# Different Aspects for Enhancing The Backpropagation Neural Networks

**Hussein Rady**

El-Shorouk Academy, Higher Institute for Computer & Information Technology,
Tel.: 0106093311
E-mail: dr_hussein_rady@yahoo.com

**Abstract**

Backpropagation (BP) algorithm is one of the most popular training algorithms for multilayer neural networks. The convergence of backpropagation learning is analyzed so as to explain common phenomenon observed by specialists. The performance of the backpropagation algorithm is studied, analysed and evaluated in this paper. A method for accelerating the convergence rate is presented. It provides useful guidelines for thinking about how to accelerate the convergence through learning rate adaptation. This work has been implemented through computer simulated using C# with different activation functions and different methods for representing the learning rates. The obtained results are encourage and promising.

**Keywords:** Artificial Neural Network, Backpropagation, Activation Functions, Learning Rates, Momentum.

## 1. Introduction:

Artificial neural network ( ANNs ) are mathematical models developed to imitate information storing and processing capabilities of the human brain. These models are developed with a quit different philosophy of information processing from that of conventional computers. It is hoped that they will overcome the conventional computer's limitation on "intelligent" information processing capability. They are assumed to be assembled from neuron-like cells that are connected by links with adjustable strengths/weights. The most attractive characteristic of ANNs is that they can be trained to perform computational tasks using some learning algorithm and few examples[14].

Neural networks, also known as connectionist systems, or parallel distributed processing models, are computer-based, self-adaptive models of Artificial intelligence. The intelligence of Artificial Neural Network and its capability to solve hard problems emerges from the high degree of connectivity that gives neurons its high computational power through its massive parallel-distributed structure [12, 17].

Backpropagation algorithm [1,4,5,9,13,16] is used for training artificial neural networks. Training is usually carried out by iterative updating of weights based on minimizing the mean square error. In the output layer, the error signal is the difference between the desired and the output values. Then the error signal is fed back through the steepest descent algorithm to the lower layers to update the weights of the network. The weights of the network are adjusted by the algorithm such that the error is decreased along a descent direction. Traditionally, two parameters, called learning rate and momentum factor, are used for controlling the weight adjustment along the descent direction and for dampening oscillations. However, the convergence rate of the BP algorithm is relatively slow, especially for networks with more than one hidden layer. The reason for this is the saturation behavior of the activation function used for the hidden and output layers. Since the output of a unit exists in the saturation area, the corresponding descent gradient takes a very small value, even if the output error is large, leading to very little progress in the weight adjustment.

The selection of the learning rate and momentum factor is arbitrary, because the error surface usually consists of many flat and steep regions and behaves differently from application to application. Large values of the learning rate and momentum factor are helpful to accelerate learning. However, this increases the possibility of the weight search jumping over steep regions and moving out of the desired regions [18].

The organization of the paper presented in the following sections. In the next section, the algorithm of the ANN Learning is presented. A modification of the original backpropagation algorithm is presented in section 4. the simulated results and other aspects are discussed in section 5,6,7 and finally in section 8, conclusions and future work are outlined.

## 2- Models of a neoron

A basic model of a neuron is depicted in Figure(1). Artificial neuron is a device with many inputs and one output. It is the information processing unit, which forms the basis for designing artificial networks [2, 7].
Three basic neural elements are identified:

- A set of weights, which characterize the strength of the synapse linking a neuron to another. Synaptic weight linking a neuron 'j' to a neuron 'i' could be identified either by $W_{ji}$ or by $W_{ij}$.
- An adder for summing the input neuron's inputs modulated by the respective connection weights.
- An activation function, which is applied on the input summation for limiting the amplitude of the neuron's output.

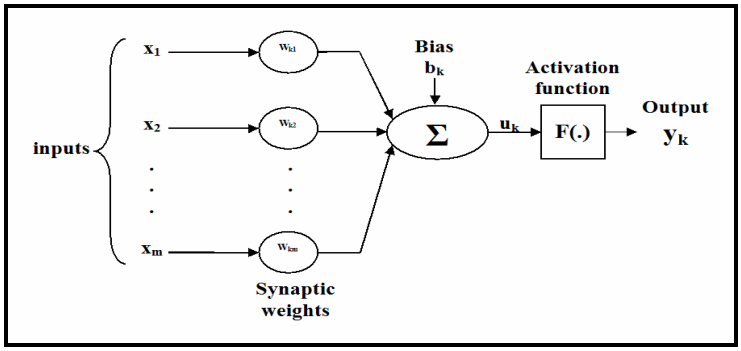


Figure (1) Nonlinear model of a neuron.

Mathematically, the neural model is described by the following equations

$$u_k = \sum_{j=1}^{m} W_{kj} x_j \quad (1)$$

$$y_k = F(u_k + b_k) \quad (2)$$

where $x_1, x_2, \ldots, x_m$ represent the input features; $W_{k1}, W_{k2}, \ldots, W_{km}$ are the synaptic weights of the neuron k;
$u_k$ is the linear summation of the input features;
$b_k$ is the bias;
F(.) is the activation function which can take several forms discussed later.
$y_k$ is the neuron output.
By including the bias in the linear summation of the input features, the combination could be reformulated as follows:

$$v_k = \sum_{j=0}^{m} W_{kj} x_j \quad \text{, and } y_k = F(v_k) \quad (3)$$

where $x_0 = +1$, and $W_{k0} = b_k$. (4)

**3- Algorithm of the ANN Learning:**

Learning of the ANN denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time [8]. Learning can refer to either acquiring new knowledge or enhancing or refining skills. Learning new knowledge include acquisition of significant concepts, understanding of their meanings and relationships to each other and to the domain concerned. The new knowledge should be assigned and put in a mentally usable form before it can be

called "learned." Thus, knowledge acquisition is designed as learning new symbolic information combined with the ability to use that information effectively.

**Backpropagation (BP) Algorithm**

The BP algorithm for multi-layer neural networks is a gradient desent procedure used to minimize the mean square error. Using the PB algorithms is summarized as [1, 3, 5, 9]:

Training process:

Step 1: Choose the structure of neural network and input parameter of the network.

Step 2: Set up initial values for both the initial weight W and the bias elements θ .

Step 3: Input training observation data matrix X and Target output matrix T.

Step 4: Compute the output vector of each neural units

a) Compute the output vector H of the hidden layer

$$net_k = \sum_i W_{ik} X_i - \theta_k \quad (5)$$

$$H_k = F(net_k) \quad (6)$$

b) Compute the output vector Y of the output layer

$$net_j = \sum_k W_{kj} H_k - \theta_j \quad (7)$$

$$Y_j = F(net_j) \quad (8)$$

Step 5: Compute the errors

a) Compute the errors $\delta_j$ of the output layer

$$\delta_j = (T_j - Y_j).F'(net_j) \quad (9)$$

*where $F'(net_j)$ is the derivation of the activation function.*

b) Compute the errors $\delta_k$ of the hidden layer

$$\delta_K = \left( \sum_j \delta_j W_{kj} \right).F'(net_j) \quad (10)$$

Step 6: Compute the modification of W and θ (η is the learning rate)

a) Compute the modification of W and θ of the output layer

$$\Delta W_{kj} = \eta \delta_j H_k \quad (11)$$

$$\Delta \theta_j = -\eta \delta_j \quad (12)$$

b) Compute the modification of W and θ of the hidden layer

$$\Delta W_{ik} = \eta \delta_k X_i \quad (13)$$

$$\Delta \theta_k = -\eta \delta_k \quad (14)$$

Step 7: Renew W and θ

a) Renew W and θ of the output layer

$$W_{kj} = W_{kj} + \Delta W_{kj} \quad (15)$$

$$\theta_j = \theta_j + \Delta\theta_j \quad (16)$$

b) Renew W and θ of the hidden layer

$$W_{ik} = W_{ik} + \Delta W_{ik} \quad (17)$$

$$\theta_k = \theta_k + \Delta\theta_k \quad (18)$$

Step 8: Repeat step 3 to step 7 until convergence

Testing process:

Step 1: Input the parameters of the network.
Step 2: Input the W and θ.
Step 3: Input an unknown data matrix X.
Step 4: Compute the output vector.

a) Compute the output vector H of hidden layer

$$net_k = \sum_i W_{ik} X_i - \theta_k \quad (19)$$

$$H_k = F(net_k) \quad (20)$$

b) Compute the output vector Y of the output layer

$$net_j = \sum_k W_{kj} H_k - \theta_j \quad (21)$$

$$Y_j = F(net_j) \quad (22)$$

## 4- Modification of the original backpropagation Algorithm.

The standard backpropagation learning algorithm is called vanilla backpropagation[10]. It is also called online backpropagation because it updates the weights after every training pattern. The modification of that algorithm is discussed as follows:-

***Enhanced backpropagation.*** The momentum strategy can be considered as an approximation to the conjugate gradient method[14], because in both the present gradient direction is modified using a term that takes the previous direction into account. The equation for weight change is given by:

$$W_{ij}(t + 1) = W_{ij}(t) + \eta\delta_j O_i + \alpha [W_{ij}(t) - W_{ij}(t - 1)] \quad (23)$$

where $0 < \alpha < 1$ is a momentum coefficient.

***Batch backpropagation.*** Batch BP has a similar formula as the original (vanilla) BP. The difference lies in the time when the update of the links take place. While in the original BP an update step is performed after each single pattern, in batch BP all weight changes are summed over a full presentation of all training patterns (one epoch). Only then, an update with the accumulated weight changes is performed.

***backpropagation with Weight Decay.*** Weight Decay decreases the weights of the links while training them with BP. In addition to each update of a weight by BP, the weight is decreased by a part d of its old value. The resulting formula is:

$$\Delta W_{ij}(t+1) = \eta\delta_j O_i - d\, W_{ij}(t) \qquad (24)$$

**5- Simulated results for different activation functions:**

The behavior of an artificial neural network depends on both the weights and the input-output function (energy function or activation function) that is specified for the units. Because the standard BP algorithm descends along the gradient of the error surface, the use of any activation function that has a larger gradient than that of the sum of the squared error at higher energy values would make for faster training. There are many kinds of activation functions such as the sigmoid function, the Cauchy energy function, the polynomial energy function, and the exponential energy function [2, 14].

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. The reason is that a linear function of linear functions is again a linear function. However, it is the nonlinearity that makes multilayer networks so powerful. Almost any nonlinear function does the job, except for polynomials. For BP learning, the activation function must be differentiable, and it helps if the function is bounded; the sigmoidal functions such as logistic, tanh and the Gaussian function are the most common choices. Functions such as tanh or arctan that produce both positive and negative values tend to yield faster training than functions that produce only positive values such as logistic, because of better numerical conditioning. In this section I discuss the category of the activation functions in the following subsections:

**5.1** ***The sigmoid function***. The sigmoid function can be written in the form:

$$F(v) = \frac{1}{1-e^{-v}} \qquad -\infty < v < \infty \qquad (25)$$

$$F'(v) = \frac{e^{-v}}{[1+e^{-v}]^2}$$

$$= F(v)\,(1 - F(v))$$

using this energy function, the activation level is between 0 and 1.

For a neuron j located in the output layer

$$\delta_j = O_j\,(1 - O_j)\,(T_j - O_j) \qquad (26)$$

For a neuron j located in the hidden layer

$$\delta_j = O_j\,(1 - O_j) \sum_k \delta_k W_{ik} \qquad (27)$$

where neuron j is hidden

Running the program on a simulated data set and a fixed learning rate, figure(2) shows a comparison of the output using the sigmoid function and tanch function.

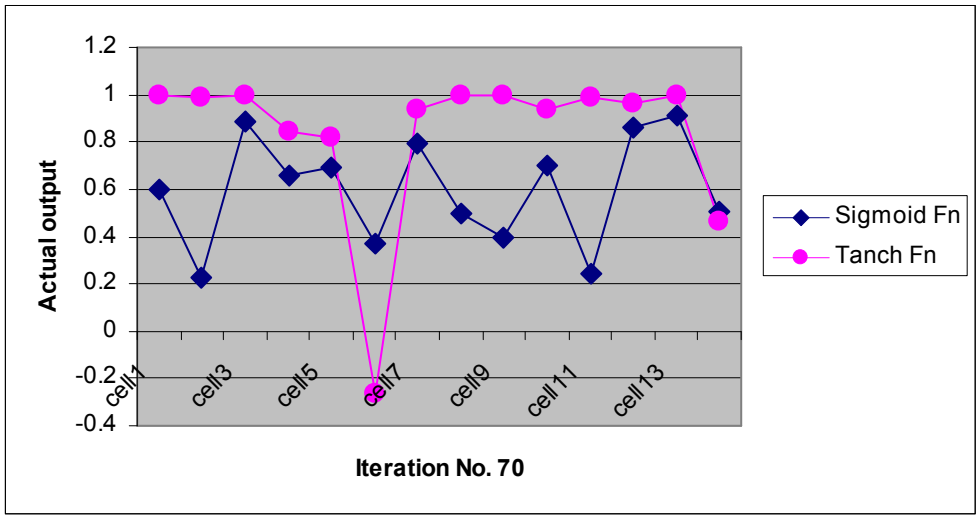


Figure (2 ) Comparing the output using the sigmoid & tanch function for a fixed learning rate.

**5.2** It has been found that if the activation level is restricted to the range from –1/2 to 1/2, convergence time may be reduced by half compared with the 0 to 1 range. This improvement proceeds from the fact that a weight coming from a neuron of zero activation will not be modified. This idea is implemented by changing the input range to –1/2 and 1/2 and using the following activation function:

$$-\frac{1}{2}+\frac{1}{1+e^{-v}} \qquad (28)$$

where v is the argument of the function. Equation (22) and (23) are also applied here for a neuron j located in the output and hidden layers respectively.

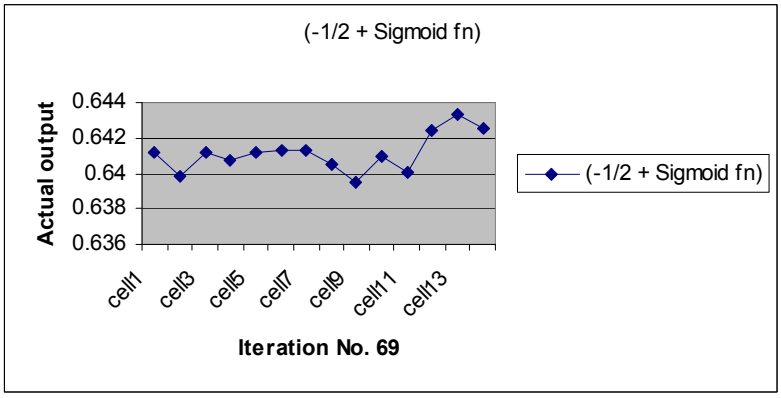


Figure (3) part of the output using the function -1/2 + 1 / (1 + $e^{-v}$).

When using the same data set and the same learning rate, we obtain the same results as in Figure(2), but the difference in this case is the convergence time is accelerated. In Figure(3) an enhancement occurs by decreasing the learning rate by a certain factor at each iteration.

***5.3 The hyperbolic tangent function.*** A multilayer perception trained with the BP algorithm may, in general, learn faster (in terms of the numbers of training iterations required) when the sigmoid activation function built into the neuron model of the network is antisymmetric than when it is nonsymmetric. We say that an activation function F(v) is antisymmetric (i.e., odd function of its argument) if

$F(-v) = - F(v)$

A popular example of an antisymmetric activation function is a sigmoidal nonlinearity in the form of a hyperbolic target, defined by:

$$F(v) = \frac{1 - \exp(-v)}{1 + \exp(-v)}$$

$$= \tanh(v) \qquad (29)$$

The limiting values of this function are -1 and +1.

The derivative of F(v) with respect to v is :

$F(v) = \operatorname{sech}^2(v)$

$= [1 - \tanh^2(v)]$

$= 1 - F^2(v)]$

$= [1 - F(v)][1 + F(v)]$

For a neuron j located in the output layer

$$\delta_j = (1 - O_j)(1 + O_j)(T_j - O_j) \qquad (30)$$

For a neuron j located in the hidden layer

$$\delta_j = (1 - O_j)(1 + O_j) \sum_k \delta_k W_{jk} \qquad (31)$$

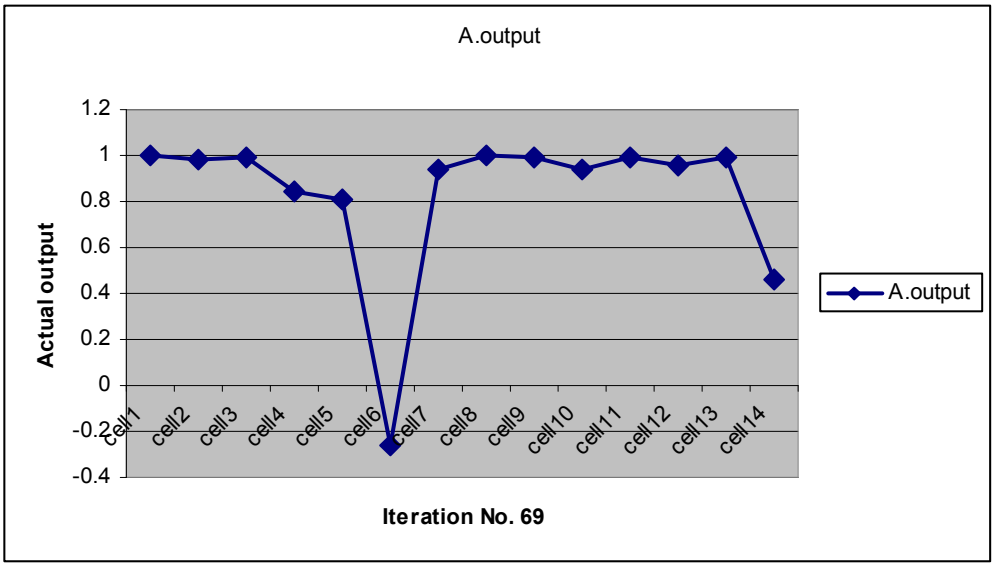


Figure (4) part of the output using the tanh function for the same learning rate.

Figure(4) shows part of the output using the same weights and the same learning rate used previously in the sigmoid function. The tanh function accelerates the convergence than the sigmoid function as shown from Figures (2), (4). This shows that the antisymmetric function is faster than the sigmoid function.

***5.4 Algebraic sigmoid function.*** This is another odd signed function that has the following formula:

$$F(v) = \frac{v}{\sqrt{1+v^2}} \tag{32}$$

Whose limiting values are -1 and +1.

$$F'(v) = \frac{1}{(\sqrt{1+v^2})^3}$$

$$= \frac{v^3}{v^3(\sqrt{1+v^2})^3}$$

$$= \frac{F^3(v)}{v^3}$$

For a neuron j located in the output layer

$$\delta_j = \frac{O_j^{\,3}}{v^3} (T_j - O_j) \quad (33)$$

For a neuron j located in the hidden layer

$$\delta_j = \frac{O_j^{\,3}}{v^3} \sum_k \delta_k W_{jk} \quad (34)$$

where neuron j is hidden.

## 6- Simulated results for learning rates

All neurons in the multilayer perception should ideally learn at the same rate. The last layers usually have layer local gradients that the layers at the front end of the networks. Hence, the learning-rate parameter should be assigned a smaller value in the last layers than in the front layers. Neurons with many inputs should have a smaller learning-rate parameter than neurons with few inputs so as to maintain a similar learning time for all neurons in the network. It is suggested that for a given neuron, the learning rate should be inversely proportional to the square root of synaptic connection made to that neuron. On the other hand, many schemes have been proposed to automatically adjust the learning rate. Most of those schemes decrease the learning rate when the weight vector "oscillates", and increase it when the weight vector follows a relatively steady direction. The main problem with these methods is that they are not appropriate for stochastic gradient or on-line learning because the weight vector fluctuates all the time.

Beyond choosing a single global learning rate, it is clear that picking a different learning rate for each weight can improve the convergence [11]. A well-principled way of doing this, based on computing second derivatives. The main philosophy is to make sure that all the weights in the network converge roughly at the same speed.

Depending upon the curvature of the error surface, some weights may require a small learning rate in order to avoid divergence, while others may require a large learning rate in order to converge at a reasonable speed. Because of this, learning rates in the lower layers should generally be large than in the higher layers. This corrects for the fact that in most neural net architectures, the second derivative of the cost function with respect to weights in the lower layers is generally smaller than that of the higher layers [6]. In the following subsections, I discuss some heuristics that provide useful guidelines for thinking about how to accelerate the convergence of BP learning through learning rate adaptation.

**6.1** *Fixed calculating of the learning rate one way to optimize the learning rate automatically*. A calculation of the learning rate for BP using batched updates is based on the assumption that similar training patterns result in similar gradients. So

it is desirable to reduce the learning rate if there are many similar training patterns. Therefore the training set must be divided in m subsets of similar patterns. Let $N_1$, $N_2$. …, $N_m$ be the sizes of these subsets. Learning rate and momentum can now be set in the following manner [15]:

$$\eta = \frac{1.5}{\sqrt{N_1^{\,2} + N_2^{\,2} + \ldots + N_m^{\,2}}} \tag{35}$$

momentum = 0.9

**6.2** *Every adjustable network parameter of the cost function should have its own individual learning rate parameter.* The BP algorithm may be slow to coverage because the use of a fixed learning rate parameter may not suit all portions of the error surface. In other words, a learning rate parameter appropriation for the adjustment of one synaptic weight is not necessarily appropriate for the adjustment of other synaptic weights in the network. Thus assigning a different learning rate parameter to each adjustable synaptic weight in the network is preferred.

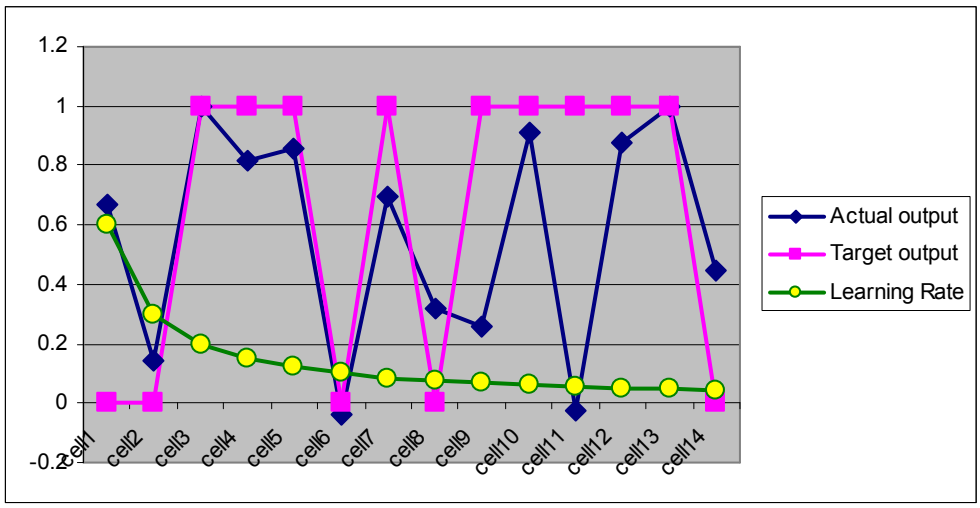


Figure (5) part of the output using tanh function for different learning rates for Each cell in Iteration No. 69.

**6.3** *Every learning rate parameter should be allowed to vary from one iteration to the next.* The error surface, typically behaves differently along different regions of a single weight dimension. In order to match this variation the learning rate parameter needs to vary from iteration to iteration.

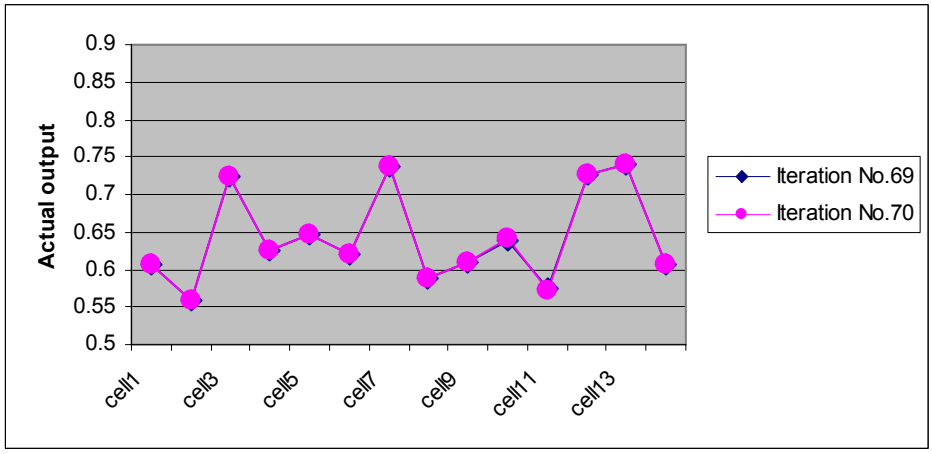


Figure (6) A figure of the output using tanh function for different learning rates in different iterations.

The learning rate parameter is time-varying. The particular time-varying form most commonly used is described by:

$$c / (1 + n) \qquad (36)$$

where n is the number of iteration, c is a constant.

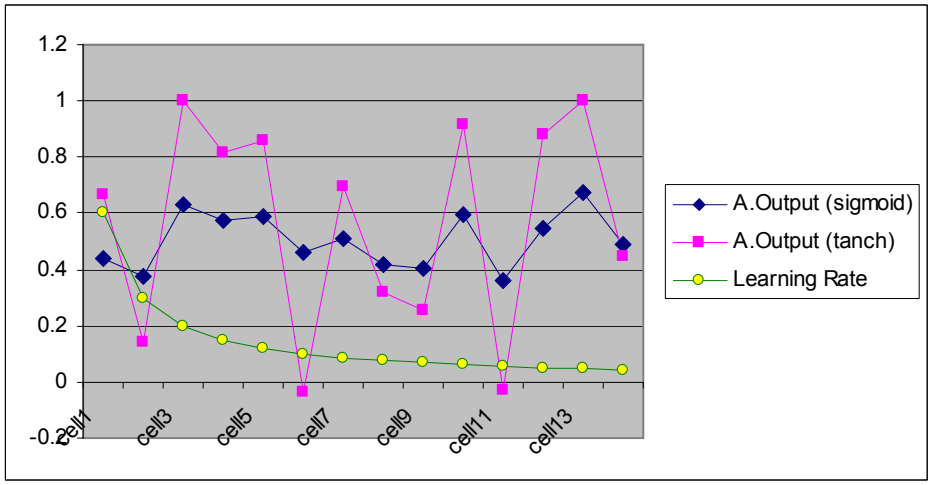


Figure (7) comparing the actual output for iteration No. 69 using the sigmoid and Tanch functions for different learning rates in the same iteration.

To decrease the learning rate during the training, a method called "Search-Then-Converge" strategy is suggested for BP using updates for every training pattern. Starting with a big learning rate $\eta(0)$ the value is decreased during the training to[5] :

$$\eta(n) = \eta(0) / ( 1 + n / r) \qquad (37)$$

The constant parameter r can be used to adjust this learning rate schedule with respect to the total training period. After the first r learning steps the learning rate is halved by this update rule. A good r can only be found by trial and error.

With respect to the tanch function, Figures(5) illustrates the output using different learning rates for each pattern. Figure(6) shows part of the output using tanch function for the same learning rates in each pattern, but it differ from iteration to iteration. Figure(7) compares the actual output for each of the sigmoid and tanch functions for iteration no. 70 using the same learning rates in each cell. Figure(8) shows part of the output using the sigmoid function for the same learning rates within the same iteration. These learning rates differ from iteration to iteration. Figure(9) shows part of the output for different learning rates in the cells of each iteration using the sigmoid function.

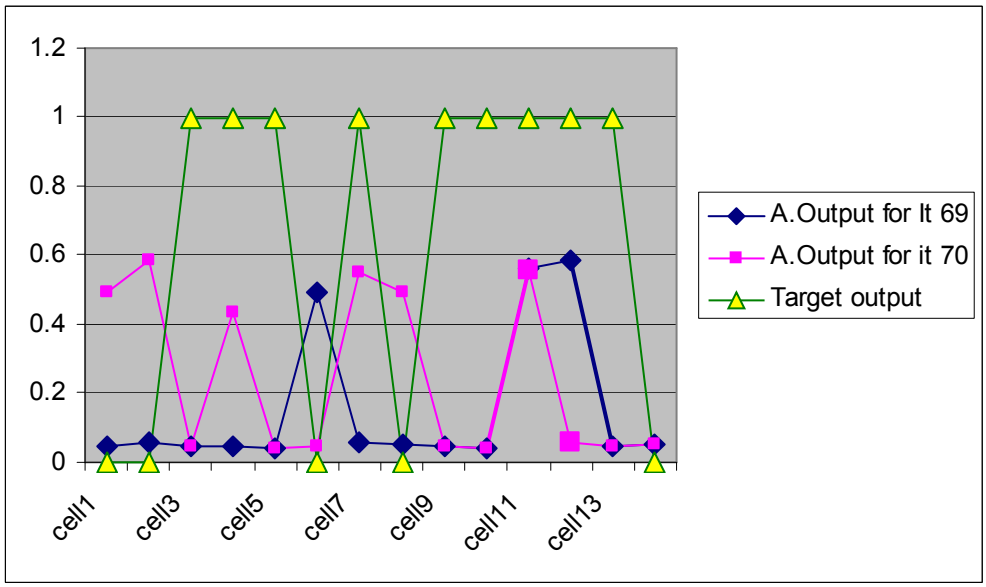


Figure (8) This figure shows part of the output for different learning rates for each iteration using the sigmoid function.

**6.4** *When the derivation of the cost function with respect to a synaptic weight has the same algebraic sign for several consecutive iteration of the algorithm, the learning rate parameter for that particular weight should be increased.* The current operating point in weight space may be on a relatively flat portion of the error surface along a particular weight dimension. This may in turn account for the derivative of the cost function (i.e., the gradient of the error surface) with respect to the weight maintaining the same algebraic sign, and therefore pointing in the same direction, for several consecutive iteration of the algorithm. Therefore in such a situation the number of iteration required to more a cross the flat portion of the error surface may be reduced by appropriately increasing the learning rate parameter.

**6.5** *When the algebraic sign of the derivative of the cost function with respect to a particular synaptic weight alternates for several consecutive iteration of the algorithm, the learning rate parameter for that weight should be decreased.* When the current operating point in weight space lies on a portion of the error surface along a weight dimension of interest that exhibits peals and valleys (i.e., the surface is highly curved), then it is possible for the derivative of the cost function with respect to that weight to change its algebraic sign from one iteration to the next. In order to prevent the weight adjustment from oscillating, the learning rate parameter for that particular weight should be decreased appropriately.

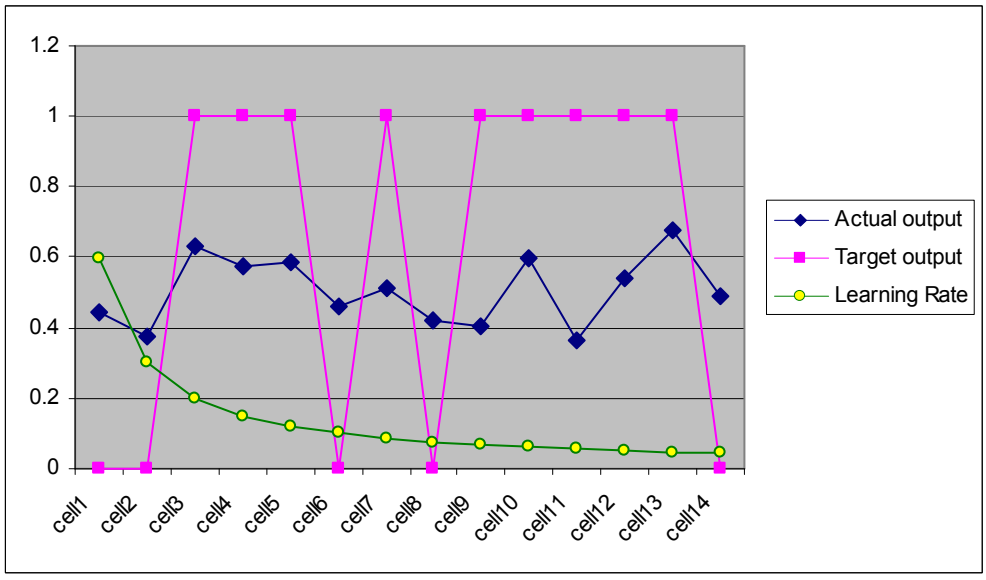


Figure (9) this part of the output using the sigmoid function for different learning rates in iteration No. 69.

## 7- Other aspects

In this section, I discuss some other aspects that affects the performance of the BP neural network algorithm. These aspects are: maximizing information content, target values, normalizing the inputs and initialization.

***Maximizing information content.*** As a general rule, every training example presented to the BP algorithm should be chosen on the basis that its information content is the largest possible for the task at hand. Two ways of achieving this aim are:

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

These two heuristics are motivated by a desire to search more of the weight space[5].

***Target values.*** It is important that the target values (desired response) be chosen within the range of the sigmoid activation function. More specifically, the desired response $T_j$ for neuron j in the output layer of the multilayer perceptron should be offset by some amount ɵ away from the limiting value of the sigmoid activation function, depending on whether the limiting value is positive or negative. Otherwise the BP algorithm tends to drive the free parameters of the network to infinity, and thereby slow down the learning process by driving the hidden neurons into saturation.

***Normalizing the inputs.*** Each input variable should be preprocessed so that its mean value, averaged over the entire training set, is close to zero, or else it is small compared to its standard deviation. To appreciate the practical significance of this rule, consider the extreme case where the input variables are consistently positive. In this situation, the synaptic weights of a neuron in the first hidden layer can only increase together or decrease together. Accordingly, if the weight vector of that neuron is to change direction, it can only do so by zigzagging its way through the error surface, which is typically slow and should therefore be avoided.

***Initialization.*** A good choice for the initial values of the synaptic weights and thresholds of the network can be of tremendous help in a successful network design. When the synaptic weights are assigned large initial values, it is highly likely that the neurons in the network will be driven into saturation. If this happens, the local gradients in the BP algorithm assume small values, which in turn will cause the learning process to slow down. However, if the synaptic weights are assigned small initial values, the BP algorithm may operate on a very flat area

around the origin of the error surface. Therefore, the use of both large and small values for initializing the synaptic weights should be avoided. The proper choice of initialization lies somewhere between these two extreme cases.

## 8- Conclusions

The backpropagtion algorithm has emerged as the most popular algorithm for the supervised training of multilayer perceptrons. Therefore it has a lot of research attention. In this paper a set of different approaches are developed and introduced to improve the learning rate and a different activation functions are proposed. We conclude that the tanh function as an activation function accelerates the convergence than the sigmoid function. Subtracting half from the sigmoid function, convergence time nearly reduced by half. Also we conclude that choosing a different learning rate for each weight improve the convergence. Varying the learning rate from one iteration to iteration is preferred than choosing a constant learning rate. These conclusions and others, contribute for enhancing the Backpropagation neural networks.

As a future work, the entropy techniques can be used, tested, evaluated and compared with the least square error as a stopping criteria. Other activation functions, different learning rate algorithms and statistical analysis can be adopted to enhance the capability of the Backpropagation neural networks.

## Acknowledgment

The author would like to thank Prof. Dr. Ahmed Gaber and Dr. Farouk Shabaan for the completely support to this work.

## References:

1- Abraham A., and Nath B.: "Hybrid Heuristics for Optimal Design of Artificial Neural Networks."
2- Brown A. (2004): "Analysing a Neural Network trained by backpropagation."
3- Chang P., and Shih J. (2002): "The Application of Backpropagation Neural Network of Multi-channel Piezoelectric Quartze Crystal Sensor for Mixed Organic Vapours."
4- Ellingsen B.: "A Comparative Analysis of Backpropagation and Counterpropagation Neural Networks."
5- Hybin S. (1999): "Neural Networks, A Comprehensive Foundation."
6- Lecun y., Botton L., OvviG, and Muller K.R (1998): "Efficient BackProp."
7- Lefley M., and Kinsellla T. (2000): "Investigating neural network efficiency and structure by weight investigation."

8- LiMin F. (2003): "Neural Networks in Computer Intelligence"
9- Luger G. (2002): "Artificial Intelligence, Structure and Strategies for Complex Problem Solving."
10- Mache N. (1995): "Backpropagation Networks."
11- Magoulas G. D., Vrahatis M. N., and Androulakis G. S. (1999): "Imporving the Convergence of the Backpropagation Algorithm using Learning Rate Adaptation Methods."
12- Malik N. (2005): "Artificial Neural Networks and their applications."
13- Ruiz M., and Srinivason (1998): "Automatic Text Categorization Using Neural Networks."
14- Sarkar D. (1995): "Methods to speed up Error Backpropagation Learning Algorithm.
15- Schiffmann W. Joost M., and Werner R. (1994): "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons."
16- Sher B., and Hseih W. (1999): "Fault Tolerant Training of Feedforward Neural Networks."
17- Torresen J., and Tomita S.: "A review of Parallel Implementation of Backpropagation Neural Networks."
18- Zweiri Y. (2006): "Optimization of a three- term Backpropagation Algorithm used for Neural Network Learning". International Journal of Computational Intelligence, volume 4.