^{-x}}{1 + e^{-x}} = (1 - \text{sgm}(x)) \text{sgm}(x) \text{-----10}$$

The gradient is defined as the vector of partial derivatives of the multivariate function with respect to each of variable. Because the error is a function of the network outputs, we first need to calculate a set of partial derivatives for each output node with respect to each associated connection weight. This turns out to be trivial, since all other variables but the one of interest are held constant when we calculate the partial derivative. Thus, only one linear term is left in the calculation of the partial derivative of the output, and leaving the coefficient - which is just the corresponding input! So, we can write

$$\frac{\partial O_n}{\partial W_{mn}^0} = \frac{\partial E}{\partial W_{mn}^0} \sum_{k=1}^2 W_{kn}^0 H_k = H_m \text{ -----11}$$

Now, the gradient of the error function can be calculated. Note

$$\begin{aligned} s^0 &= \sum_{k=1}^2 W_{kn}^0 H_k \\ \frac{\partial E}{\partial W_{mn}^0} &= \frac{\partial}{\partial W_{mn}^0} \sum_{n=1}^2 (D_n - O_n)^2 \\ &= 2 (D_n - O_n) \frac{\partial}{\partial s^0} \text{sgm}(s^0) \frac{\partial s^0}{\partial W_{mn}^0} \\ &= 2 (D_n - O_n) (1 - \text{sgm}(s^0)) \text{sgm}(s^0) H_m \text{ -----12} \end{aligned}$$

The new values for the network weights are calculated by multiplying the negative gradient with a step size parameter (called the **learning rate**) and adding the resultant vector to the vector of network weights attached to the current layer. This change does not take place, however, until after the middle-layer weights are updated as well, since this would corrupt the weight-update procedure for the middle layer. Clearly, the error at the output will be affected by the weights at the middle layer, too. However, the relationship is more complicated. A new gradient is derived, but this time the output weights are treated as constants rather than the hidden-layer weights. Now, the actual output is a function of the weights attached to the middle layer only (and in a generic network there are LM of those, for L input nodes and M middle-layer nodes).

Fortunately, it is still a relatively simple expression.

$$\frac{\partial E}{\partial W_{lm}^h} = ((1 - \text{sgm}(s^h)) \text{sgm}(s^h)) \sum_{n=1}^2 \delta_n^0 W_{mn}^0 I_l \quad \text{----13}$$

The middle weights are updated using the same procedure as for the output layer, and the output layer weights are updated as well. This is a complete training cycle for one piece of training data. It should be noted that the input layer is really only a buffer to hold the input vector. Therefore, it has no weights which need to be modified. However, in a more generic network, one may have more than one hidden layer. Again, the update procedure is quite similar. Once the modifications have been calculated, all weights (hidden and output) may be updated.

Please note : The above description assumes a (2, 2, 2) network.

The only difference in the mathematics resulting from a larger network are longer summations. All of the principles are the same. The training process is analogous to the biological process of learning - the strength of individual connections between the neurons increases or decreases as we learn.

3.2.3 Repetition :

Since we have only moved a small step towards the desired state of a minimized error, the above procedure must be repeated many times until the MSE drops below a specified value. When this happens, the network is performing satisfactorily, and this training session for this particular example has been completed. Once this occurs, randomly select another example, and repeat the procedure. Continue until you have used all of your examples many times ('many' may be anywhere between twenty or less and ten thousand or more, depending on the particular application, complexity of data and other parameters).

3.2.4 Running :

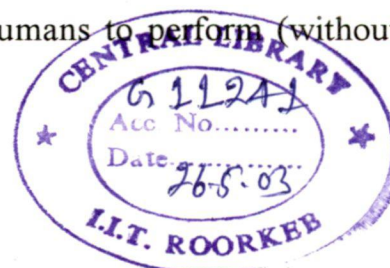
Finally, the network should be ready for testing. While it is possible to test it with the data you have used for training, this isn't really telling you very much. Instead, get some real data which the network has never seen and present it at the input. Hopefully it should correctly classify, compress, or otherwise process (however you trained it!) the data in a satisfactory way.

3.2.5 Hazards :

A consequence of the back-propagation algorithm is that there are situations where it can get 'stuck'. Think of it as a marble dropped onto a steep road full of potholes. The potholes are 'local minima' - they can trap the algorithm and prevent it from descending further. In the event that this happens, you can resize the network (add extra hidden-layer nodes or even remove some) or try a different starting point (i.e. randomize the network again). Some enhancements to the BP algorithm have been developed to get around this - for example one approach adds a momentum term, which essentially makes the marble heavier - so it can escape from small potholes. Other approaches may use alternatives to the Mean Squared Error as a measure of how well the network is performing.[9]

3.3 Existing System :

Before the age of the computer, there were many mathematical problems that humans could not easily solve, or more precisely (and this distinction is extremely important) humans were too slow in solving. Computers enabled these often simple but slow and tedious tasks to be performed quickly and accurately. The first problems solved with computers were calculating equations to resolve important physical problems, and later displaying a nice GUI, making word processors and so on. However, there are many common tasks which are trivial for humans to perform (without even



any conscious effort) yet which are extremely difficult to formulate in a way that a computer may easily solve.

These include:

- Signal processing such as (pattern recognition, voice recognition, image processing etc.)
- Compression
- Data reconstruction (e.g. classification where part of the data is missing)
- Data mining
- Data simplification

3.3.1 Template Matching :

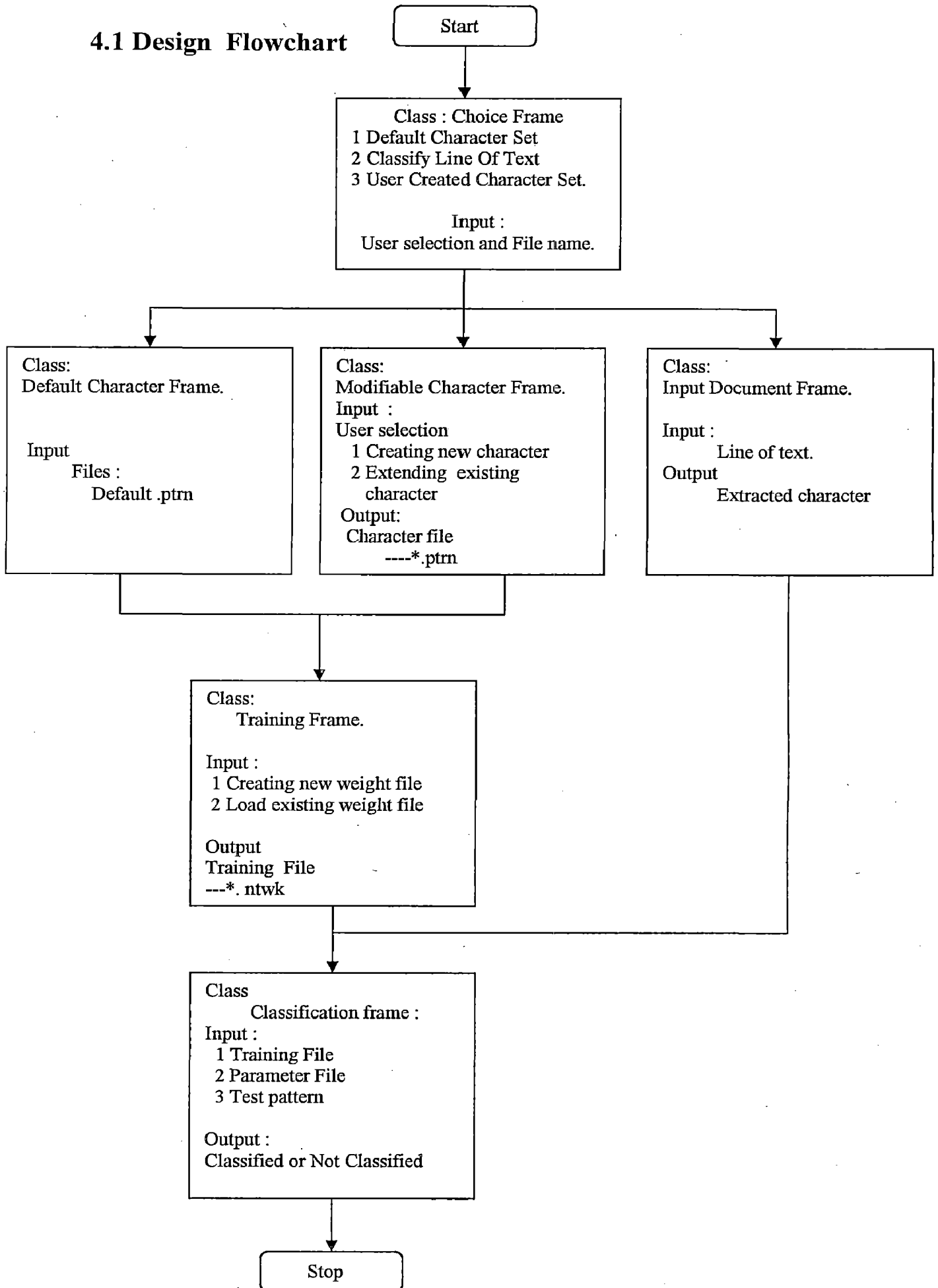
Earlier techniques for pattern recognition's, include the technique of Template Matching. In this technique the patterns are just matched together as a human compare two structures with their exact features & characteristics matching. Template Matching are oversensitive to shift in position and distortions in shape of the stimulus patterns, and it is necessary to normalize the position and the shape of stimulus pattern beforehand. A good method for normalization have not been developed yet. Therefor, the finding of an algorithm for character recognition which can cope with shift in position and distortion has long been desired. In this project, we implement an algorithm which gives an important solution to this problem. The algorithm used here can be realized with a multilayered network consisting of neuron like cells. It is organized by supervised learning and acquires the ability for correct character recognition. So, naturally, scientists, engineers and mathematicians tried to make an intellectual abstraction which would enable a computer work in a similar way to that in which the human brain works – a neural network. [9]

DESIGN AND IMPLEMENTATION

Word Recognition using Artificial Neural Networks, implemented the back-propagation neural network for developing word recognition software. . The word recognition software is efficiently recognizes all alphanumeric characters which the user writes on the screen. The user has to train the network . Design flowchart is given in the section 4.1

Class Choice Frame which takes the input from the user, User selection and File name. when the user select the Default Character Set it load s the default file .which is the number file. selection of Classify Line Of Text invoke the Input Document Frame ,which takes the input from the user that is line of text. and given the extracted patterns as output and finally the option for creating the or load the patterns is given , which invoke the class Modifiable Character Frame. Which create the new pattern file *,ptrn when the user clicks on he train button the Training Frame class is invoked it either create or load the weight file as per the user selection. and creates the training file *. ntwk. when the user select the classify option Classification frame class is invoked,. which takes the input Training File , Parameter File .and Test pattern . and gives the output whether the pattern is classified or not.

4.1 Design Flowchart



4.2 Implementation Strategy :

To create an ANN through the means of software, object oriented programming is required because a neuron resembles several components, and OOP is the best choice due to its capability of creating objects that contains different variables and methods. The first step is to create an object that simulates the neuron. The object would contain several functions and variables including weight (a random number generated when the neuron is created, similar to the synapse in BNN), a non-linear function (to determine whether to activate the neuron or not), a method that adds up all the inputs, and a bias/offset value (optional) for the characterization of the neuron.

After the object is created, the next step is to create a network. A typical ANN has three layers: input layer, hidden layer and output layer. The input layer is the only layer that receives signals outside the network. The signals are then sent to the hidden layer, which contains interconnected neurons for pattern recognition and relevant information interpretation. Afterwards, the signals are directed to the final layer for outputs. Usually a more sophisticated neural network would contain several hidden layers and feedback loops to make the network more efficient and to interpret the data more accurately. Using figure 3.1 as a model, the network is like a big matrix. However, it would be easier if the three layers were separated into three small matrixes. Each small matrix will contain neurons and when signals are inputted, the neurons will send inputs through the non-linear function to the next neuron. Afterward, the weight of the neuron is increased or decreased.

4.3 Algorithm: Backpropagation

Given : A set of input-output vector pairs.

Compute : A set of weights for a three-layer network that maps inputs onto Corresponding outputs.

1. Let A be the number of units in the input layer, as determined by the length of the training input vectors. Let C be the number of units in the output layer. Now choose B , the number of units in the hidden layer. The input and hidden layers each have an extra unit used for thresholding; therefore, the units in these layers will sometimes be indexed by the ranges $(0, \dots, A)$ and $(0, \dots, B)$. We denote the activation levels of the units in the input layer by x_j , in the hidden layer by h_j , and in the output layer by o_j . Weights connecting the input layer to the hidden layer are denoted by w_{1ij} , where the subscript i indexes the input units and j indexes the hidden units. Likewise, weights connecting the hidden layer to the output layer are denoted by w_{2ij} , with i indexing to hidden units and j indexing output units.

2. Initialize the weights in the network. Each should be set randomly to a number between -0.1 and 0.1.

$w_{1ij} = \text{random}(-0.1, 0.1)$ for all $i = 0, \dots, A, j = 1, \dots, B$

$w_{2ij} = \text{random}(-0.1, 0.1)$ for all $i = 0, \dots, B, j = 1, \dots, C$

3. Initialize the activation of the network. The values of these thresholding units should never change.

$$x_0 = 1.0$$

$$h_0 = 1.0$$

4. Choose an input-output pair. Suppose the input vector is x_i and the target output vector is y_i . Assign activation levels to the input units.

5. Propagate the activation's from the units in the input layer to the units in the

$$h_j = \frac{1}{1 + e^{-\sum_{i=0}^B w_{1ij} h_i}} \quad \text{for all } j = 1, \dots, C$$

hidden layer using the activation functions

Note that i ranges from 0 to A . w_{10j} is the thresholding weight for hidden unit j (its propensity to fire irrespective of its inputs).

x_0 is always 1.0.

6. Propagates the activation's from the units in the hidden layer to the units in the output layer. Again, the thresholding weight w_{2j} for output unit j plays a role in the weighted summation. h_0 is always 1.0.

$$h_j = \frac{1}{1 + e^{-\sum_{i=0}^B w_{2ij} h_i}} \quad \text{for all } j = 1, \dots, C$$

7. Compute the errors of the units in the output layer denoted δ_{2j} . Error are based on the network's actual output (o_j) and the target output (y_j).

$$\delta_{2j} = o_j(1 - o_j)(y_j - o_j) \quad \text{for all } j = 1 \dots B$$

8. Compute the errors in the units in the hidden layer, denoted

$$\delta_{1j}$$

$$\Delta_{1j} = h_j(1 - h_j) \sum_{i=1}^C \delta_{2i} \times w_{ji} \quad \text{for all } j = 1, \dots, B$$

9. Adjust the weights between the hidden layer and the output layer. The learning rate denoted η ; its functions is in the same as in perception learning. A reasonable value of η is 0.35. Δ is denoted by

$$\Delta w_{2ij} = \eta \cdot \delta_{2j} \cdot h_i \quad \text{for all } i = 0, \dots, B, j = 1, \dots, C$$

10. Adjust the weights between the input layer and the hidden layer.

$$\Delta w_{1ij} = \eta \cdot \delta_{1j} \cdot h_i \quad \text{for all } i = 0, \dots, A, j = 1, \dots, B$$

11. Go to step 4 and repeat. When all the inputs-output pairs have been presented to the network, one epoch has been completed. Repeat steps 4 to 10 for as many epochs as desired. [2]

RESULTS AND DISCUSSION

User Interface :

1. Choice frame.

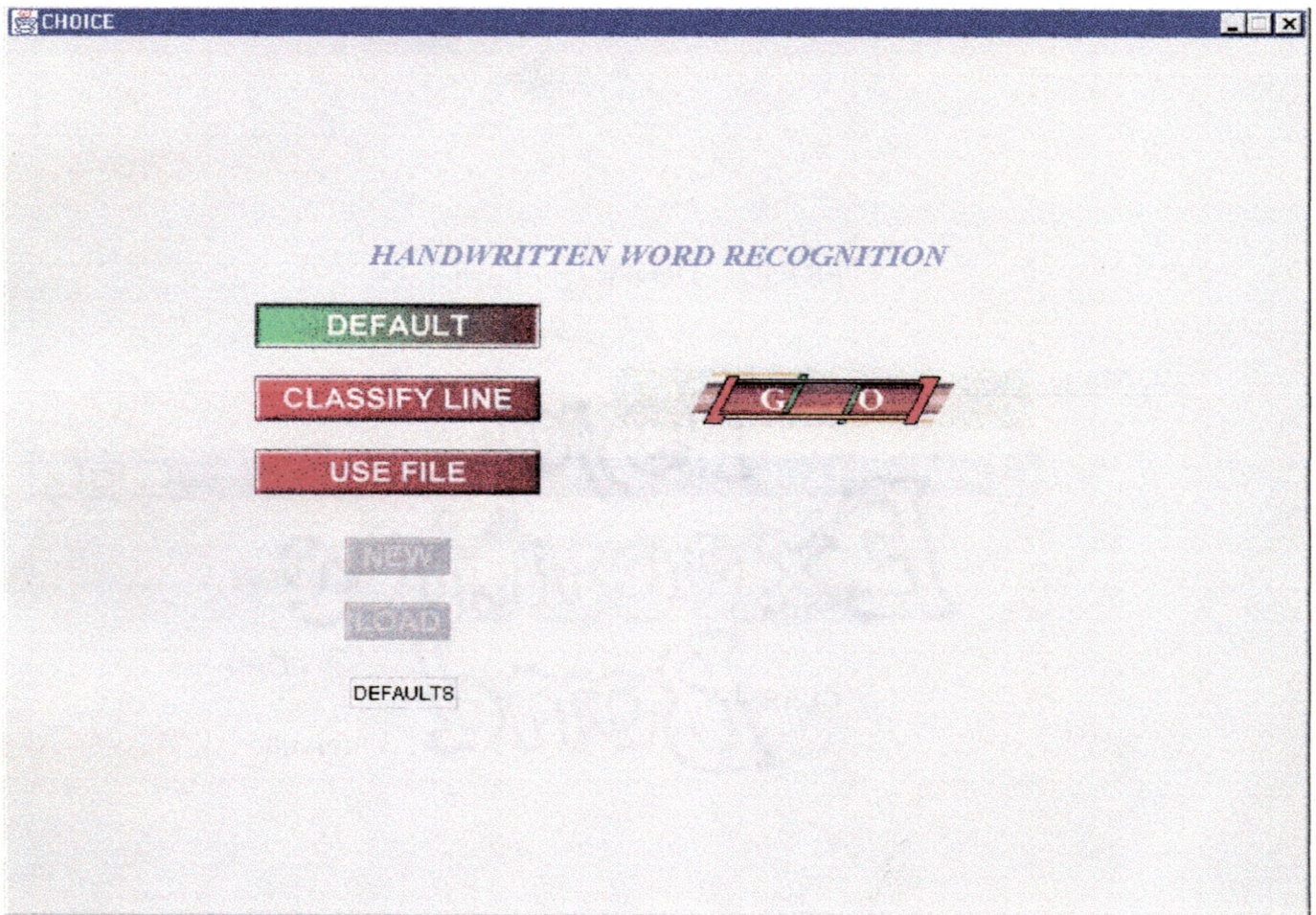


FIG : 5.1 : Screen displaying Opening Menu

2. Default Patterns

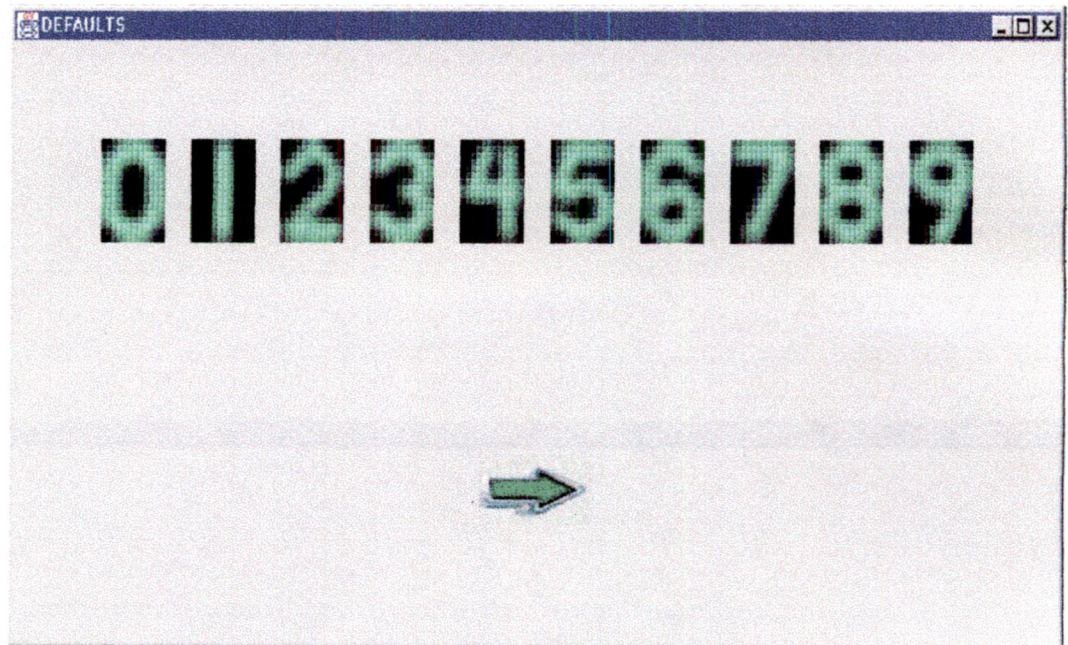


FIG 5.2 : Default set of patterns

3. Mode select

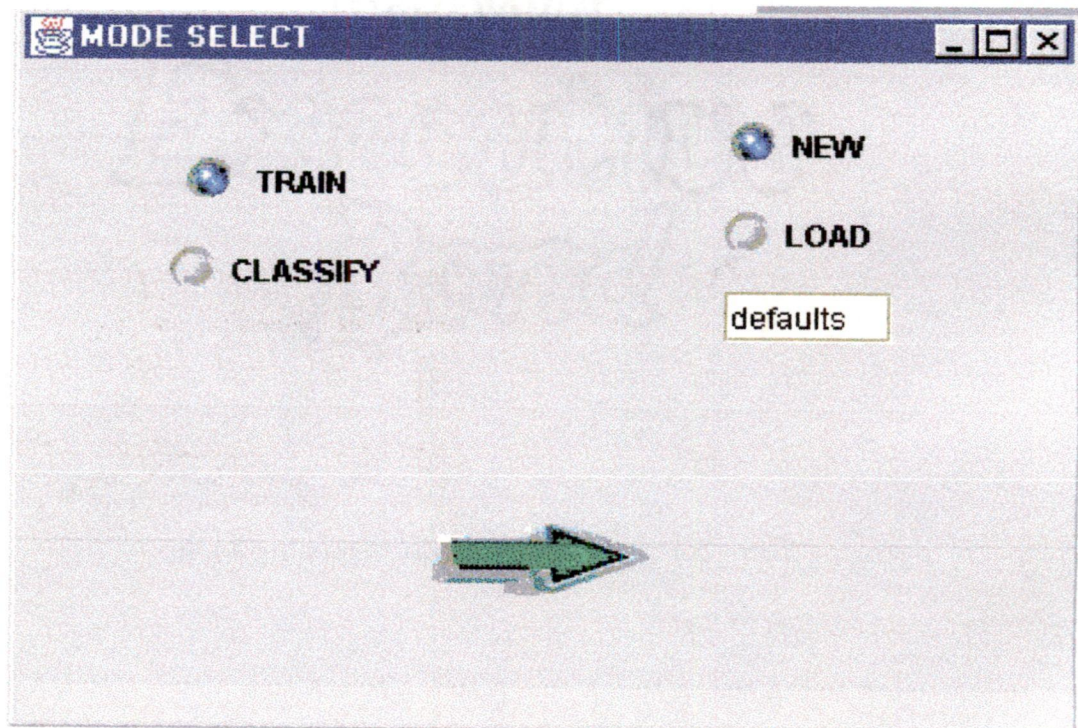


FIG 5.3 : The mode select frame

4. Training frame

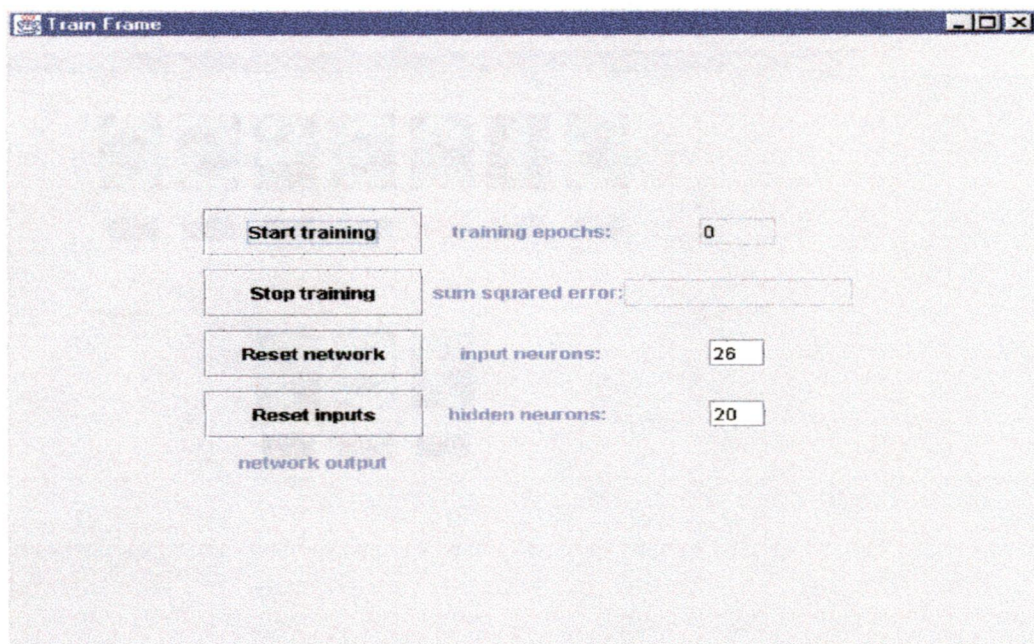


FIG 5.4 : The training frame.

5. Classify Frame.

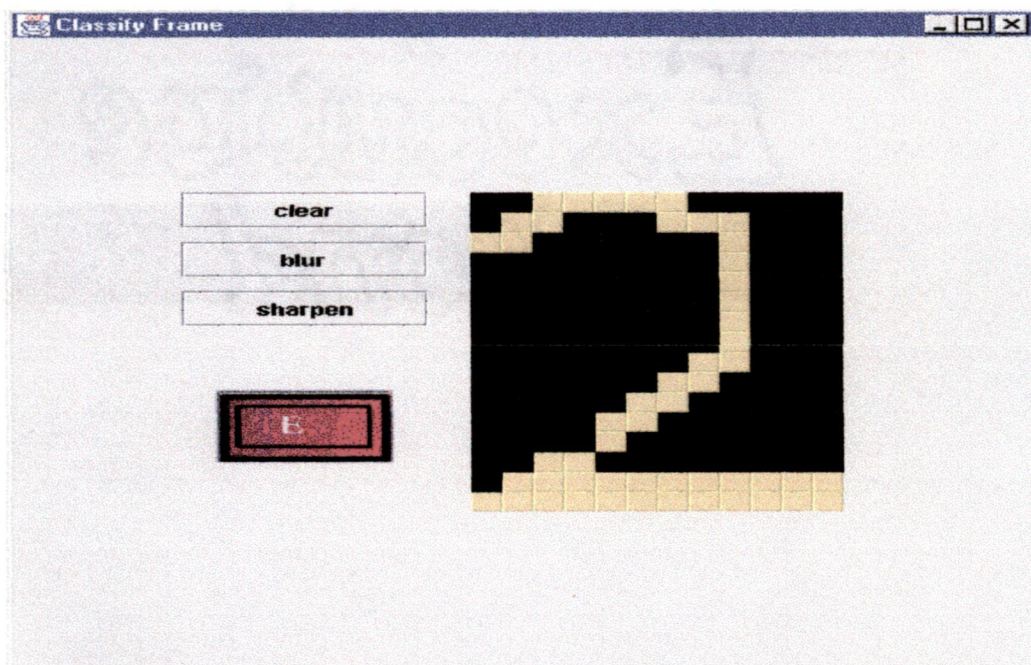


FIG 5.5 : Classification Frame

6 Output frame :



FIG 5.6 : The Output Frame

7 Line Classify :

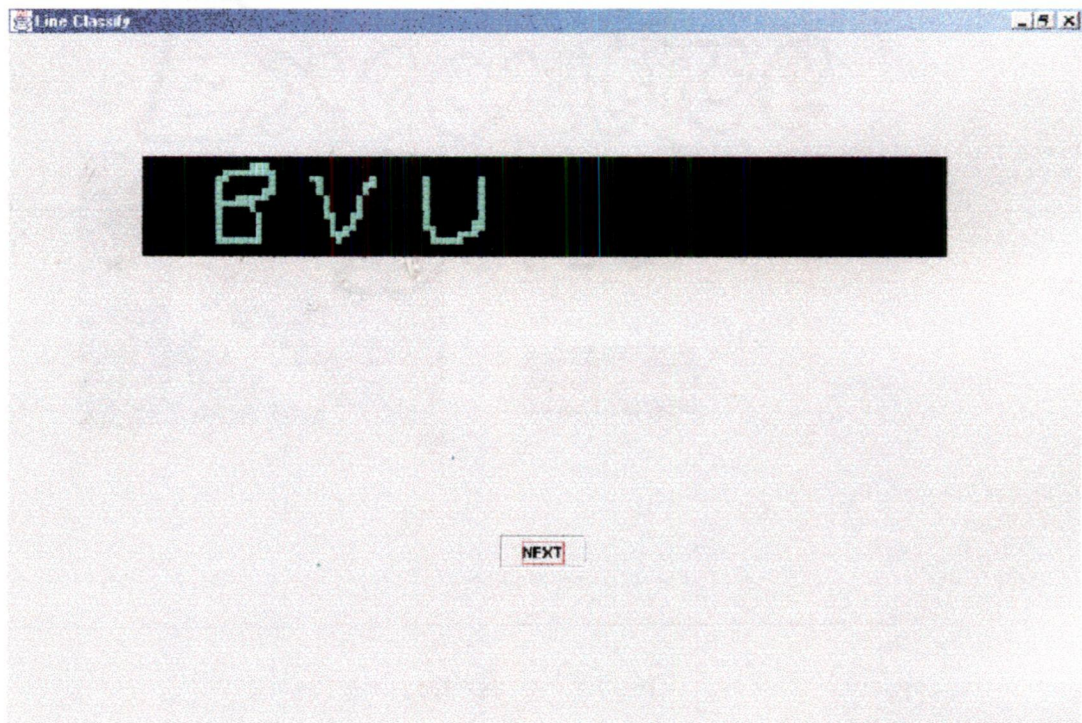


FIG 5.7 : Line Classification frame .

8 Classified Patterns :



FIG 5.8 : The Classified Pattern

9 Custom patterns :

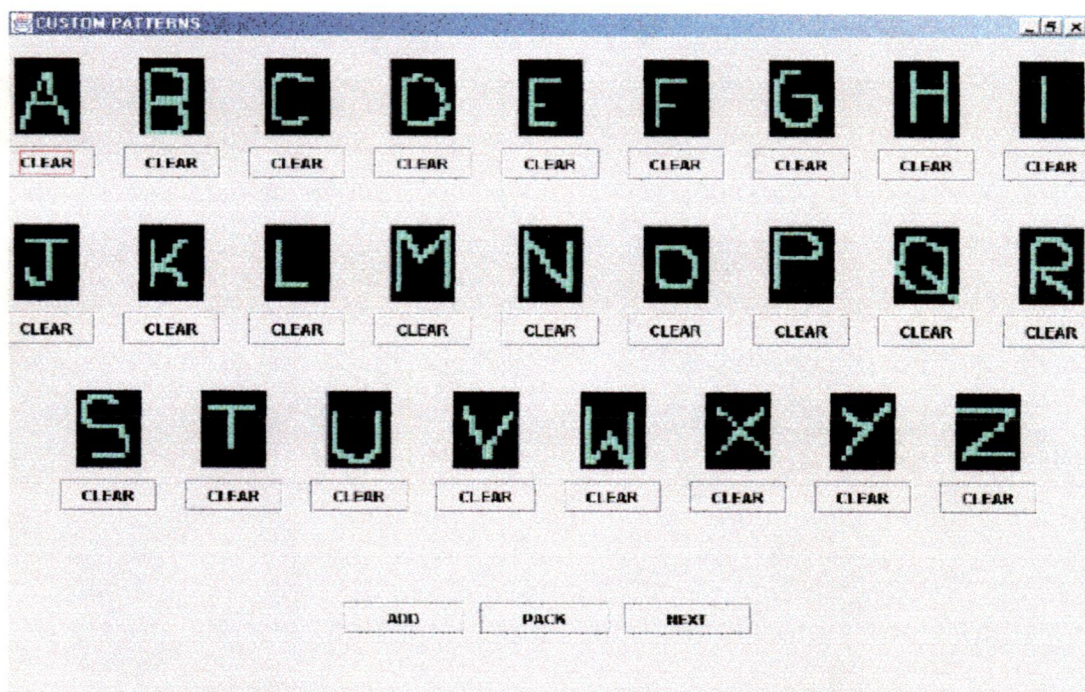


FIG 5.9 : Default set of patterns

A program is written in Java to perform optical character recognition using backpropagation neural networks. The program is working in the following ways.

When the program starts, a main CHOICE screen is displayed. Where the user can load the default letter file, classify the line and can load or create new pattern file. When the user clicks on DEFAULT button the next default frame is displayed. click on NEXT button to get the MODE frame as shown above. Where the options like Train Classify, New, and Load are available. When user clicks on train button training frame is displayed on the screen. To train the network, click the "Start Training" button. As the training proceeds, the number of learning cycles increases and the sum squared error decreases. After a while, the sum squared error decreases to a low value (of ≤ 0.01). At this point, press the "Stop Training" button. The network could be tested with any user drawn symbol by drawing a symbol on the user input icon panel with a mouse. There are 12 X 16 pixels in the icon panel. The left mouse button will draw a pixel. To see what the network thinks the user drew a picture of, press the "test" button at the left of the icon panel, and see which of the lights glows red. The "CLEAR" button can be used to clear the current user drawn symbol to draw a new one. The "RESET" button resets the neural network, so that training can be started again. To get the classification frame click on the CLASSIFY LINE button on the CHOICE frame. on this frame the user can test any word by drawing the symbols on the user input icon panel with a mouse. To see the output click on the next button. The user has given the option to create or to load the pattern files with the help of USE FILE button.

Currently, the program has been tested with two sets of training patterns – letters and numbers. The target patterns and classified patterns are shown in the next section.

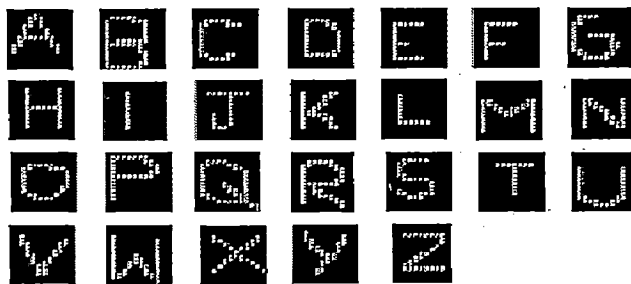
Trained Patterns And Classified Patterns :

Trained Patterns :

Set 1



Set 2



Classified Patterns :

Set 1



Set 2



CONCLUSION

The BPN network designed in this project has the ability to recognize stimulus patterns without affecting by shift in position nor by a small distortion in shape of input pattern. It also has a function of organization, which processes by means of "Learning with a Teacher" (Supervised Learning). If sets of input patterns are repeatedly presented to it, it gradually acquires the ability to recognize these patterns. The performance of the network has been demonstrated by simulating on a computer.

The design of information processing proposed in this project is of great me not only as an inference upon the mechanism of brain but also to the field of Engineering. One of the largest and longstanding difficulty is in designing a pattern recognizing machine has been the problem how to cope with the shift in position and the distortion in the shape of the input patterns. The network designed in this project gives a partial solution to this difficulty.

REFERENCES

1. James A. Anderson " An Introduction To Neural Networks " PHI 1998.
2. Elaine Rich And Kevin Knight , "Artificial Intelligence" TMH, 1991.
3. Google search available at
"http://www.shef.ac.uk/psychology/gurney/notes/index.html"
4. Google search available at
"http://www.emsl.pnl.gov:2080/proj/neuron/neural/what.html "
5. Google search available at
"http://www.willamette.edu/~gorr/classes/cs449/intro.html
6. Google search available at
"http://www.faqs.org/faqs/ai-faq/neural-nets/part 1-6 "
7. Google search available at "http://www.generation5.org"
8. Google search available at
"http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/a_gloss.shtml"
9. Google search available at
"http://ieee.uow.edu.au/~daniel/software/libneural/BPN_tutorial/BPN_English/BPN_English/BPN_English.html"
10. Google search available at
"http://psychology.about.com/gi/dynamic/offsite.htm?site=http%3A%2F%2Fwww.dacs.dtic.mil%2Ftechs%2Fneural%2Fneural_ToC.html",
11. Google search available at
"www.cedar.buffalo.edu/~jaehwap/papers/thesis.pdf "
12. Google search available at
"http://claymore.engineer.gvsu.edu/eod/software/software-162.html"
13. Dr. Roger S. Gaboriski, " Introduction to Artificial Intelligence "
14. James A. Freeman and David M. Skapura , " Neural Networks "

USER MANUAL

A.1 For Choice Frame

Buttons :



Default button is used to load the default letter file



Classify Line : to classify the line of character.



Use File : Activate the NEW and LOAD buttons .



New : To create new pattern file



Load : To load stored pattern file.



Go Button : To start the user defined task

A.2 For Default Pattern Frame.



This is the default letter file .

A.3 For Mode Select Frame

Buttons :



Train : to get the training frame.



Classify : To get the classification frame.



New : To get the new file



Load : To load the already created pattern file.

A.4 For Training Frame

Buttons :



Start Training : To start training .



Stop Training : To stop training .



Reset Network Reset" button resets the neural network, so that training can be started again.



Reset Input Reset" button resets the neural network, so that training can be started again.

Text Fields :

- Training epochs : refer Appendix C
- Sum Squared Error : refer Appendix C
- Input Neuron : the number of neurons in the input layer. This field is set to 26
- Hidden Neuron : the number of neurons in the hidden layer , set to 20.

A.5 For Classify Frame :

Buttons



Clear : used to clear the current user drawn symbol to draw a new one



Blur



Sharpen to sharpen the pattern



Test To see what the network thinks the user drew a picture of, press the "test" button

A.6 For Line Classify Frame

Button :



Next : to get the classification frame where the classified patterns are shown.

A.7 For Classified Patterns Frame

Shows the classified pattern inputted by the user

A.8 For Custom Frame

Buttons :



Add to add the patterns



Pack to pack the patterns.



Next :when the user clicks on next mode select frame is displayed for further operation.

GLOSSARY

- **bias** - A neuron parameter that is summed with the neuron's weighted inputs and passed through the neuron's transfer function to generate the neuron's output.
- **classification** - An association of an input vector with a particular target vector.
- **connection** - A one-way link between neurons in a network
- **epoch** - The presentation of the set of training (input and/or target) vectors to a network and the calculation of new weights and biases. Note that training vectors can be presented one at a time or all together in a batch
- **error vector** - The difference between a network's output vector in response to an input vector and an associated target output vector
- **generalization** - An attribute of a network whose output for a new input vector tends to be close to outputs for similar input vectors in its training set
- **gradient descent** - The process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases. This is done to minimize network error.
- **learning** - The process by which weights and biases are adjusted to achieve some desired network behavior.
- **learning rate** - A training parameter that controls the size of weight and bias changes during learning.
- **mean square error function** - The performance function that calculates the average squared error between the network outputs a and the target outputs t .
- **momentum** - A technique often used to make it less likely for a backpropagation networks to get caught in a shallow minima
- **squashing function** - A monotonic increasing function that takes input values between
- infinity and + infinity and returns values in a finite interval.

- **weight matrix** - A matrix containing connection strengths from a layer's inputs to its neurons. The element w_{ij} of a weight matrix W refers to the connection strength from input j to neuron i .
- **local minimum** - The minimum of a function over a limited range of input values. A local minimum may not be the global minimum.
- **log-sigmoid transfer function** - A squashing function of the form shown below that maps the input to the interval (0,1). (The toolbox function is `logsig`.)
- **local minimum** - The minimum of a function over a limited range of input values. A local minimum may not be the global minimum.
- **log-sigmoid transfer function** - A squashing function of the form shown below that maps the input to the interval (0,1).

$$f(n) = \frac{1}{1 + e^{-n}}$$

