International Journal of Computer Applications in Engineering Sciences
[VOL III, ISSUE IV, DECEMBER 2013] [ISSN: 2231-4946]

Back Propagation Neural Network for Recognition of Dynamic Objects

Jajati Mallick[1], Prangyan Prakash Mohapatra[2] , Dulal Acharjee[3]

[1]*Dept. of CSE,* [2]*Dept. of ETE,* [3]*Dept. of IT*
Purushottam Institute of Engineering and Technology
Mandiakudar-770034, Sundargarh, Rourkela, Odisha, India.
mallick.jajati@gmail.com[1], prangyanm@gmail.com[2], dulalacharjee@gmail.com[3]

***Abstract*—This article gives all-inclusive idea into one of the main branches of pattern recognition using neural network. The algorithm, developed in this work, can recognize any static and also dynamic object using methods of the same class which is one of the main unique features of this research work. Using this idea, Back Propagation Neural Network for Object Recognition (BPNNOR) is experimented and presented in this article.**

In this work, a new algorithm is implemented for recognizing any objects like: images, characters or any type of two dimensional digital objects. Using object oriented concept embedded within existing BPN, data fitting rate is increased for recognizing critical nonlinear pattern of objects.

***Keywords*—BPNNOR, Artificial Neural Network, dynamic object.**

I. INTRODUCTION

A trained neural network can be thought of as an expert in the category of information it has been given to analyze. If all the information belongs to same class then the neural network recognize the objects (pattern) belongs to the same class. A natural scheme of neural network is implemented on three dimensional objects for recognizing it and using iteration of processes maximizes the percent to match an object. An object can recognize using class analysis rather than structural approach.

Neural networks are parallel processing structures that provide the capability to perform various pattern recognition tasks. A network is typically trained over a set of exemplars by adjusting the weights of the interconnections using a back propagation algorithm. This gradient search converges to locally optimal solutions which may be far removed from the global optimum. An Artificial Neural Network is a network of many very simple processors ("units"), each possibly having a (small amount of) local memory. The units are connected by unidirectional communication channels ("connections"), which carry numeric (as opposed to symbolic) data. The units operate only on their local data and on the inputs they receive via the connections. The design motivation is what distinguishes neural networks from other mathematical techniques: A neural network is a processing device, either an algorithm**,** or actual hardware, whose design was motivated by the design and functioning of human brains and components thereof. There are many different types of Neural Networks, each of which has different strengths particular to their applications. The abilities of different networks can be related to their structure**,** dynamics and learning methods. Neural Networks offer improved performance over conventional technologies in areas which includes Machine Vision**,** Robust Pattern Detection**,** Signal Filtering**,** Virtual Reality, Data Segmentation**,** Data Compression**,** Data Mining**,** Text Mining, Artificial Life**,** Adaptive Control, Optimization and Scheduling**,** Complex mapping and more. Neural networks are members of a class of software that has the potential to simulate characteristics of biological thinking and learning. Owing their origins to the study of the human brain, neural networks possess the ability to acquire and store knowledge. They are well suited for domains of non-linearity and high complexity that are ill defined, unknown, or just too complex for standard programming practices. Rather than explicit programming, learning algorithms are used to adjust neural networks to fulfill the needs of a desired function.

A. Operation of Neurons

Let us consider what each individual neuron actually does. At the simplest level, neurons produce pulses, called 'Action Potentials' and they do this when stimulated by other neurons (or, if they are sensory neurons, by outside influences, which they pick up through their modified dendrites). When a neuron is at rest, before it become stimulated, it is said to be polarized [1, 2]. This means that, although the neuron is not receiving any electrical signal from other neurons, it is charged up and ready to produce a pulse. Each neuron has associated with it upto a level of stimulus, above which a nerve pulse or action potential will be generated. Only, when it receives enough stimulation from one or more sources can initiate a pulse. The mechanism by which the pulses travel and the neuron maintains its general

electrical activity is rather complex and we need not go into it in detail here. It works through an exchange of ions in the fluid that surrounds the cell, rather than by the flow of electrons as you would get in a wire. This means that signals travel very slowly - at a couple of hundred meters per second. The pulse, which the neuron generates and travels down the axon, is shown in figure 1.

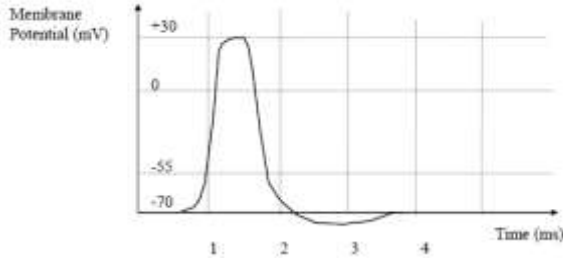

Figure 1 The action of potential.

Because, these pulses are only a couple of milliseconds wide, they often appear as spikes if viewed on an oscilloscope screen. So, if one neuron is receiving lots of stimulation from another (receiving lots of pulses through its dendrites) then it will itself produce a strong output - that is more pulses per second. The history of Artificial Neural Networks began in 1943 when Warren McCulloch and Walter Pitts [3] proposed a simple artificial model of the neuron. Their model (in a slightly modified and improved form) is what most Artificial Neural Networks are based on to this day. Such simple neurons are often called *Perceptions*. The basic neuron is shown in figure 2.

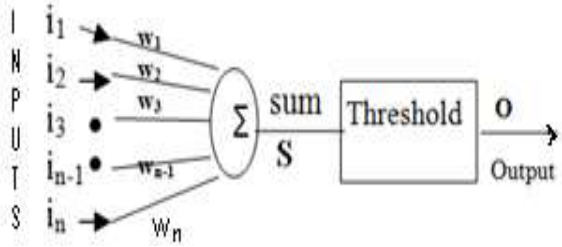

Figure 2 A basic Artificial Neuron

B. The basic Artificial Neuron

The figure 2 shows that the inputs to the neuron are represented by *i* (in the biological sense, these are the activities of the other connecting neurons or of the outside world, transmitted through the dendrites). Each input is weighted by a factor which represents the strength of the synaptic connection of its dendrite, represented by *w*. The sum of these inputs and their weights is called the *activity* or *activation* [5] of the neuron and is denoted by *S*. In mathematical terms:

$S = i_1w_1 + i_2w_2 + .. + i_nw_n$ (1)

A threshold is then applied, which is a simple binary level:

if S >=0.5 then O = 1

if S <0.5 then O = 0 (2)

Threshold value is set at 0.5.

So, the neuron takes its inputs, weights them according to how strong the connection is and if the total sum of the weighted inputs is above a certain threshold, the neuron '*fires*', just like the biological one, as a neuron learns by changing its weights. Early Artificial Neural Networks used simple binary outputs in this way. However, it was found that a continuous output function was more flexible. One example is the Sigmoid Function [4].

$$O = \frac{1}{1+e^{-s}} \quad (3)$$

This function simply replaces the Threshold Function and produces an output which is always between zero and one, it is therefore often called a Squashing (or Activation) Function. Other Activation Functions are also sometimes used, including: Linear, Logarithmic and Tangential Functions; however, the Sigmoid Function is probably the most common. Figure 1.3 shows how the Sigmoid and Threshold Functions compare with each other. A output of *n* inputs of neurons can be written as:

$$S = \sum_{x=1}^{x=n} w_x\, i_x \quad (5)$$

Or, if the inputs are considered as forming a vector $\bar{I}$, and the weights as vector of matrix $\bar{W}$, then output may be written as:

$$S = \bar{I} \,.\, \bar{W} \quad (6)$$

In this case the normal rules of matrix arithmetic apply. The Output Function remains a Threshold or Sigmoid. This model is not the only method of representing an Artificial Neuron; however, it is by far the most common. A Neural Network consists of a number of these neurons connected together.

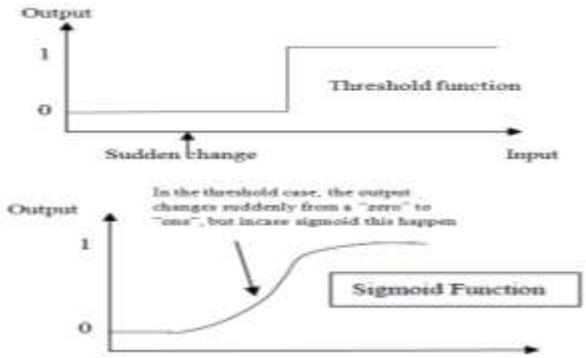

Figure 3 The Threshold and Sigmoid Functions.

II. PROBLEM DEFINATION

In static objects, the features of objects are finite and known previously, but in case of dynamic objects features are changeable and it is hard to recognize those features. It is required to develop a system which can recognize any objects in motion or changeable with respect to time domain. Objects, visible for a short time, are hard to

detect and recognize. This work, aims to formulate dynamic objects which have a short lives.

III. RECOGNISING PATTERNS

In the late 1950s Rosenblatt [6] succeeded in making the first successful practical networks, to everyone's surprise they turned out to have some remarkable qualities, one of which was that they could be used for pattern recognition. Let's take a very simple example to see how this works and shown in figure 4.

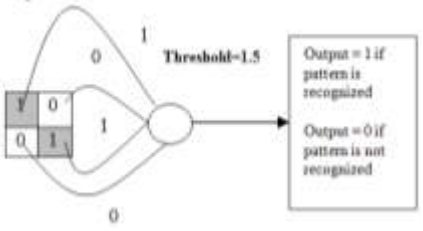

Figure 4 Recognising a simple pattern.

Here, we have a simple picture with four pixels. Each shaded pixel is given a value 1 and each white pixel a value 0. These are connected up to the neuron, with the weights as shown. The total sum of the neuron is (1 x 1) + (0 x 0) + (1 x 1) + (0 x 0) = 2. So, the neuron's output is 1 and it recognizes the picture. Even if there were some "interference" to the basic picture (some of the white pixels weren't quite 0 and some of the shaded ones weren't quite 1) the neuron would still recognize it, as long as the total sum was greater than 1.5. So the neuron is "Noise Tolerant" or able to "Generalize" which means it will still successfully recognize the picture, even if it isn't perfect - this is one of the most important attributes of Neural Networks.

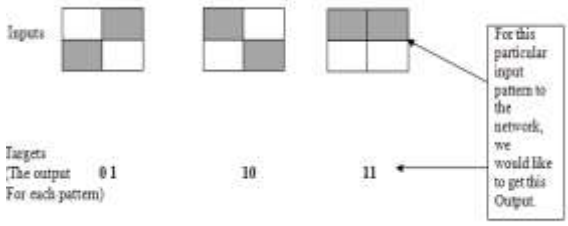

Figure 5 A Back Propagation training set.

IV. THE BACK PROPAGATION ALGORITHM

The Back Propagation is the training or learning algorithm rather than the network itself. A Back Propagation network learns by example we give the algorithm examples of what we want the network to do and it changes the network's weights so that, when training is finished. It will give the required output for a particular input. Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks [7]. As just mentioned, to train the network you need to give it examples of what you want the output you want (called the *Target*) for a particular input as shown in Figure 5. So, if we put in the first pattern to the network, we would like the output to be 0 1 as shown in figure 5 (a black pixel is represented by 1 and a white by 0 as in the previous examples). The input and its corresponding target are called a *Training Pair*. Once the network is trained, it will provide the desired output for any of the input patterns.

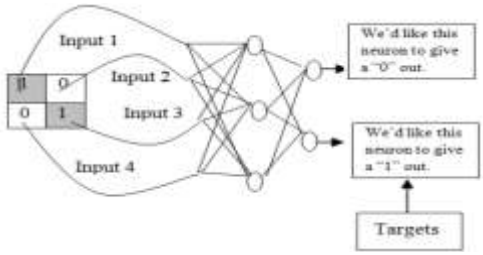

Figure 6 Applying a training pair to a network.

Let's now look at how the training works. The network is first initialized by setting up all its weights to be small random numbers – say between –1 and +1. Next, the input pattern is applied and the output calculated (this is called the *forward pass*). The calculation gives an output which is completely different to what we want (the Target), since all the weights are random. We then calculate the *Error* of each neuron, which is essentially [8]: *Target – Actual Output.* This error is then used mathematically to change the weights in such a way that the error will get smaller. In other words, the Output of each neuron will get closer to its Target (this part is called the *reverse pass*). The process is repeated again and again until the error is minimal. Let's consider an example with an actual network to see how the process works. We'll just look at one connection initially, between a neuron in the output layer and one in the hidden layer, figure 7. The connection we are interested in is between neuron A (a hidden layer neuron) and neuron B (an output neuron) and has the weight W_{AB} .The diagram also shows another connection, between neuron A and C, but we'll return to that later. The algorithm works like this [9]:

i. First apply the inputs to the network and work out the output, this initial output could be anything, as the initial weights were random numbers.

ii. Next work out the error for neuron B. The error is, in other words:

$$Error_B = Output_B (1-Output_B)(Target_B – Output_B)$$

The "*Output(1-Output)*" term is necessary in the equation because of the Sigmoid Function – if we were only using a threshold neuron it would just be *(Target – Output).*

iii. Change the weight. Let $\mathbf{W^+_{AB}}$ be the new (trained) weight and $\mathbf{W_{AB}}$ be the initial weight.

$$W^+_{AB} = W_{AB} + (Error_B \times Output_A)$$

It is the output of the connecting neuron (neuron A) we use (not B). We update all the weights in the output layer in this way.

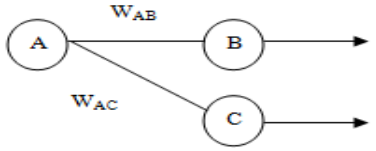

Figure **7** A single connection learning in a Back Propagation network.

iv. Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't calculate these directly (because we don't have a Target), so we *Back Propagate* them from the output layer. This is done by taking the Errors from the output neurons and running them back through the weights to get the hidden layer errors. For example, if neuron A is connected as shown to B and C then we take the errors from B and C to generate an error for A.

$Error_A = Output_A(1-Output_A)(Error_B\ W_{AB} + Error_C\ W_{AC})$

Again, the factor "*Output (1 - Output)*" is present because of the sigmoid squashing function.

v. Having obtained the Error for the hidden layer neurons now proceed as in step 3 to change the hidden layer weights. By repeating this method we can train a network of any number of layers.

As an example, all calculations for a full sized network of 2 inputs, 3 hidden layers and 2 outputs of neurons are shown in figure 8. $\mathbf{W}^+$ represents the new recalculated weight, whereas, W (without the superscript) represents the old weight.

i. Calculation of errors of output neurons

$$\delta_\alpha = out_\alpha\ (1 - out_\alpha)\ (Target_\alpha - out_\alpha)$$
$$\delta_\beta = out_\beta\ (1 - out_\beta)\ (Target_\beta - out_\beta)$$

ii. Changing output layer weights

$$W^+_{A\alpha} = W_{A\alpha} + \eta\ \delta_\alpha\ out_A$$
$$W^+_{A\beta} = W_{A\beta} + \eta\ \delta_\beta\ out_A$$

iii. Calculating (back-propagate) hidden layer errors

$$\delta_A = out_A\ (1 - out_A)\ (\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$$
$$\delta_B = out_B\ (1 - out_B)\ (\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$$
$$\delta_C = out_C\ (1 - out_C)\ (\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$$
$$W^+_{B\alpha} = W_{B\alpha} + \eta\ \delta_\alpha\ out_B$$
$$W^+_{B\beta} = W_{B\beta} + \eta\ \delta_\beta\ out_B$$
$$W^+_{C\alpha} = W_{C\alpha} + \eta\ \delta_\alpha\ out_C$$
$$W^+_{C\beta} = W_{C\beta} + \eta\ \delta_\beta\ out_C$$

iv. Changing hidden layer weights

$$W^+_{\lambda A} = W_{\lambda A} + \eta\delta_A\ in_\lambda$$
$$W^+_{\Omega A} = W^+_{\Omega A} + \eta\delta_A\ in_\Omega$$
$$W^+_{\lambda B} = W_{\lambda B} + \eta\delta_B\ in_\lambda$$
$$W^+_{\Omega B} = W^+_{\Omega B} + \eta\delta_B\ in_\Omega$$
$$W^+_{\lambda C} = W_{\lambda C} + \eta\delta_C\ in_\lambda$$
$$W^+_{\Omega C} = W^+_{\Omega C} + \eta\delta_C\ in_\Omega$$

The constant η (called the learning rate, and nominally equal to one) is put in to speed up or slow down the learning if required.

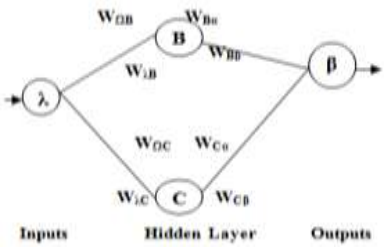

Figure 8 All the calculations for a reverse pass of Back Propagation.

V. Running the Algorithm

Now we see how the algorithm runs with a large data set. Suppose we wanted to teach a network to recognize the first four letters of the alphabet on a 5x7 grid, as shown in figure 9. The correct way to train the network is to apply the first letter and all the weights in the network once.

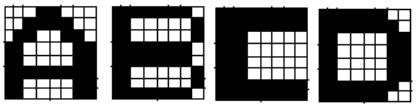

Figure 9 The first four letters of the alphabet change

Next, apply the second letter and do the same, then the third and so on. Algorithm is applied for all four letters, return to the first one again and repeat the process until the error becomes small [10]. Recognition means, a percent of recognition is achieved, but for better fitting the pattern, it is required to adjust η, the learning rate, and δ, the learning coefficient.

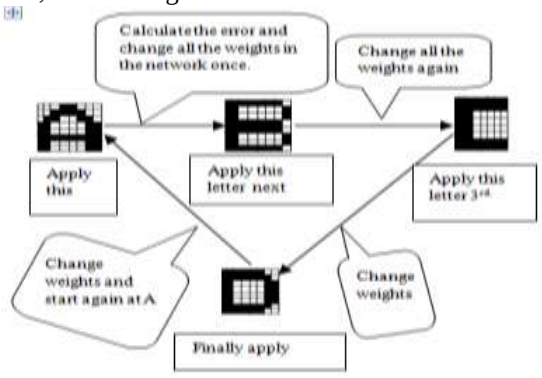

Figure 10 The correct flow of running the algorithm.

Figure 10 summarizes how the algorithm should work. One of the most common mistake is to apply the first letter to the network, run the algorithm and then repeat it until the error reduces to a desired level, then apply the second letter and do the same. If we did this, the network would learn to recognize the first letter, then forget it and learn the second letter and we did only end up with the last letter the network learned.

VI. Stopping Training

We could stop it once the network can recognize all the letters successfully, but in practice it is usual to let the error fall to a lower value first. This ensures that the letters are all being well recognized. We can evaluate the total error of the network by adding up all the errors for each individual neuron and then for each pattern in turn to give you a total error as shown in figure 1.11. In other words, the network keeps training all the patterns repeatedly until the total error falls to some pre-determined low target value and then it stops. When calculating the final error used to stop the network (which is the sum of all the individual neuron errors for each pattern) you need to make all errors positive so that they add up and do not subtract (an error of -0.5 is just as bad as an error of+0.5). Once the network has been trained, it should be able to recognize not just the perfect patterns, but also corrupted or noisy versions [11, 12].

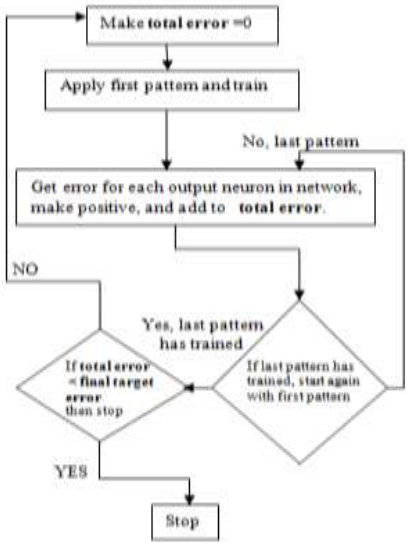

Figure 11 Total error for network.

In fact, if we deliberately add some noisy versions of the patterns into the training set as we train the network (say one in five), we can improve the network's performance in this respect. The training may also benefit from applying the patterns in a random order to the network. There is a better way of working out when to stop network training - which is to use a Validation Set. This stops the network overtraining (becoming too accurate, which can lessen its performance) [13, 14]. It does this by having a second set of patterns which are noisy versions of the training set (but aren't used for training themselves). Each time after the network has trained; this set (called the Validation Set) is used to calculate an error. When the error becomes low the network stops. Figure 12 shows the idea. When the network has fully trained, the Validation Set error reaches a minimum. When the network is overtraining (becoming too accurate) the validation set error starts rising. If the network over trains, it won't be able to handle noisy data so well.

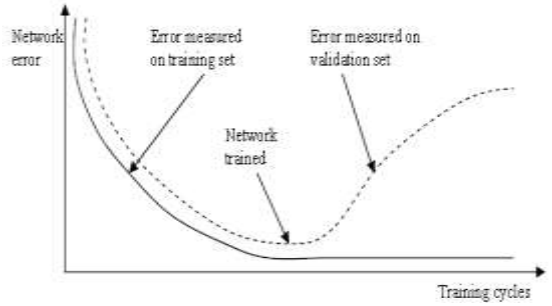

Figure 12 use of validation sets.

VII. Experimental Setup and Results

An interface is design with C sharp code using Back propagation algorithm, which is used to recognize a character or digit. In the following sections describes how this interface will work to give the output. This interface considers each 2D and 3D images in the form of object. So thinking each object belongs to a particular class can be expressed as:

Thinking Like Object: showing some code:

```
namespace BPSimplified
{
   //A Custom Panel To Draw Characters
   class DrawingPanel : Panel
   {
      private Bitmap BufferImage = null;
      private bool MouseMoving = false;
      private Graphics g_Image = null;
      private Graphics g_Panel = null;
      void DrawingPanel_Paint(object sender,
PaintEventArgs e){ }
      private void CreateNewImage(){ }
      protected override void Dispose(bool disposing){ }
      public Bitmap ImageOnPanel{ }
      public int PointSize{ }
   }
class ImageProcessing
   {
      //Convert RGB To Matrix [Of Double]
      public static double[] ToMatrix(Bitmap BM, int
MatrixRowNumber, int MatrixColumnNumber){  }
   }
      //Convert Double To Grey Level
      public static Bitmap ToImage(double[] Matrix, int
MatrixRowNumber, int MatrixColumnNumber,
                                    int ImageHeight, int
ImageWidth { }
}
```

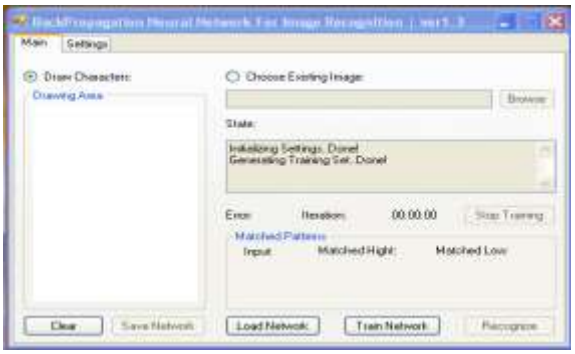

Figure 13 Initial State of the simulator.

A. Initial State

In this state we have to set all inputs we required, by hand drawing any object of image on the paint panel, the image is appeared on drawing area which is shown in Figure 13 and Figure 14.

B. Network Properties

In this experiment, we have initialized three layers of neurons, and experiments were performed over the parameters value of maximum error, number of input unit, number of hidden unit and number of output unit for running the network as shown in Figure 15.

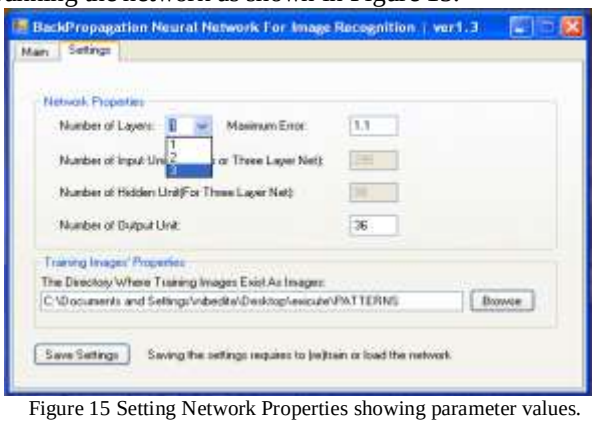

Figure 15 Setting Network Properties showing parameter values.

C. Train Network

After setting the network we have to train the network by clicking Train-Network button, which train our image and gives the Error value, number of iteration and time taken are shown in Figure 16**.** A snapshot of running our algorithm is shown in figure 16 indicating error= 1.088, iterations= 100 and simulation time= 03mili seconds which is found to train all the patterns. If it trains all the patterns, then it shows the message "Completed Training Process Successfully".

D. Recognizing Image

Once training process is successfully completed, pressing on 'Recognize' button to get the desired output as shown on Figure 17 description of other parameters used in simulation are:

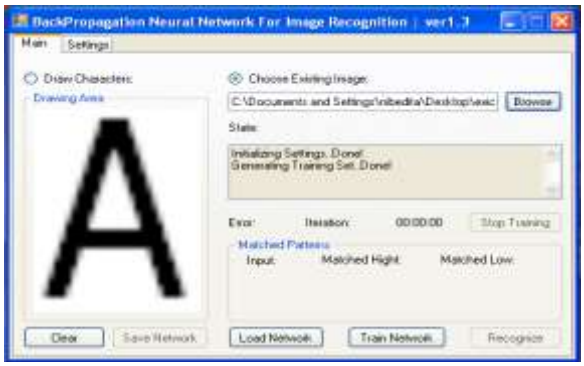

Figure 14 Selecting Image drawn within image pan.

(i) The 'Input' shows the Input supply by us.

(ii) The 'Height' shows first highest image match with respect to input

(iii) The 'Low' shows second highest image matches with respect to input considering the above steps we can also recognize a digit which shown in Figure 18. It shows recognition of the digit '9'. The height shows the maximum pattern matched and low shows the second maximum pattern matched for a set of supplied input.

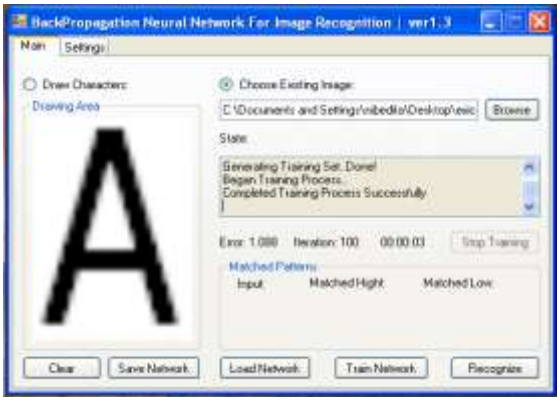

Figure 16 After Training the Network after 100 iteration.

VIII. CONCLUSION

Back propagation has some problems associated with it. Perhaps the best known is called 'Local Minima'. This occurs because the algorithm always changes the weights in such a way as to cause the error to fall. But the error might briefly have to rise as part of a more general fall, as shown in figure 12. If this is the case, the algorithms will get stuck (because it can't go uphill) and the error will not decrease further. There are several solutions to this problem. One is very simple and that is to reset the weights to different random numbers and try training again (this can also solve several other problems). Another solution is to add "momentum" to the change of weight. This means that the weight-change in a particular iteration depends not just on the current error, but also on previous changes of errors.

Yet, it is required to develop the algorithm with more hidden layers and our future works would target using recurrent network which has very strong recognition capacity in case of highly nonlinear irregular patterns.

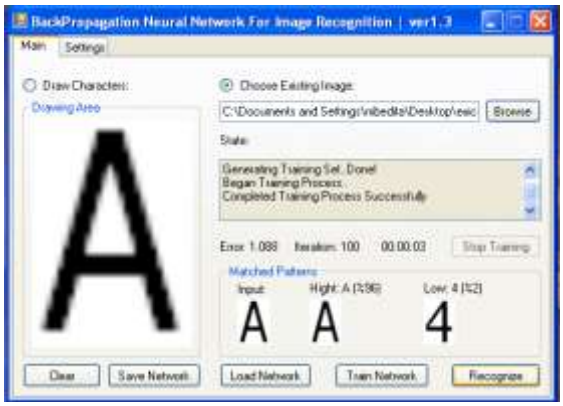

Figure 17 Shows the Final Output of a Letter 'A'.

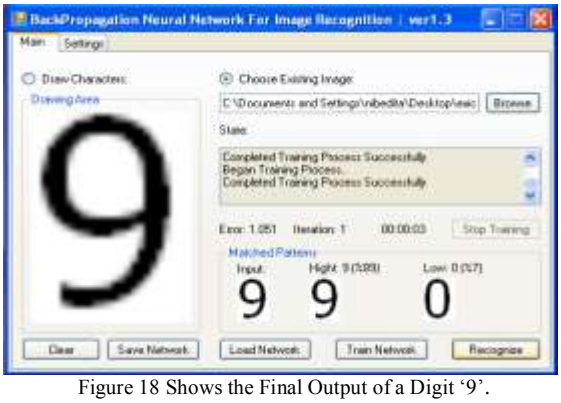

Figure 18 Shows the Final Output of a Digit '9'.

REFERENCES

[1] I B Levitan and L K Kaczmarek, The Neuron, cell and molecular biology, Oxford, 1997 (2nd ed). p45–146.

[2] D Hebb, Wiley, The Organisation of Behaviour, 1949.

[3] W McCulloch and W Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of mathematical biophysics, Vol 5, 1943. p115-137.

[4] W Pitts and W McCulloch, How we know universals, Bulletin of mathematical biophysics, Vol 9, 1947. p127-147.

[5] S Haykin, Neural Networks, Prentice-Hall, 1999 (2nd ed).

[6] F Rosenblatt, The Principles of Neurodynamics, Spartan books, 1961.

[7] M L Minsky and S A Papert, Perceptrons, MIT Press, 1989 (revised edition).

[8] A N Kolmogorov, Doklady Akademii SSSR, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition, Vol 144 p 679 - 681, 963. (American Mathematical Society translation – 28: 55 - 59).

[9] T Khanna, Foundations of neural networks, Addison Wesley, 1990.

[10] P J Werdos, Beyond regression: New tools for prediction and analysis in the behavioural sciences, PhD Thesis, Harvard University, 1974.

[11] D B Parker, Learning logic, Technical report TR-47, MIT, 1985.

[12] D E Rumelhart, Learning internal representations by error propagation, in Parallel Distributed Processing, Exploration in the microstructure of cognition Vol 1, MIT, 1986.

[13] P Frasconi et al, O. Omidvar (ed),Successes and failures of Backpropagation: A theoretical investigation, Progress in Neural Networks, Ablex Publishing, 1993. p205-242.

[14] P K Simpson, Artificial Neural Networks: Foundations, Paradigms, Applications and Implementations, Pergamon Press, 1990.