Classification using Artificial Neural Networks - A study

By
Mayur Mudigonda

1. Abstract

Inspired by biological models, researchers in a variety of fields are applying neural network models to solve problems that so far have not been solvable using Von Neumann's architecture. The aim of the project is to perform a comparative study of some of the existing Neural Network models available for Classification. For the purpose of comparing a standard problem of "Classifying Iris Flowers" is used.

2. Neural Networks an Introduction

Artificial Neural Networks, which are also referred to as neural computation, network computation, connectionist models and parallel distributed processing (PDP), are massively parallel computing systems consisting of an extremely large number of processors with many inter-connections between them. ANNs were designed with the goal of building "intelligent machines" to solve complex problems, such as pattern recognition and classification. It also allows us an insight to the workings of the human brain, thus enabling us to test various hypotheses about the brain.

Thus *ANN* is an information processing system that has certain performance characteristics in common with biological neural networks. *ANN* have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at simple elements called *neurons*
2. Signals are passed between neurons over connection links
3. Each connection link has an associated weight, which in a typical neural net multiplies the signal transmitted
4. Each neuron applies an activation function (usually non-linear) to its net input (sum of weighted input signals) to determine its output signal

A neural network is characterized by the following:

1. Its pattern of connections between the neurons (called its *architecture*)
2. Its method of determining the weights on the connections (called its *training, or learning, algorithm*)

3. Its *activation function*

Architecture:

A neural net consists of a large number of simple processing elements called neurons, units, cells or nodes. Each neuron is connected to other neurons in links which have a weight associated with them. The weights represent the information being used by the network to solve the problem. Thus based on the problem we are trying to solve the interconnections and the weights associated vary. The initial assignment of weights is a topic of important discussion, as the closer the initial weights are to the final weights the faster the network is trained and ready to use.

Training and Activation:

Each neuron has an internal state called the *activation level*, which is a function of the inputs it has received. Typically a neuron sends its activation as a signal to several neurons in the next layer. The neurons can send only one signal but the signal may be *broadcasted*. The *activation function* of a neuron Y is given by some function of its net input, for eg the Log-Sigmod function or any of the other activation functions.

The presence of a hidden layer often enables the neural network to solve more problems than those solved by a net with simple inputs and outputs. As an aside such networks are harder to train.

3. Biological Inspiration

The *neuron* -the atomic element, the essence of life, is the fundamental unit of computation for the human brain[1]. A schematic diagram of the neuron is shown in Fig 1. A neuron consists of a cell body or soma and two types of out-reaching tree-like branches: *axon* and *dendrites*. The cell body has a nucleus that contains the hereditary information and plasma containing molecular equipment for the production of material needed by the neuron. A neuron receives signals from other neurons through its dendrites, and transmits signals generated by its cell body along the axon(transmitter) which eventually branches into strands and sub strands. At the terminals of these strands are the *synapses*. A synapse is a place of contact between two neurons (between an axon and dendrite of two neurons). When the impulse reaches the synapse's terminal, certain chemicals, called neurotransmitters are released. The neurotransmitters diffuse across the gap and their role is to either inhibit or enhance, depending on the type of the synapse, the receptor neurons own tendency to emit electrical impulses. The effectiveness

[1] The introduction of neurons as fundamental constituents of the nervous system was credited to Ramon y Cajal who won the 1906 Nobel Prize for physiology and medicine.

of a synapse can be adjusted by the signals passing through it so that synapses can *learn* from the activities which they take part in.

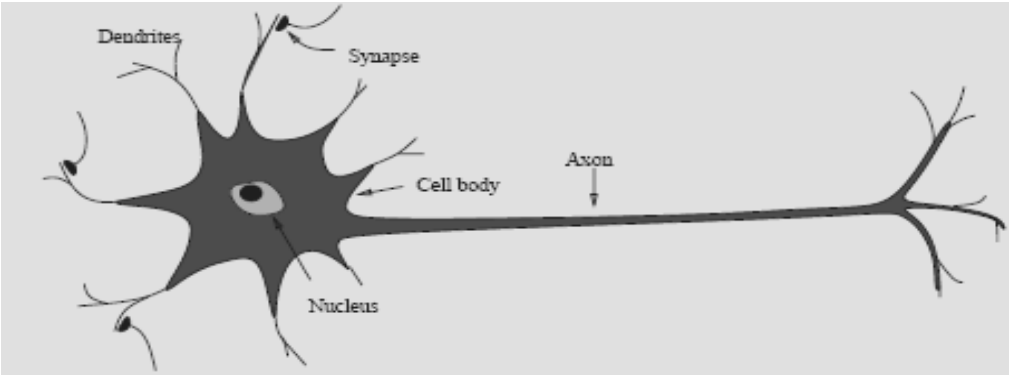


Fig 1. A neuron

4. Neural Network Architectures

An assembly of neurons is called a neural network. ANNs can be viewed as weighted graphs in which nodes are neurons and directed edges are connections from output neurons to input ones. Shown in Fig 3- taxonomy of neural network architectures or topologies.

The first ever proposed Neural Network model was the McCulloch and Pitt's model. They proposed a binary threshold unit as a computational model for a neuron. A schematic representation of the model is shown in Fig 2.

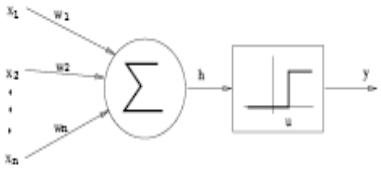


Fig 2- A McCulloch and Pitt's model

The mathematical model computed a weighted sum of its input. If the output was above a threshold value then it declared the output to be "1" otherwise the output was "0".

$$Y = \theta \left(\sum w_j x_j - \mu \right)$$

Where θ is the unit step function and w_j is the synapse weight associated with the jth input. Positive weights correspond to *excitatory* synapses and negative weights correspond to *inhibitory* synapses.

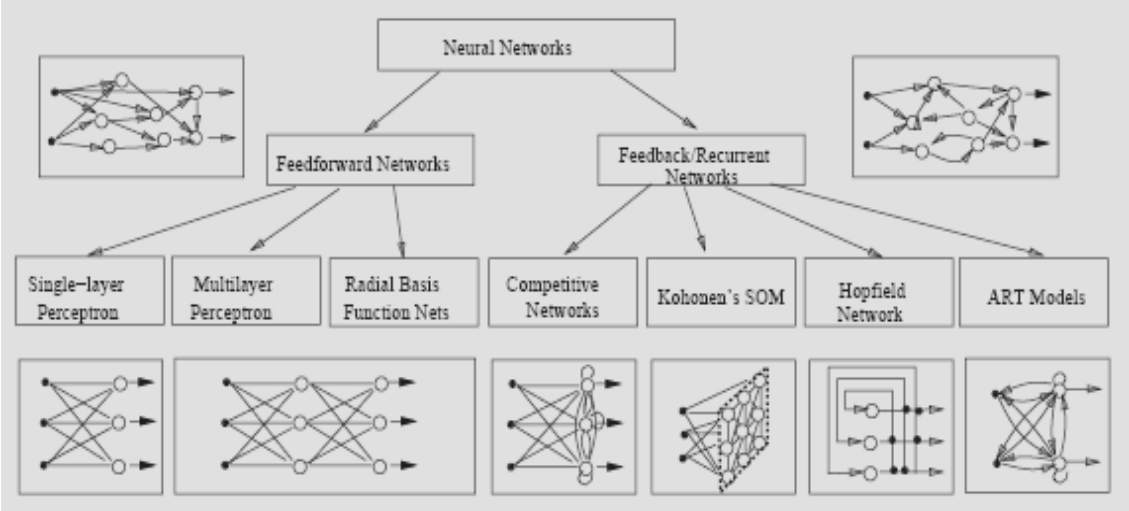
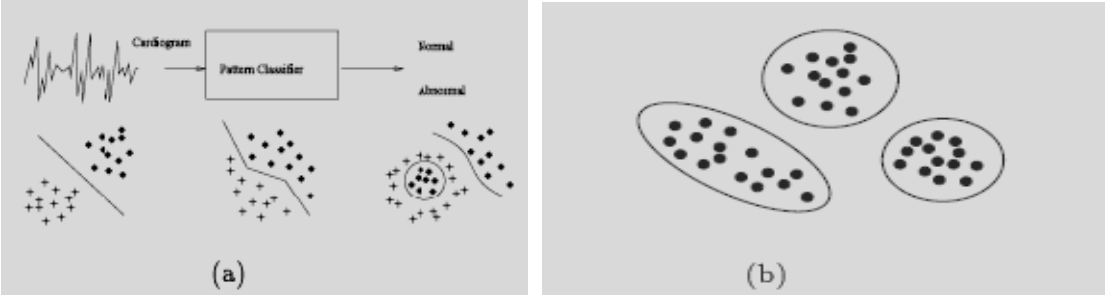


Fig 3. Taxonomy of neural network architectures

5. Classification and Neural Networks

Classification is the process by which we separate or segregate each input vector to a particular group or category. Neural Networks have played an increasingly significant role in the process of classification. Here a few of the architectures used for classification are explained.



6. HEBB NET

The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebb proposed that learning occurs by modification of the synapse strengths in a manner such that if two interconnected neurons are both "on" at the same time, then the weight between those neurons should be increased. A stronger learning occurs if we also increase the weights if both neurons are "off".

Algorithm

1. *Initialize all the weights wi=0 (i=1 to n)*
2. *For each input training vector and target output pairs, s:t, do steps 3-5*
3. *Set activation for input units xi=si (i=1 to n)*
4. *Set activation for output units y=t*

5. *Adjust the weights for*
   Wi(new) = wi(Old) + xi y
   Adjust the bias
   b(new) = b(old) + y

Hebb Nets are used to classify 2-d input patterns very easily. Thus it is used for character recognition and other such applications. Hebb Nets cannot learn any pattern for which the target is 0. So targets must be converted at least to a bipolar form so to say. The Hebb faces difficulties classifying certain problems if bipolar patterns are employed.

7. PERCEPTRON

Perceptrons had perhaps the most far-reaching impact of any early neural nets. The perceptron learning rule is more powerful than the Hebb rule. Under suitable conditions its iterative learning algorithm can be shown to converge to the correct weights. A number of different types of perceptrons are described in Rosenblatt(1962). Although some were self-organizing maps, most were trained. Typically, the original perceptrons had three layers of neurons- sensory units, associator units and a response unit- forming an approximate model of a *retina*. One particular simple perceptron is as described. It consisted of binary activations for the sensory and associator units. The response unit employed the use of +1,0 and –1.

The output of a perceptron is y = *f(y_in)*, where the activation function is

$$f(y\_in) = \begin{cases} 1 & y\_in > \theta \\ 0 & -\theta <= y\_in <= \theta \\ -1 & y\_in < -\theta \end{cases}$$

The net did not distinguish between an error in which the calculated output was zero and the target –1, as opposed to an error in which the calculated output was +1 and the target –1. In either of these cases, the sign of the error denotes that the weights should be changed in the direction indicated by the target value. However, only the weights on the connections from units that sent a non-zero signal to the output unit would be adjusted (since only these signals contributed to the error). If an error occurred for a particular training input pattern, the weights be changed according to the formula

$$Wi(new) = Wi(old) + \alpha t Xi$$

Where the target value t is +1 or –1 and α is the learning rate. If an error did not occur, the weights would not be changed. Training would continue until no error occurred. The perceptron learning rule convergence theorem states that if weights exists to allow the net to respond correctly to all the training patterns,

then the rule's procedure for adjusting the weights will find values such that the net respond correctly to all training patterns.

Algorithm:

1. *Initialize the weights and bias*
   (For learning, set weights and bias to zero)
   Set learning rate α ($0 <= \alpha <= 1$)
   (For simplicity, α can be set to 1)

2. *While stopping condition is false, do steps 3-7*

3. *For each training pair s:t, do steps 4-6*

4. *Set activations of input units:*
   $Xi = Si$
5. *Compute Response of output unit:*
   $y\_in = b + \sum Xi\ Wi$;

$$f(y\_in) = \begin{cases} 1 & y\_in > \theta \\ 0 & -\theta <= y\_in <= \theta \\ -1 & y\_in < -\theta \end{cases}$$

6. *Update weights and bias if an error occurred for this pattern*
   If y != t

   $Wi(new) = Wi(old) + \alpha t Xi$
   $B(new) = B(old) + \alpha t$

   Else

   $Wi(new) = W(old)$
   $B(new) = B(old)$

7. *Test Stop Conditon*

The limitation of the *Perceptron Convergence Rule* is that it is applicable only to linearly separable problems. But on employing *Gaussian Functions* the perceptron may solve non-linear problems.

8. ADALINE Learning Algorithm

Widrow and Hoff proposed the ADALINE or Adaptive Linear Neuron, 1960. It typically uses bipolar (1 or –1) activations for its input signals and its target output (although it is not restricted to such values). The weights on the connections from the input units to the ADALINE are adjustable. The ADALINE also has a bias, which acts like an adjustable weight on a connection from a unit whose activation is always 1.

In general, an ADALINE can be trained using the delta rule, also known as the Least Mean Squares or Widrow-Hoff rule. The rule can also be used for single layer nets with several output units. The ADALINE is a special case in which there is only one output unit. During training, the activation of the unit is its net input, i.e., the activation function is the identity function. The learning rule minimizes the mean squared error between the activation and the target value. This allows the net to continue learning on all training patterns; even after the correct output value is generated for some patterns.

When ADALINE's are used for pattern classification where the output is either a +1 or –1, a threshold function is applied to the input to obtain the activation. If the net input to the ADALINE is greater than or equal to 0, then its activation is set to 1; otherwise it is set to –1. Any problem for which the input patterns corresponding to the output value +1 are linearly separable from input patterns corresponding to the output value –1, an ADALINE unit can model the problem successfully.

Algorithm:

1. Initialize the weights and bias
(Small random values are usually used)
Set learning rate α ($0 <= \alpha <= 1$)
(See comments following algorithm)

2. While stopping condition is false, do steps 3-7

3. For each bipolar training pair s:t do steps 4-6

4. Set activations of input units, i=1...n
$Xi = Si,$
5. Compute net input to output unit:
$y\_in = b + \sum Xi\ Wi$
6. Update bias and weights, i=1...n
$B(new) = B(old) + \alpha(t - y\_in),$
$Wi(new) = Wi(old) + \alpha(t - y\_in)Xi$

7. Test for stopping condition

Setting the learning rate to a suitable value requires some care. According to Hecht-Nielsen (1990), an upper bound for its value can be found from the eigen value of the correlation matrix R of the input (row) vectors x(p):

$R = (1/P) * (\sum X(p)^T X(p)) \quad For \sum, p = 1...P$

$\alpha <$ one-half the largest eigen value of R
Usually α is taken to be a value such as 0.1 initially. The proof of the ADALINE training process is contained in the derivation of the *delta rule*.

9. MADALINE Learning Algorithm

As mentioned earlier, a MADALINE consists of Many Adaptive Linear Neurons arranged in a multi layer net. The problems solvable by the perceptron and the derivation rule for several output units both indicate there is essentially no change in the process of training if several ADALINE units are combined in a single-layered net. In this section we will discuss the MADALINE.

The MRII training algorithm of the MADALINE

Algorithm:

1. Initialize weights
Set the learning rate α
While Stopping condition is false, do steps 2-8
2. For each bipolar training pair, s:t, do steps 3-7
3. Set the input units
Xi = Si
4. Compute net input to each hidden ADALINE unit
z_in1 = b1 + x1 w11 + x2 w21,
z_in2 = b2 + x1 w12 + x2 w22
.
.
.
5. Determine the output of each hidden ADALINE unit
z1=f(z_in1)
z2=f(z_in2)
.
.
.
6. Determine the output of net:
y_in = b3 + z1v1 + z2v2
y = f(y_in)

7. Determine error and update weightsl

If t=y, no weight updates are performed.
Otherwise:
If t != y, do steps 7 (a) & (b) for each hidden unit whose net input is sufficiently close to 0. Start with the unit closest to 0 then the next one...
7(a) Change the unit's output
7(b) Re compute the response of the net
If the error is reduced:
Adjust the weights on this unit
8. Test stopping condition

10. BACK PROPOGATION Algorithm

The demonstration of the limitations of single layer neural network was a single significant factor in the decline in interest in neural networks in the 1970's. The discovery (by several researchers independently) and widespread dissemination f an effective general method of training a multi layer neural network[Rumelhart, Hinton & Williams, 1986a, 1986b].

The very general nature of the back propagation training method means that a back propagation net (a multi layer, feed forward net trained by back propagation) can be used to solve problems in many areas.

The training of a network by back propagation involves three stages: the feed forward of the input training pattern, the calculation and back propagation of the error and the adjustment of weights. Although a single layer net can learn it is severely limited in its mapping. A multi layered net can be mapped to solve any problem up to any arbitrary accuracy.

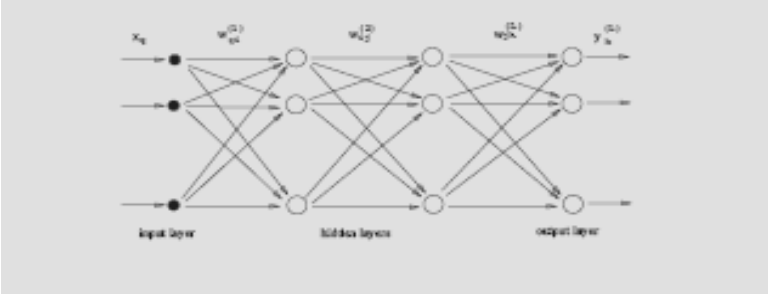


Fig 4 A Multi layer perceptron with back propagation training

Algorithm:

1. *Initialize weights*
2. *While Stopping condition if false, do steps 3-10*
3. *For each training pair, do steps 4-9*
   Feed Forward
4. *Each Input unit (Xi ,i=1...n) receives input signal xi and broadcasts this signal to all units in the hidden layer above it*

5. *Each hidden unit (Zj, j=1...p) sums its weighted input signals*
   $Z\_in\ j = v0j + \sum xi\ vij$,
   Applies its activation function to compute its output signal
   $Zj = f(z\_in\ j)$
   And sends this signal to all units in the layers above
6. *Each output unit (Yk, k=1...m) sums its weighted input signals,*
   $Y\_in\ k = w0k + \sum zj\ wjk$
   Applies its activation function to compute its output signal
   $Yk = f(y\_in\ k)$

BACK PROPAGATION OF ERROR

7. *Each output unit (Yk, k=1...m) receives a target pattern corresponding to the input training pattern, computes its error information term*
   $\delta' \Delta\alpha$
   $\delta k = (tk - yk)\ f'(y\_in\ k)$
   calculates its weight correction term used to update wjk later
   $\Delta w\ jk = \alpha\ \delta k\ Zj$
   calculates its bias correction term used to update w0k later
   $\Delta w\ 0k = \alpha\ \delta k$
   and sends δk *to units in the layer below*

8. *Each hidden unit (Zj, j =1...p) sums its delta inputs (from units in the layer above)*

   $\delta\_inj = \sum \delta k\ wjk$
   multiplies by the derivative of its activation function to calculate its error information term

   $\delta j = \delta\_inj\ f'(z\_in\ )$
   calculates its weight correction term (used to update vij later)

   $\Delta v\ ij = \alpha\ \delta k\ xi$
   and calculates its bias correction term(used to update v0j later)
   $\Delta v\ 0j = \alpha\ \delta j$

9. *Update weights and biases:*
   Each output unit (Yk, k=1...m) updates its bias and weights (j =0...p)
   $W\ jk(new) = Wjk(old) + \Delta Wjk$

   Each hidden unit (Zj, j=1...p) updates its bias and weights (i=0...n)
   $Vij(new) = Vij(old) + \Delta Vij$

10. *Test Stopping Condition*

The mathematical basis for the back propagation algorithm is the optimization technique known as *gradient descent*. The gradient of a function (in this case, the function is the error and the variables are the weights of the net) gives the direction in which the function increases more rapidly; the negative of the gradient gives the direction in which the function decreases more rapidly.

Initialization can be done randomly there are many procedures for this. A common procedure is to initialize the weights (and biases) to random values between −0.5 and 0.5 or some other suitable interval. The Nguyen-Widrow initialization makes simple modifications to the common random weight initialization. The approach is based on a geometrical analysis of the response of the hidden neurons to a single input.

Shown below are training graphs of *Performance vs Epochs* for few of the networks implemented. The training pairs, number of epochs and performance requirements have been varied and experimented with. The change in the number of hidden layers poses an interesting case. It follows the standard bell curve for performance, i.e. when the number of layers is just less the performance increases on increasing the number of layers, but after a saturation point increase in the number of layers inhibits performance.

The graphs shown below are graphs of *Performance vs Epochs* of networks used to solve the *Iris Flower Classification Problem*. Inputs of 50 flowers of each type were entered, totaling to 150. The three classes of flowers were *Setosa*, *Versi Color* and *Virginica*. Four parameters were chosen based on statistical studies for identifying the class. These were the 1) Sepal Length 2) Sepal Width 3) Petal Length 4) Petal Width. The values of these parameters for 150 flowers were entered as the input and their corresponding classes were *normalized* and taken as outputs. After the training was performed, each of the networks was tested with separate test data kept aside for the simulation.

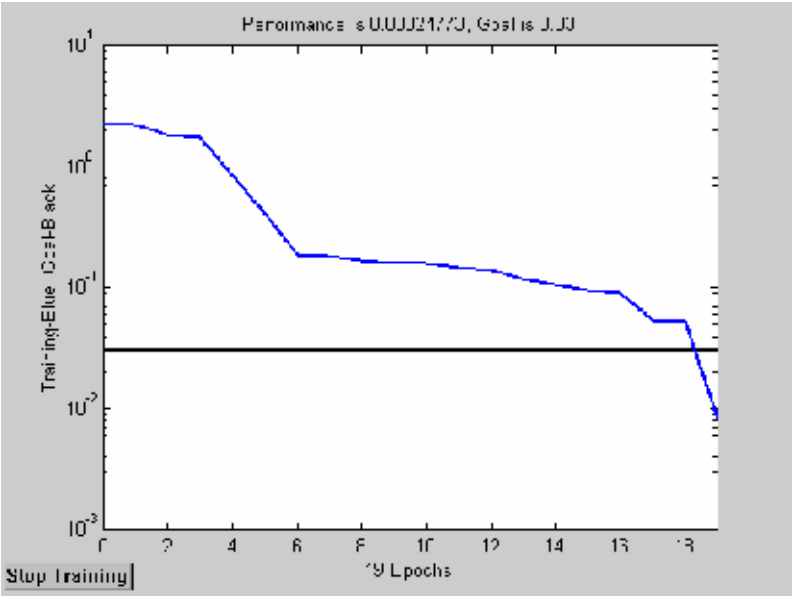


Fig 4. Training with the Cascade Feed Forward Network using trainlm function

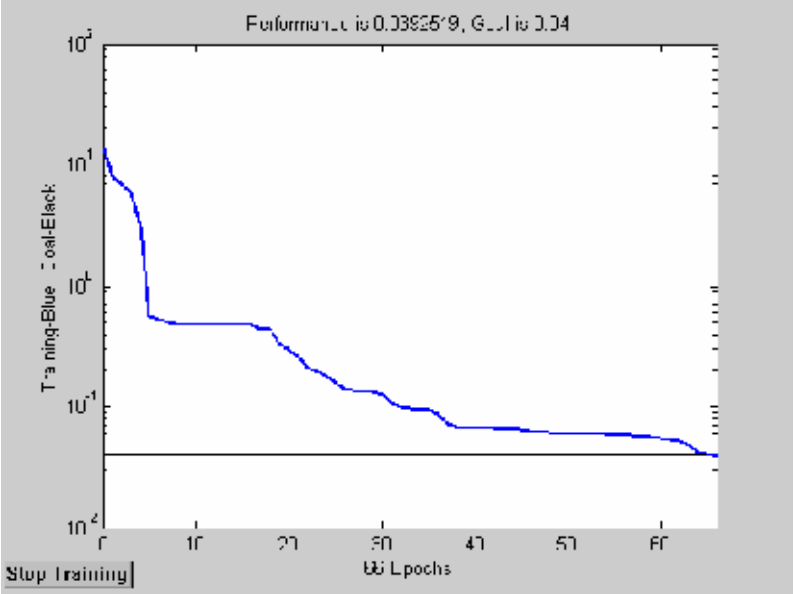


Fig 5. Training with the Cascade Forward Network using the trainbfg function

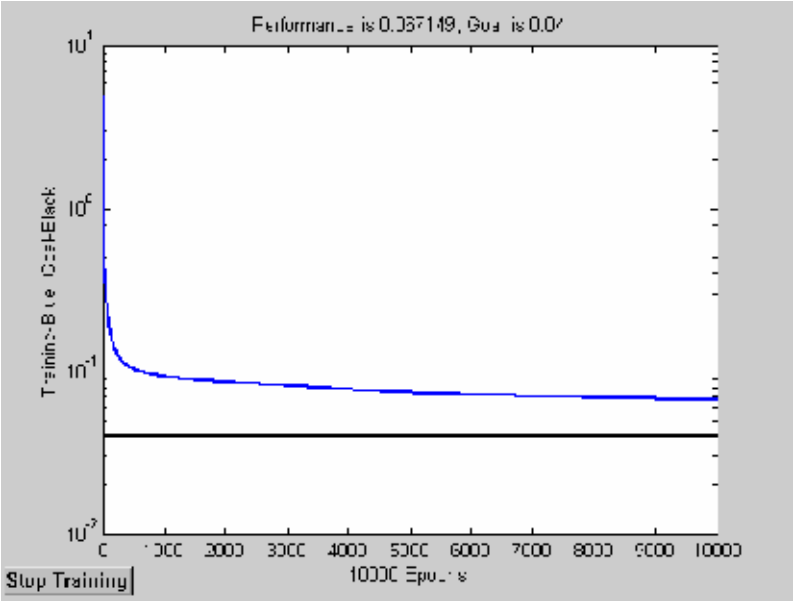


Fig 6 Training with Feed forward network using traingd function

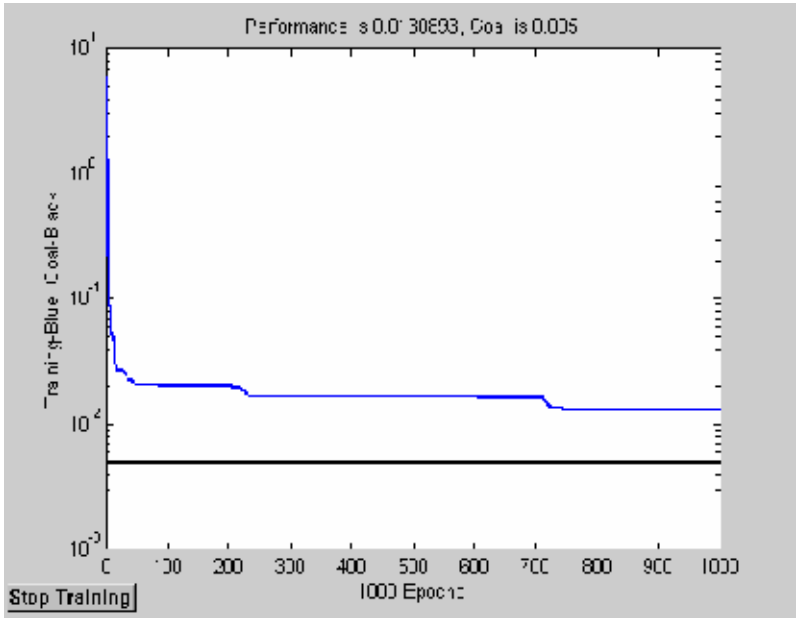


Fig7 Training with the Elman Backward Propagation Algorithm using trainlm function

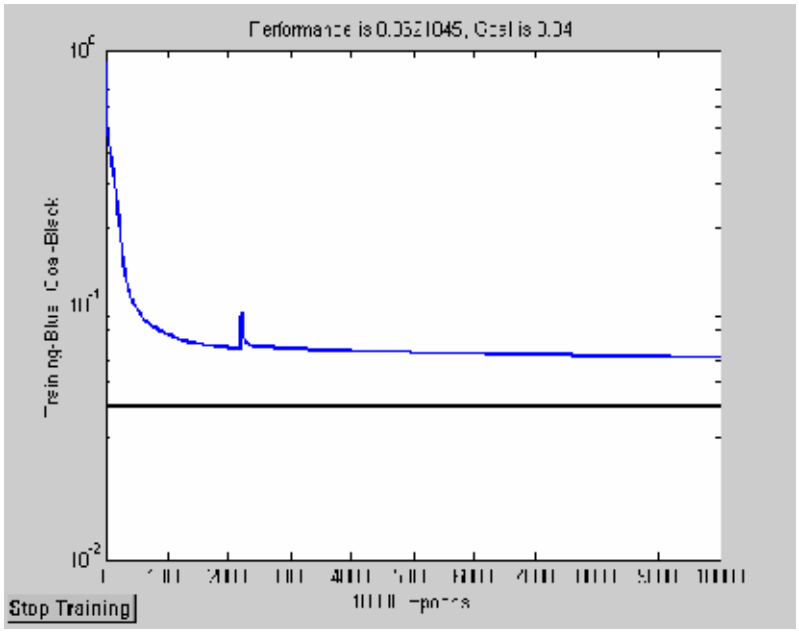


Fig 8. Training with the Elman Backward Propagation Algorithm using the traingd function

11. CONCLUSION

The report consists only few of the existing networks. Radial Basis Function, Linear Vector Quantization and other models have not been discussed in this short document. The following is a summary of the various networks, the training algorithms they are used with and their applications.

Learning Paradigm	Learning Rule	Architecture	Learning Algorithm	Task
Supervised	Error-correction	Single- or Multi-layer Perceptron	Perceptron learning algorithms Backpropagation ADALINE & MADALINE	pattern classification function approximation control
	Boltzmann	Recurrent	Boltzmann Learning algorithm	pattern classification
	Hebbian	Multi-layer Feedforward	Linear Discriminant Analysis	data analysis pattern classification
	Competitive	Competitive	Learning Vector Quantization	within-class categorization data compression
		ART network	ARTMAP	pattern classification within-class categorization
Unsupervised	Error-correction	Multi-layer Feedforward	Sammon's projection	data analysis
	Hebbian	Feedforward or Competitive	Principal Component Analysis	data analysis data compression
		Hopfield Net	Associative memory learning	associative memory
	Competitive	Competitive	Vector Quantization	categorization data compression
		Kohonen SOM	Kohonen's SOM	categorization data analysis
		ART networks	ART1, ART2	categorization
Hybrid	Error-correction and Competitive	RBF network	RBF Learning algorithm	pattern classification function approximation control

Fig 9. Well known learning algorithms

12. References:

1. Fundamentals of Neural Networks – Architectures, Algorithms and Applications by Laurene Fausett, Pearson Education
2. Artificial Neural Networks : A Tutorial , Anil K Jain, Jianchang Mao and K.Mohiuddin, IEEE Computer Special Issue on Neural Computing, March 1996
3. W.S. McCulloch and W.Pitts, a logical calculus of ideas immanent in nervous activity. *Bulletin of mathematical biophysics, 5:115-133, 1943*
4. S.Haykin. *Neural Networks:* A comprehensive Foundation. MacMillan College Publishing Company, New York, 1994
5. R.P.Lippman. An introduction to computing with neural nets. *IEEE ASSP Magazine,* 4 (2): 4-22, Apr 1987
6. M.Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine,* 65(2): 34-51, 1991
7. Tutorials from Mathworks Online ™