Contemporary Engineering Sciences, Vol. 4, 2011, no. 4, 149 - 163

Performance Evaluation of Artificial Neural Networks for Spatial Data Analysis

Akram A. Moustafa

Department of Computer Science
Al al-Bayt University, P.O. Box 922283
Mafraq, 11192, Jordan
hamarchi@yahoo.com

Abstract

In this paper the artificial neural network training algorithm is implemented in MATLAB language. This implementation is focused on the network parameters in order to get the optimal architecture of the network that means (the optimal neural network is the network that can reach the goals in minimum number of training iterations and minimum time of training). Many examples were tested and it was shown that using one hidden layer with number of neuron equal to the square of the number of inputs will lead to optimal neural network by mean of reducing the number of training stages (number of training iterations) and thus the processing time needed to train the network. Our goal in this paper is reaching the optimal solution by reaching minimum training time and minimum number of training iterations.

Keywords: Artificial neural network (ANN), Back-propagation, training rate and training iteration (epochs), hidden layer, net simulation, multilayer perceptron (MLP).

1. Introduction

Artificial Neural Network (ANN) is a Mathematical model designed to train, visualize, and validate neural network models [10], [12], [13] and the Artificial Neural Network (ANN) is a model-free estimator as it does not rely on an assumed form of the underlying data [2]. And we can define the neural network model is a data structure that can be adjusted to produce a mapping from a given set of input data to features of or relationships among the data. The model is adjusted, or trained, using a collection of data from a given source as input,

typically referred to as the training set. Note that the number of inputs to a network is set by the external specifications of the problem.To experiment with a two-input neuron; use the Neural Network Design Demonstration Two-Input Neuron (nnd2n2) [3].

After successful training, the neural network will be able to perform classification, estimation, prediction, or simulation on new data from the same or similar sources. The Neural Networks package supports different types of training or learning algorithms.

More specifically, the Neural Networks model uses numerical data to specify and evaluate artificial neural network models. Given a set of data, $\{x_i, y_i\}i^N = 1$, from an unknown function, y=f(x), this model uses numerical algorithms to derive reasonable estimates of the function, f(x). This involves three basic steps. First, a neural network structure is chosen that is considered suitable for the type of data and underlying process to be modeled. Second, the neural network is trained by using a sufficiently representative set of data. Third, the trained network is tested with different data, from the same or related sources, to validate that the mapping is of acceptable quality.

Choosing the proper type of neural network for a certain problem can be a critical issue. So it is very important to define the following in order to achieve a good model: The input sets, the target sets, the network architecture, the activation functions, the training function, the training rate, the goal, the number of iterations used to train the network.

Neural networks have arisen from analogies with models of the way that humans might approach pattern recognition tasks, although they have developed a long way from the biological roots. Great claims have been made for these procedures, and although few of these claims have withstood careful scrutiny, neural network methods have had great impact on pattern recognition practice [6].

2 Back-Propagation

There are many different types of ANN and techniques for training them but we are just going to focus on the most basic one all of them is classic back propagation neural network (BPN) [7].

Back propagation is by far the most widely used and understood neural network paradigm. Its popularity arises from its simple architecture and easy to understand learning process, the back propagation scheme consists of two major steps. These are the forward activation and the backward error flows.

The training process begins with the assignment of random weights to the connections between the nodes of the various layers. The various input patterns are then presented to the network, and the forward activation flow produces the output patterns. These output patterns will not be the same as the desired output patterns. The errors in the outputs are calculated for the output layer nodes as the

difference between the desired and actual outputs. For the hidden layers, the errors are calculated by back propagating the errors in the output layer to the hidden layers. The errors of each of the nodes are summed over the whole set of training patterns. These errors are used to change the weights in the interconnections between the layers. The weights connecting to the output layer are changed according to the delta rule, whereas for the weights in the hidden layers the generalized delta rule is used. There are many good references which describe the mathematics of the back propagation approach in detail including [17], [18].

The neural network is a computerized software program, which can be used to develop a non-linear statistical model in which there are many parameters, called weights. These weights control the output of the model given the input. The type of network used was a fully connected feed forward back propagation multi-layer perceptron (MLP) [15].

Multilayer perceptron & BP (Back-propagation) model Standard multilayer perceptron (MLP) architecture consists more than 2 layers; A MLP can have any number of layers, units per layer, network inputs, and network outputs [16].

In this paper, a Multi Layer Perceptron (MLP) network with a Back Propagation (BP) algorithm, a network with an optimum learning rate is proposed.

The MLP consists of neurons that are arranged in multiple layers with connections only between nodes

in the adjacent layers by weights. The layer where the optimum learning rate in BP Network input information is presented is known as the input layer and the layer where the processed information is retrieved is called the output layer. All layers between the input and output layers are known as hidden layers.

For all neurons in the network, except the input layer neurons, the total input of each neuron is the sum of the weighted outputs of the neurons in the previous layer. Each neuron is activated with input to the neuron and by the activation function of the neuron [18].

The input and output of the neuron, *i*, (except for the input layer) in a MLP mode, according to the BP algorithm [19], are:

$$\text{Input} \quad x_i = \sum w_{ij} o_j + b_i \tag{1}$$

$$\text{Output} \quad o_i = f(x_i) \tag{2}$$

where *Wij i*s the weight of the connection from neuron *i* to node *j*, *bi* is the numerical value and *f* is the activation function.

The sum in Equation 1 is over all neurons, *j*, in the previous layer. The output function is a nonlinear function, which allows a network to solve problems that a linear network can't [20]. In this study, the tan-sigmoid and linear transfer functions are used to determine the output.

A Back-Propagation (BP) algorithm is designed to reduce error between the actual output and the desired output of the network in a gradient descent manner.

Finally the back propagation algorithm for training neural networks has been discussed in many papers [1]. The back propagation refers to the fact that any mistakes made by the network during training get sent backwards through it in an attempt to correct it and so teach the network what right and wrong [8].

The BPN learns during a training epoch, you will probably go through several epochs before the network has sufficiently learnt to handle all the data you have provided it and the end result is satisfactory. A training epoch is described below [7],[14]:

For each input entry in the training data set:

- Feed input data in (feed forward).
- Check output against desired value and feed back error (back-propagate).

Where back-propagation consists of:

Calculate error gradients.

- Update weights.

Thus we can summarize the training Back-propagation algorithm into the following steps:

Step 1: Input training vector.
Step 2: Hidden nodes calculate their outputs.
Step 3: Output nodes calculate their outputs on the basis of Step 2.
Step 4: Calculate the differences between the results of Step 3 and targets.
Step 5: Apply the first part of the training rule using the results of Step 4.
Step 6: For each hidden node, n, calculate d(n).
Step 7: Apply the second part of the training rule using the results of Step 6.
Steps 1 through 3 are often called the forward pass, and steps 4 through 7 are often called the *backward pass*. Hence, the name: back-propagation.

An actual algorithm for a 3-layer network (only one hidden layer) [9] – [13]:
Initialize the weights in the network (often randomly)

Do
For each example e in the training set
O = neural-net-output (network, e); forward pass
T = teacher output for e
Calculate error (T - O) at the output units
Compute delta_wh for all weights from hidden layer to output layer; backward pass
Compute delta_wi for all weights from input layer to hidden layer; backward pass continued
Update the weights in the network

Until all examples classified correctly or stopping criterion satisfied
Return the network

3 Experimental Results

We used MATLAB because it is already in use in many institutions, And it is useful for statistical task to see the demeanor of the system, It is used in research in academia and industry, MATLAB is useful for data analysis, data extraction and data processing, we can perform operations from MATLAB help command,. It is used to generate stimulus for verification of the system. Prototype solutions are usually obtained faster in MATLAB than solving a problem by using other programming languages [11].

The artificial neural network back propagation algorithm is implemented in Matlab language. This implementation is compared with several other software packages.

The effect of reducing the number of iterations in the performance of the algorithm is studied. The speed of the back propagation program, written in Matlab language is compared with the speed of several other back propagation programs which are written in the C language. The speed of the Matlab program is**,** also compared with the C program which is a variant of the back propagation algorithm, As the next test shows, MATLAB was about 3 times faster than a C++ programming both doing a matrix multiply [10].

A program in C++ was written to multiply two matrices containing double precision numbers. The result of the multiplication is assigned into a third matrix. Each matrix contained 1000 rows and 1000 columns. A MATLAB M file was written to do the same multiply as C++ program did. Only the segment of the code which does the multiplication is timed. The test was run on an IBM PC P4 computer, the result is shown in Table 1

Table 1
Speed Comparison of Matrix Multiply In Matlab and a C++ Program

Tools	**Execution time in seconds 1000 X 1000 multiply**
C++	3.16652
MATLAB	1.204000

As appears from the above table that the MATLAB runs 2.63 times faster than the C++ program.

We can summarize the methodology of the experiment: Training, Testing and Validation Datasets into:

- In the ANN methodology, the sample data is often subdivided into *training*, *validation*, and *test* sets [14].
- The distinctions among these subsets are crucial.
- defines the following :
  - *Training set:* A set of examples used for learning that is to fit the parameters [weights] of the classifier [14].
  - *Validation set:* A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network.
  - *Test set:* A set of examples used only to assess the performance [generalization] of a fully-specified classifier [14].
- A popular Artificial neural network architectures chosen are multilayer perceptrons using the backpropagation
algorithm [5].

- In overview, a MLP is composed of layers of processing units that are interconnected through weighted connections:
  - The first layer consists of the input vector.
  - The last layer consists of the output vector representing the output class.
  - Intermediate layers called `hidden` layers receive the entire input pattern that is modified by the passage through the weighted connections. The hidden layer provides the internal representation of neural pathways.
- The network is trained using back propagation with three major phases.
  - *First phase*: an input vector is presented to the network which leads via the forward pass to the activation of the network as a whole. This generates a difference (error) between the output of the network and the desired output.
  - *Second phase:* compute the error factor (signal) for the output unit and propagates this factor successively back through the network (error backward pass).
  - *Third phase:* compute the changes for the connection weights by feeding the summed squared errors from the output layer back through the hidden layers to the input layer.
- Continue this process until the connection weights in the network have been adjusted so that the network output has converged, to an acceptable level, with the desired output.
- Assign "unseen" or new data:
  - The trained network is then given the new data and processing and flow of information through the activated network should lead to the assignment of the input data to the output class.
- For the basic equations relevant to the back propagation model based on generalized delta rule, the training algorithm was known [11].

We have to note the following things while using the back propagation algorithm:

- *Learning rate:*
  - Standard back propagation can be used for incremental (on-line) training (in which the weights are updated after processing each case) but it does not converge to a stationary point of the error surface. To obtain convergence, the learning rate must be slowly reduced. This methodology is called "stochastic approximation."
  - In standard back propagation, too low a learning rate makes the network learn very slowly. Too high a learning rate makes the weights and error function diverge, so there is no learning at all.
  - Trying to train a NN using a constant learning rate is usually a tedious process requiring much trial and error. There are many variations proposed to improve the standard back propagation as well as other learning algorithms that don't suffer from these limitations [11].
- *Output Representation:*
- *Input Data:*
  - Normalize or transform the data into [0,1] range. This can help for various reasons.
- *Number of Hidden Units:*
  - Simply try many networks with different numbers of hidden units, estimate the generalization error for each one, and choose the network with the minimum estimated generalization error.
- *Activation functions*:
  - For the hidden units, are needed to introduce nonlinearity into the network.
  - Without nonlinearity, hidden units would not make nets more powerful than just plain perceptrons (which do not have any hidden units, just input and output units).
  - The sigmoidal functions such as logistic and tanh and the Gaussian function are the most common choices [4].

To get some results we choose the following examples:

Example 1:

Let us take an XOR operation with 2 inputs and 1 output and study the implementation of the neural model for this operation [9,12]:

Step 1: Generate the training data.

```
p= [0 0 1 1; 0 1 0 1]; % input for training
```

Step 2: Generate the target output.

```
t= [0 1 1 0]; % output for training
```

Step 3: Create the neural network and define the activation functions and training function.

```
net = newff([0 1;0 1],[4 1],{'logsig' 'purelin'});
```

Step 4: Initialize the network.

```
net = init(net);
```

Step 5: Define the parameters of the network.

```
net.trainParam.epochs = 10000;
% Maximum epochs are 10000 (iteration number of optimization)
net.trainParam.goal = 0.000001; % Error limit is 0.000001
%-------------------------------------------------
% training the MLP by using a training pattern
%-------------------------------------------------
net.trainParam.lr = 0.05;
```

Step 6: Train the network.

```
[net tr] = train(net,p,t);
% training the MLP neural network by using a train function
%-------------------------------------------------
% to draw the result making the test pattern
%-------------------------------------------------
```

Step 7: Find the output of the network.

```
Output=sim(net,p);
```

The previous steps were implemented using different network architecture and the optimal performance was achieved using 1 hidden layer with 4 neurons and 1 output layer with 1 neuron as shown in figure 1.

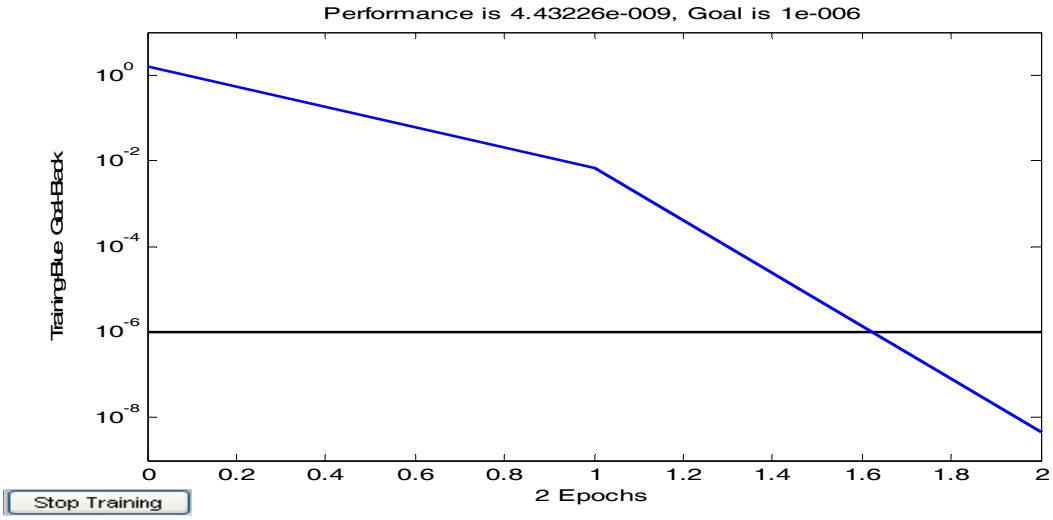

Fig. 1 Optimal solution to reach the goal for example1

The results of implementation the network using different network architecture are shown in Table 2.

Table 2
Simulation Resells for Example 1

Number of neurons in the hidden layer	Average training time in seconds	Number of training iterations	The goal was met
1	5.985	592	No
2	1.407	198	Yes
3	0.456	3	Yes
4	**0.152**	**2**	**Yes**
5	0.172	2	Yes
6	0.156	2	Yes
7	0.157	3	Yes
2X2(2 hidden layers each with 2 neurons)	0.578	51	Yes

Example 2:

Let us now take an XOR operation with 3 inputs and 1 output and study the implementation of the neural model for this operation:

Step 1: Generate the training data.

```
p=[0 0 0 0 1 1 1 1;0 0 1 1 0 0 1 1;0 1 0 1 0 1 0 1]; % input for training
```

Step 2:: Generate the target output.

```
t=[0 1 1 0 1 0 0 1]; % output for training
```

Step 3: Create the neural network and define the activation functions and training function.

```
net = newff([0 1;0 1;0 1],[9 1],{'logsig' 'purelin'});
```

Step 4: Initialize the network.

```
net = init(net);
```

Step 5: Define the parameters of the network.

```
net.trainParam.epochs = 10000;
% Maximum epochs is 10000 ( iteration number of optimization )
net.trainParam.goal = 0.000001; % Error limit is 0.000001
%-------------------------------------------------
% training the MLP by using a training pattern
%-------------------------------------------------
net.trainParam.lr = 0.05;
```

Step 6: Train the network.

```
[net tr] = train(net,p,t);
% training the MLP neural network by using a train function
%-------------------------------------------------
% To draw the result making the test pattern
%-------------------------------------------------
```

Step 7: Find the output of the network.

```
Output=sim(net,p);
```

The previous steps were implemented using different network architecture and the optimal performance was achieved using 1 hidden layer with 9 neurons and 1 output layer with 1 neuron as shown in figure 2.

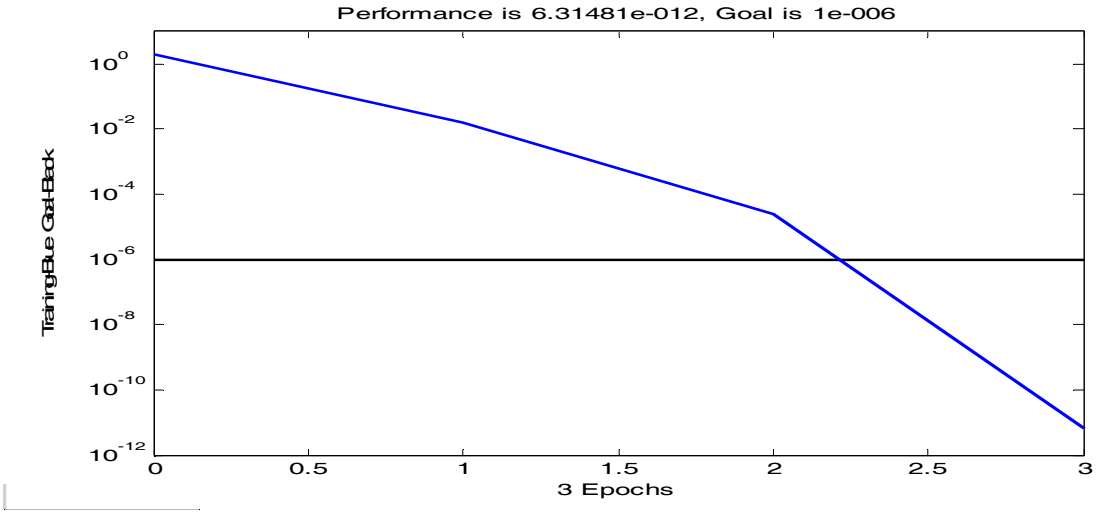

Fig. 2 Optimal solution to reach the goal for example 2

The results of implementation the network using different network architecture are shown in Table 3.

Table 3
Simulation resells for example 2.

Number of neurons in the hidden layer	Average training time in seconds	Number of training iterations	The goal was met
1	1.687	233	No
2	1.297	167	No
3	4.344	636	Yes
4	0.437	36	Yes
5	0.328	27	Yes
6	0.187	7	Yes
7	0.172	5	Yes
8	0.156	4	Yes
9	**0.141**	**3**	**Yes**
10	0.172	3	Yes
11	0.157	3	Yes
12	0.187	4	Yes
2 hidden layers(6,3)	0.188	8	Yes
2 hidden layers(3,6)	0.219	11	Yes

Example 3:

Let us now take an XOR operation with 4 inputs and 1 output and study the implementation of the neural model for this operation:

Step 1: Generate the training data.

```
p=[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1;
  0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1;
  0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1;
  0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]; % input for training
```

Step 2:: Generate the target output

```
t=[0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0]; % output for training
```

Step 3: Create the neural network and define the activation functions and training function

```
net = newff([0 1;0 1;0 1;0 1],[16 1],{'logsig' 'purelin'});
```

Step 4: Initialize the network

```
net = init(net);
```

Step 5: Define the parameters of the network

```
net.trainParam.epochs = 10000;
```

```
% Maximum epochs is 10000 ( iteration number of optimization )
net.trainParam.goal = 0.000001; % Error limit is 0.000001
%-------------------------------------------------
% training the MLP by using a training pattern
%-------------------------------------------------
net.trainParam.lr = 0.05;
```

Step 6: Train the network

```
 [net tr] = train(net,p,t);
% training the MLP neural network by using a train function
%-------------------------------------------------
% To draw the result making the test pattern
%-------------------------------------------------
```

Step 7: Find the output of the network

```
Output=sim(net,p);
```

The previous steps were implemented using different network architecture and the optimal performance was achieved using 1 hidden layer with 9 neurons and 1 output layer with 1 neuron as shown in figure 3.

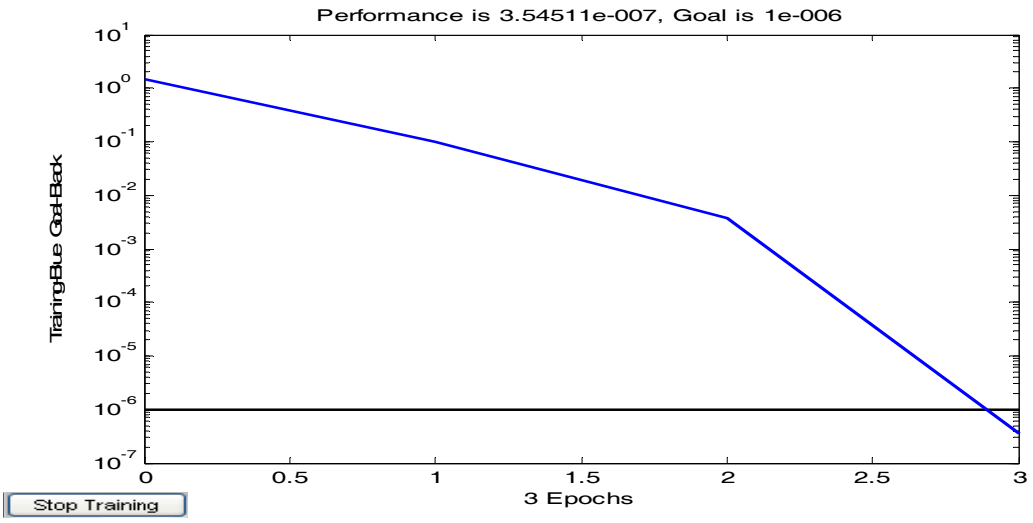

Fig. 3 Optimal solution to reach the goal for example 3

The results of implementation the network using different network architecture are shown in Table 4.

Table 4

Simulation Resells for Example 3.

Number of neurons in the hidden layer	Average training time in seconds	Number of training iterations	The goal was met
1	3.937	511	No
2	223.672	9784	No
3	2.203	334	No
4	61.344	4839	No
5	1.64	240	Yes
6	0.594	75	Yes
7	1.5	200	Yes
8	0.172	12	Yes
9	0.141	7	Yes
10	0.141	7	Yes
11	0.140	6	Yes
12	0.140	6	Yes
13	0.140	4	Yes
14	0.145	5	Yes
15	0.140	4	Yes
16	0.125	3	Yes
17	0.156	4	Yes
18	0.142	3	Yes
19	0.145	4	Yes
20	0.157	3	Yes
2 hidden layers(8X8)	0.147	3	Yes

4 Results Discussions

The optimal neural network is the network that can reach the goals in minimum number of training iterations and minimum time of training. From the experimental results shown in the Tables 2, 3, and 4 we can see that we can reach the goals by adding a hidden layer to the network, and an optimized solution can be achieved if the number of neurons is equal to the power of 2 of the number of inputs. Splitting the number of neurons in the hidden layer to 2 or more hidden layers make the network not optimal by mean of increasing the number of training iterations and increasing the training time. During the experiments implementations it was seen that changing the training rate around 0.05 does not much affect the network performance.

5 Conclusions

This paper proves that the efficiency of Artificial Neural Networks to attain the optimal Neural network and optimal solution. That means we can attaining the goals in minimum training time and minimum number of training iterations and that are depends on the network architecture.
Using one hidden layer with number of neurons equal to the power of 2 of the number of inputs leads to optimal and efficient Artificial Neural Networks.

References

[1] P.J. Werbos, “Backpropagation through Time: What It Does and How to Do it" Proceedings of the IEEE, 78_10_, pp.1550-1560, 1990.
[2] D. H. Chang and S. Islam, "Estimation of Soil Physical Properties Using Remote Sensing and Artificial Neural Network," Remote Sensing of Environment, vol. 74, pp. 534- 544, 2000.
[3] M.Hagan, H. Demuth, M. Beele, Neural Network Design, University of Colorado Bookstore, 2002, ISBN: 0- 9717321- 0-8.
[4] Sucharita Gopal. Artificial Neural Networks for Spatial Data Analysis, NCGIA Core Curriculum in GIScience, 1998.
[5] Benediktsson, J. A.; Swain, P. H. and Ersoy, O. K., 1990, “Neural network approaches versus statistical methods in classification of multisource remote sensing data”. IEEE Trans. on Geoscience and Remote *Sensing*, GE-28, Pp. 540-552.

[6] Ripley, B.D., Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press, ISBN 0-521-46086-7 (hardback), (1996), pp 354 and pp 403.
[7] Bishop, C.M. Neural Networks for Pattern Recognition, Oxford: Oxford University Press. ISBN 0-19-853849-9 (hardback) or 0-19-853864-2, (1995), pp.482.
[8] W. Chang, B. Bosworth, G.C. Carter, On using back propagation neural networks to separate single echoes from multiple echoes, Acoustics, Speech, and Signal Processing, ICASSP, IEEE International Conference on, (27-30 Apr, 1993), vol. 1, pp.265-268.
[9] Kwan, H.K., simple sigmoid. Like activation function suitable for digital hardware implementation, Electronic Letters, vol.28, no.15, (1992), pp.1379 – 1380.
[10] Jamshid nazari and okan k.ersoy, implementation of back propagation neural networks with Matlab, school of electrical engineering purdue university1992.
[11] The MathWorks Inc. PRO-MATLAB for Sun Workstations, User's Guide, January 1990.
[12] Paul T. Baffes. NETS Users's Guide, Version 2.0 of NETS. Technical Report JSC-23366, NASA, Software Technology Branch, Lyndon B. Johnson Space Center, September 1989.
[13] E. Fahlman, Scott. An Empirical Study of Learning Speed in Back Propagation Networks. Technical Report CMU-CS-88-162, CMU, CMU, September 1988.
[14] D. C. Silverman, Discussion of The Corrosion of Nickel 270 In Phosphate Containing Solutions, Corrosion, vol. 41, no. 4, (1985), pp. 244.
[15] Patterson DW. Artificial Neural Networks, Theory and Applications. Singapore: Prentice Hall, 1996 pp.141–179.
[16] Insung Jung, and Gi-Nam Wang, "Pattern Classification of Back-Propagation Algorithm Using Exclusive Connecting Network, World Academy of Science, Engineering and Technology, vol 36, 2007, pp189-193.
[17] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network", IJCNN International Conference on Neural Networks, 1, 1989, pp.593-606.
[18] D. E. Rumelhart, J. L.McClell and the PDP Research Group institute for Cognitive Science, University of California, SanDiego, and Parallel Distributed Processing: Explorations in the Microstructure of Cognition, (MIT Press, Cambridge, Mass. 1986).
[19] Pao, Y.H. Adaptive Pattern Recognition and Neural Network, Addison-Wesley Publishing Company, Inc. 1989.
[20] Heermann, P. and Khazenie, N. Classification of Multi spectral remote sensing data using a backpropagation neural network", IEEE Trans. on Geo science and Remote Sensing, 30, pp. 81-88, 1992.