NEW APPROACH TO A SELFTUNING BACK PROPAGATION ALGORITHM (SELF-PROP)

Dr: Ali Ahmed Abdulaziz[*]

Abstract:

In Artificial Neural Networks, Back-propagation algorithm is a first order approximation of the steepest-descent technique in the sense that it depends on the gradient of the instantaneous error surface in weight space. The algorithm is therefore "stochastic" in nature; that is, it has a tendency to zigzag its way about the true direction to a minimum on the error surface. However, the effect of using inexact learning technique, as opposed to exact ones, is to increase the number of iterations needed to reach a minimum in the cost function, while reducing the overall training time because each iteration requires less computation. What is well known is that the uses of a fixed step length gradient based back-propagation algorithm have difficulties with local minima. On the other hand, the use of more efficient classical minimisation algorithms is even more likely to be trapped in sub-optimal solutions.

* Assistant Professor, At the Faculty of Engineering Tahadi University.

The research work outlined in this paper aims to improve the convergence and learning rate of the gradient based back-propagation algorithm through possibilities of parallelism for different optimisation techniques. The favourable qualities, methodology and internal structure in the "Quick-prop" learning rule by Fahlman, the "Delta-Bar-Delta" learning rule by Jacobs, and the heuristic arguments for back-propagation improvement by Samad are investigated. Parallel version of these routines is proposed to form a new self-tuning back-propagation algorithm. The algorithms generally make a compromise between the computational complexity and the performance in terms of convergence rate and robustness. Analytic results; both theoretically and practical, as well as the effects of the tuning parameters of the new algorithm, is presented. Simulation results are presented with comparison of counterpart algorithms, e.g. the standard back-propagation, and the quick-prop back-propagation.

1) INTRODUCTION

Since the back-propagation rule [24, 28] was exploited it found wide acceptance in a number of fields; in control system approaches for example Back-propagation networks can adapt to changes in data and learn the characteristics of the input signal. They can learn a mapping between an input and output space and synthesise an associative memory that retrieves the appropriate output when presented with a known input and generalise when presented with a new input [7, 13, and 18]. Moreover, because of their non-linear nature, back-propagation networks can perform functional approximation and signal filtering operations that are beyond the capability of optimal linear control theory [33].

The back-propagation algorithm is a first order approximation of the steepest-descent technique in the sense that it depends on the gradient of the instantaneous error surface in weight space. The algorithm is therefore "stochastic" in nature; that is, it has a tendency to zigzag its way about the true direction to a minimum on the error surface.

Indeed, back-propagation learning is an application of a statistical method known as "stochastic approximation" that was originally proposed by Robbins and Monroe [22]. According to the empirical study of Saarinen et al. [25], the local convergence rates of the back-propagation algorithm are linear and justified the grounds that the Jacobean (matrix of first order partial derivatives) is almost rank-deficient. The Hessian (matrix of second order partial derivatives of the error surface with respect to the weights) are consequences of the intrinsically ill-conditioned nature of the neural-networks training problems. Saarinen et al. [25], interpret the linear local convergence rates of back-propagation learning in two ways:

1 - It is a vindication of back-propagation (gradient descent) in the sense that higher-order method, such as conjugate gradient or the quasi-Newton methods may not converge much faster while requiring more computation effort.

2 - Large-scale neural networks training problems are so inherently difficult to perform that no supervised learning strategy is feasible and other approaches such as the use of pre-processing may be necessary.

However, the effect of using inexact learning technique, as opposed to exact ones, is to increase the number of iterations needed to reach a minimum in the cost function. What is well known is that the use of a fixed step length gradient based back-propagation algorithm has difficulties with local minima. On the other hand, the use of more efficient classical minimisation algorithms is even more likely to be trapped in sub-optimal solutions.

The standard form of back-propagation error rule [24], although it has been tested, analysed and implemented quite successfully in a variety of situations, its limited because of its slowness and long convergence time. This may be attributed to the lack of self-tuning strategy, that is because each of the learning parameters, e.g. learning rate parameter, (ϵ), and learning rate momentum, (μ), have been tuned

manually by trial-and-error methods. Moreover, it is highly probable that it tends to a local minimum situation whereupon the tuning factor for connection weights is very small or even nil. It is for these forementioned limitations a number of formulations for replace or improved back-propagation algorithms already exist in the literature, see e.g. [5, 14, 15,19, 20, 26].

2) PROBLEM FORMULATION

Whilst the steepest descent method can be an effective method for obtaining the weight values that minimise an error measure, the error surface frequently possesses properties that make this procedure slow to converge. There are at least two reasons for this slow rate of convergence [14]. These reasons involve the magnitude of the component and the direction of the gradient vector.

The magnitude of a particular derivative of the error with respect to a weight may be such that modifying the weight by a constant proportion of that derivative yields a minor reduction in the error measure. This occurs in two situations: First where the error surface is fairly flat along a weight dimension in which the derivative of the weight is small in magnitude. Hence, the weight is adjusted by a small amount and therefore many steps are required to achieve a significant reduction in error measure. Alternatively, the error surface is highly curved along a weight dimension in which the derivative of the weight is large in magnitude. Hence, the weight value is adjustable by a large amount, which may overshoot the minimum of the error surface along that weight trajectory.

The second reason for the slow rate of convergence of a steepest descent algorithm is that the direction of the negative gradient vector may not point directly towards the minimum of the error surface, e.g. for a two dimensional weight space, illustrated in figure 1,

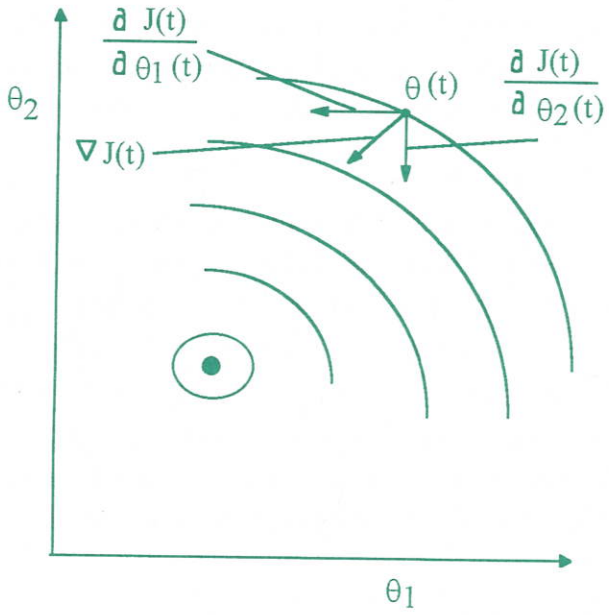


Figure 1: - Error surface over two-dimensional weight space.

Where $\nabla J(t) = \left[\frac{\partial J}{\partial \theta_1(t)}, \ldots, \frac{\partial J}{\partial \theta_n(t)}\right]$, and n is the number of weights.

The error surface is drawn topographically using half-contours to represent regions of equal error. The minimum of the error surface is represented by the black dot. The current weight vector is given by $\theta(t)$. Since at the current point in weight space the error surface is steeper along the θ_2 dimension than the θ_1 dimension, the derivative of θ_2 is greater than the derivative of θ_1. In general, if the direction of the negative gradient vector is equal to the direction of the minimum of the error surface for all points in weight space, the contours that represent regions of equal error is circular [10].

In this section we describe procedures for increasing the rate of convergence by maintaining the locality constraint which is inherent characteristic of the back-propagation learning algorithm [24] through two main concepts.

First: Based on the above discussion, the error vs. weight curve can be approximated by a parabola whose arms open upward, for each weight. Independently, we use the gradient descent, with self-tuning learning rate parameter, ϵ, to make a change in the weight proportional to the negative gradient of the derivative of the error surface, [14].

Second: The changes in the slope of error curve as seen by each weight may not be affected by all the other weights, i.e. all weights are changing at the same time. For each weight, independently, the previous and current error slopes and the weight change between the points at which these slopes were measured are used to determine a parabola. The computation is then to jump directly to the minimum point of that parabola by using a second order method based loosely on Newton's method (i.e. quadratic estimation technique), [5, and 27].

3) MODEL STRUCTURE

NN (X1-X2-X3) is a feed-forward neural net model with three layers, namely; an input layer of X1 nodes, a hidden layer of X2 nodes, and an output layer of X3 nodes. There is a full connection from every node in the input layer to every node in the hidden layer. The number of hidden layers is assumed to be of one or two, in most cases we preferred to use just one hidden layer where the mathematical justification for that has been shown in Abdulaziz and Farsi [1]. Each node in the first hidden layer is connected to all nodes in the second hidden layer. Then all nodes in the second hidden layer are fully connected to the output layer nodes. There are no direct connections between nodes in the input layer and nodes in the output layer. Both of the input and output layers are assumed to have linear transfer functions. This makes the output nodes in the output layer a summation of last hidden layer nodes activation. The transfer function in each of the hidden nodes is assumed to be of the tanh type function as we will see later.

4) DATA PRESENTATION

Data presentation plays an important part in updating the connection weights of the back-propagation algorithm [24, 28]. The "pattern data presentation" (sample presentation), and the "batch data presentation" (vector presentation) are common for updating the connection weights in back-propagation algorithm. In the former type presentation, the connection weights are updated immediately after each sample is fed into the net. In the latter all learning data (samples) are taken as a whole before the weight updating process begins. More detailed information about such data presentation schemes can be found in Werbos [28] and Hecht-Nielsen [12].

Here, only the pattern data presentation is considered. In this approach, each training sample is propagated forward through the network to produce an output. The output is then compared with the desired output, and the error difference is propagated back through the net to set the new changes for the connection weights. This pattern updating method is sometimes called "on-line" updating [12]. The motivation for using pattern learning is that it eliminates the need for memory requirements for the accumulated error vector in "batch" learning method. Furthermore, this method gives the true gradient in the weight space, which leads to fast convergence. The only disadvantage, however, as described by Fahlman [5] is the time consumption. That is because for each data sample the error signal is propagated back through the network structure to set the new connection weight values. This process strongly affects the speed of operation for the algorithm especially when more than one hidden layer with a large number of nodes is used. Using only one hidden layer with a previously determined number of node [1] may eliminate this problem.

5) TUNING STRATEGY

From the generalised delta rule by Jacobs's [14], the updating for node-j for example is given by:

$$\theta_{ij}(t) = \theta_{ij}(t-l) + \delta\theta_{ij}(t) + \mu\Delta\theta_{ij}(t-l) \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

The net internal activity level xj(t) produced at the input of the transfer function associated with node (neuron) j is therefore [30]

$$x_j(t) = \sum_{i=0}^{n} \theta_{ij}(t) \cdot y_i(t)$$

Where n is the total number of inputs applied to neuron j. The synaptic weight $\theta j0$ (corresponding to the fixed input y0 = 1) is equal to the threshold θj applied to neuron j. Hence the function signal yj(t) appearing at the output of node j at iteration t is

$$y_j(t) = \phi(x_j(t))$$

In which according to the gradient rule [24], $\Delta(t)$ is equal to

$$\Delta\theta_{ij} = -\varepsilon \frac{\partial J(t)}{\partial \theta(t)} = \varepsilon \delta_j(t)\ y_i(t) \quad \text{..... (2)}$$

Where $\delta j(t)$ is the local gradient, defined by

$$\delta_j(t) = -\frac{\partial J(t)}{\partial e_j(t)} \frac{\partial e_j(t)}{\partial y_j(t)} \frac{\partial y j(t)}{\partial x_j(t)}$$

$$= e_j(t) \cdot \phi'_j(x_j(t)) \qquad \text{(for the output nodes) ..... (3)}$$

And

$$\delta_j(t) = -\frac{\partial J(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial x_j(t)}$$

$$= -\frac{\partial J(t)}{\partial y_j(t)} \cdot \phi'_j\ (x_j(t)) \quad \text{(for hidden nodes)..... (4)}$$

The approach here is towards improving the updating formula in equation (2) through an on-line tuning for the learning rate parameter, ε. In this sense, based on the recommendation of Jacobs [14], four heuristics (H1 to H4) may be viewed as useful guidelines for thinking about how to accelerate the convergence of the back-propagation

learning through the learning rate parameter adaptation:

H1: Every adjustable network parameter of the performance measure should have its own individual learning rate.

As mentioned above the back-propagation rule may be slow because of the use of a fixed learning rate parameter, ε. This may not be suitable for all portions of the error surface. In other words the learning rate parameter, ε, may be adequate for the adjustment of one connection weight but may not be appropriate for the adjustment of the others. H1 recognises this fact by assigning a different learning rate parameter, ε, to each adjustable connection weight.

H2: learning rate parameter, ε, should be varying from one iteration to another.

It is a typical behaviour for the error surface to possess different properties along different regions of a single weight dimension. Thus, in order to take appropriate steps, H2 states that the learning rate parameter, ε, needs to vary from one iteration to another.

H3: When the derivative of the performance measure with respect to a connection weight has the same algebraic sign for several consecutive operations of the algorithm, the learning rate parameter for that particular weight should be increased.

This occurs when the current operating point in the weight space lies on a relatively flat portion of the error surface along a particular weight dimension. This may, in turn, account for the derivative of the performance measure with respect to the connection maintaining the same algebraic sign, and therefore pointing in the same direction for several consecutive operations. H3 states that in such situations the number of iterations required to move across the flat portion of the error surface may be reduced by increasing the learning rate parameter, ε, appropriately.

H4: When the sign of the derivative of the performance measure with respect to a particular connection weight alternates for several

consecutive iterations, the learning rate parameter, ε, for that weight should be decreased.

When the current operating point in weight space lies in a highly curved portion of the error surface. The error surface exhibits peaks and valleys, then it is possible for the derivative of the performance measure with respect to the connection weight to change its algebraic sign from one to another iteration. In order to prevent this quick change of sign, H4 states that the learning rate parameter, ε, should be decreased appropriately.

Hence, to accelerate convergence of the back-propagation algorithm through the learning rate parameter, ε, it is required to re-derive the back-propagation rule, by using the local learning rate parameter method [14]. In this new form of the back-propagation rule, the performance function (cost function) assumed to be governed by

$$J(t) = \frac{1}{2}\sum_{k} (e_k)^2 \quad \ldots\ldots (5)$$

Where

$$e_k(t) = w_k(t) - y_k(t) \quad \ldots\ldots (6)$$

Where $y_k(t)$ is the output of node k of the output layer, and $w_k(t)$ is the desired response for that node.

Now let $_{kj}(t)$ denote the learning rate parameter, ε, assigned to connection weight $_{kj}(t)$ at iteration t. Applying the chain rule to the derivative $\partial J(t)$ with respect to $\partial \varepsilon_{kj}(t)$ by using the $y_k(t)$ and $x_k(t)$ as

$$\frac{\partial J(t)}{\partial \varepsilon_{kj}(t)} = \frac{\partial J(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial x_k(t)} \frac{\partial x_k(t)}{\partial \varepsilon_{kj}(t)} \quad \ldots\ldots (7)$$

Thus, the parameters of equation (7) may be computed as follows:

$$x_k(t) = \sum_{j=0}^{n} \theta_{kj}(t) \cdot y_j(t) \quad \ldots\ldots (7)$$

Where

$$\theta_{kj}(t) = \theta_{kj}(t-1) - \varepsilon \frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)} + \mu \Delta \theta_{kj}(t-1) \quad \ldots\ldots\ldots\ldots (8)$$

Substituting (9) in (8), to get

$$x_k(t) = \sum_j y_j(t) \left[\theta_{kj}(t-1) - \varepsilon_{kj}(t) \frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)} + \mu \Delta \theta_{kj}(t-1) \right] \quad (10)$$

Hence, by differentiating equation (10) with respect to ϵ kj(t), yields

$$\frac{\partial x_j(t)}{\partial \varepsilon_{kj}(t)} = -y_j(t) \frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)} \quad \ldots\ldots\ldots\ldots (11)$$

The derivative of yk(t) with respect to xk(t) in equation (7) may be defined as:

$$\frac{\partial y_k(t)}{\partial x_k(t)} = \phi_k' (x_k(t)) \quad \ldots\ldots\ldots\ldots (12)$$

Next, evaluate the derivative of J(t) with respect to yk(t). First consider the case when the node k is an output layer node where the desired response wj(t) is known. Accordingly, from equation (5) we get:

$$\frac{\partial J(t)}{\partial y_k(t)} = -e_k(t) \quad \ldots\ldots\ldots\ldots (13)$$

Finally, using the final derivatives of equation (11), (12) and (13) in (7), and rearranging terms, we obtain

$$\frac{\partial J(t)}{\partial \varepsilon_{kj}(t)} = -\phi_k' (x_k(t)).e_k(t).y_j(t). \left[-\frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)} \right] \quad \ldots\ldots\ldots\ldots (14)$$

From the delta rule [24], the local gradient j(t) can be defined by

, $[e_k(t).\phi'_k(x_k(t))y_k(t)]$, which represents the local gradient k(t) at time instance t. Using this relation we may simply redefine equation (14) as

$$\frac{\partial J(t)}{\partial \varepsilon_{kj}(t)} = -\left[\frac{\partial J(t)}{\partial \theta_{kj}(t)}\right] \cdot \left[\frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)}\right]$$

$$= \delta_k(t).\delta_k(t-1) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (15)$$

Which defines the derivative of the error surface with respect to the learning rate parameter , ε, for output layer nodes. In case of hidden nodes, the same formula can be obtained simply by using local gradient equation

$$\delta_j(t) = \phi'_j(x_j(t)) \sum_k^m \delta_k(t)\ \theta_{kj}(t)$$

in (14). In other words equation (15) may be applied to all modified nodes in the net structure. Therefore, from equation (15), the changes in the learning rate parameter, ε, should be proportional to the current and last changes of gradient. Hence, we may define the adjustment applied to learning rate parameter, $\varepsilon_{kj}(t)$, as

$$\varepsilon_{kj}(t) = \varepsilon_{kj}(t-1) + \Delta\varepsilon_{kj}(t) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (16)$$

Where

$$\Delta_{kj}(t) = -\gamma \frac{\partial J(t)}{\partial \varepsilon_{kj}(t)}$$

$$= -\gamma \left[\frac{\partial J(t)}{\partial \theta_{kj}(t)}\right] \cdot \left[\frac{\partial J(t-1)}{\partial \theta_{kj}(t-1)}\right] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (17)$$

$$= -\gamma\ .\ \delta_k(t)\ .\ \delta_k(t-1)$$

Where γ is the proportionality constant for the learning rate procedure, which is usually a positive scalar.

Use (17) in (16) and substituting in (2) yields the general form

$$\Delta\theta(t) = -\left[\varepsilon(t-1) + \gamma\left[\frac{\partial J(t)}{\partial(t)}\right]\left[\frac{\partial J(t-1)}{\partial\theta(t-1)}\right]\right] \cdot \frac{\partial J(t)}{\partial\theta(t)} \quad \ldots\ldots \quad (18)$$

Accordingly, two important procedures have been reported in [14] which are:

1: If the derivative of the error surface with respect to connection weight $\theta(t)$ has the same algebraic sign for more than one consecutive iteration, the adjustment of the learning rate parameter, $\epsilon(t)$, for the next iteration should be increased by a certain value to speed up learning. The value of the increment is proportional to the nodes local gradient for the last two consecutive iterations, equation (17).

2: On the other hand, if the derivative of the error surface with respect to the connection weight $\theta(t)$ alternates on two consecutive iterations, the adjustment of the learning rate parameter, $\varepsilon(t)$, should correspondingly be decreased by a value proportional to the last two consecutive iterations to slow down the learning step, equation (17).

These two observations satisfy the heuristics mentioned in H3 and H4. Therefore, from equation (17), the learning parameter, ε, may be tuned on-line during connection weights adaptation by using equation (16).

The limitation of this tuning strategy as stated by Jacobs [14] is that, if the derivative of the error surface with respect to a particular connection weight has the same sign but small in magnitude. The positive adjustment applied from equation (17) is very small. Also, if the derivative of the error surface with respect to a particular connection weight has opposite signs and large magnitudes at two consecutive iterations, the negative adjustment for that weight will be very large which may create an overflow to the learning process.

The fore mentioned using the second concept of the new self-tun-

ing back-propagation algorithm as follows could eliminate limitations of the learning rule in equation (16).

$$\Delta\theta(t) = \left(\frac{\partial J}{\partial\theta(t)} \Big/ \left(\frac{\partial J}{\partial\theta(t-1)} - \frac{\partial J}{\partial\theta(t)}\right)\right) . \Delta\theta(t-1) \quad (19)$$

This form of quadratic estimation may be used in two cases:

1: where the current slope is somewhat smaller than the previous one but in the same direction (same sign), the weight will change again in the same direction. The step of this change may be large or small depending on how much the slope was reduced by the previous step.

2: when the current slope is in the opposite direction from the previous one, which means we have crossed over the minimum and that we are now on the opposite side of the valley. In this case, the next step will be somewhere between the current and the previous position.

Here, it is worth concluding that, the use of different computation techniques and a different time-varying learning rate parameter, ε, for each connection weight modifies the back-propagation in a fundamental way. In other words, the connection weights are updated frequently either with the steepest descent search method with the tuned learning rate parameter, ε, using equations (18), or with the quadratic estimation method using equation (19).

A summary of the new self-tuning back-propagation algorithm is presented in the following:

1: Initialise the learning parameters which consist of initial weights, θ, initial learning rate parameter, ε, learning momentum, and the learning jump factor, γ.

2: Take the first training sample vector, which consists of the input and the desired system output.

3: Forward input data through all nodes in the hidden layer/layers,

and calculate the actual output, which is

$$y_j(t) = \phi_j(x_j(t)) \quad \text{..... (20)}$$

4: Calculate the global error using equation (6), then propagate the global error back to calculate the local gradient for each output and hidden nodes by using equations (3) and (4), respectively.

5: Depending on the error surface, adjust the connection weights according to equations (18) or (19) using equation (1).

6: Repeat steps 1 to 5 until the end.

The overall flow chart of the new algorithm is illustrated in figures-2 and 3, below.

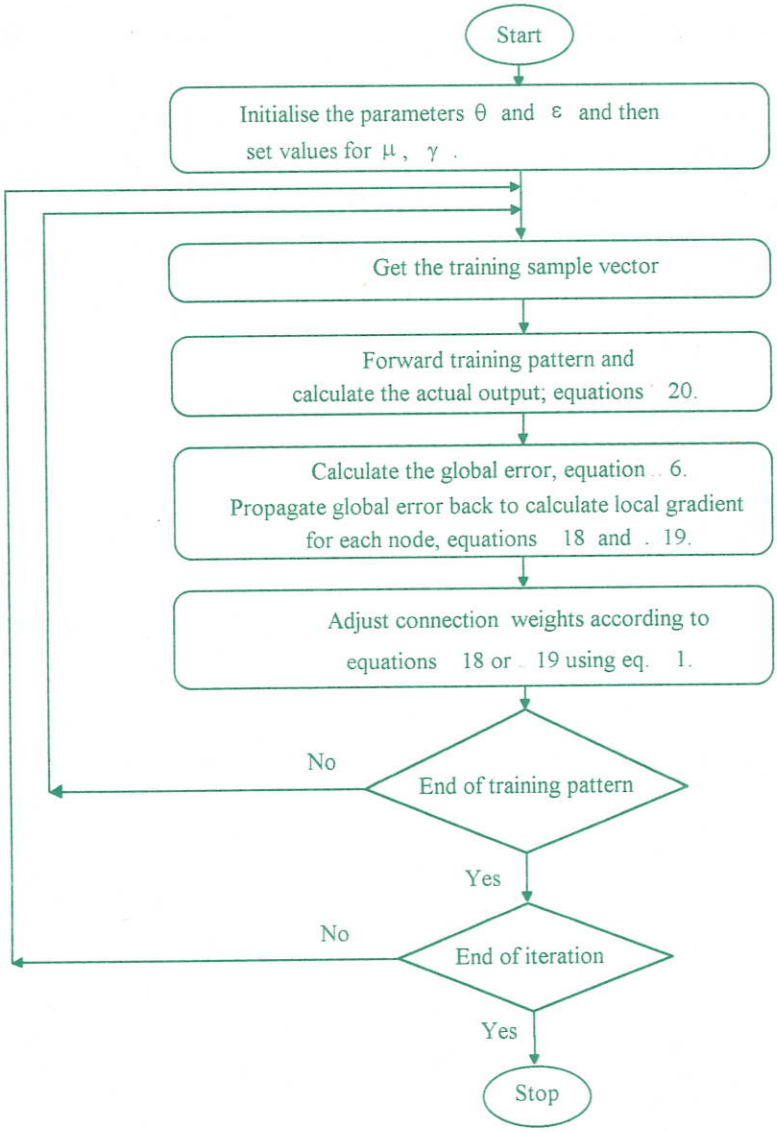
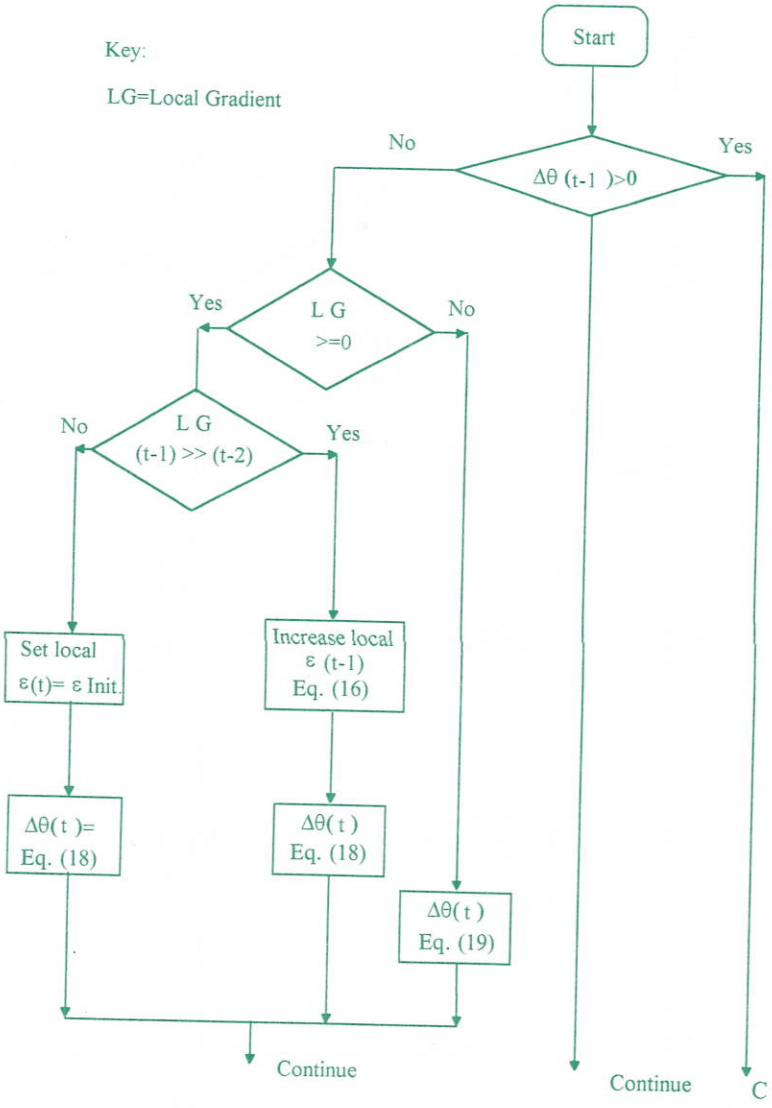
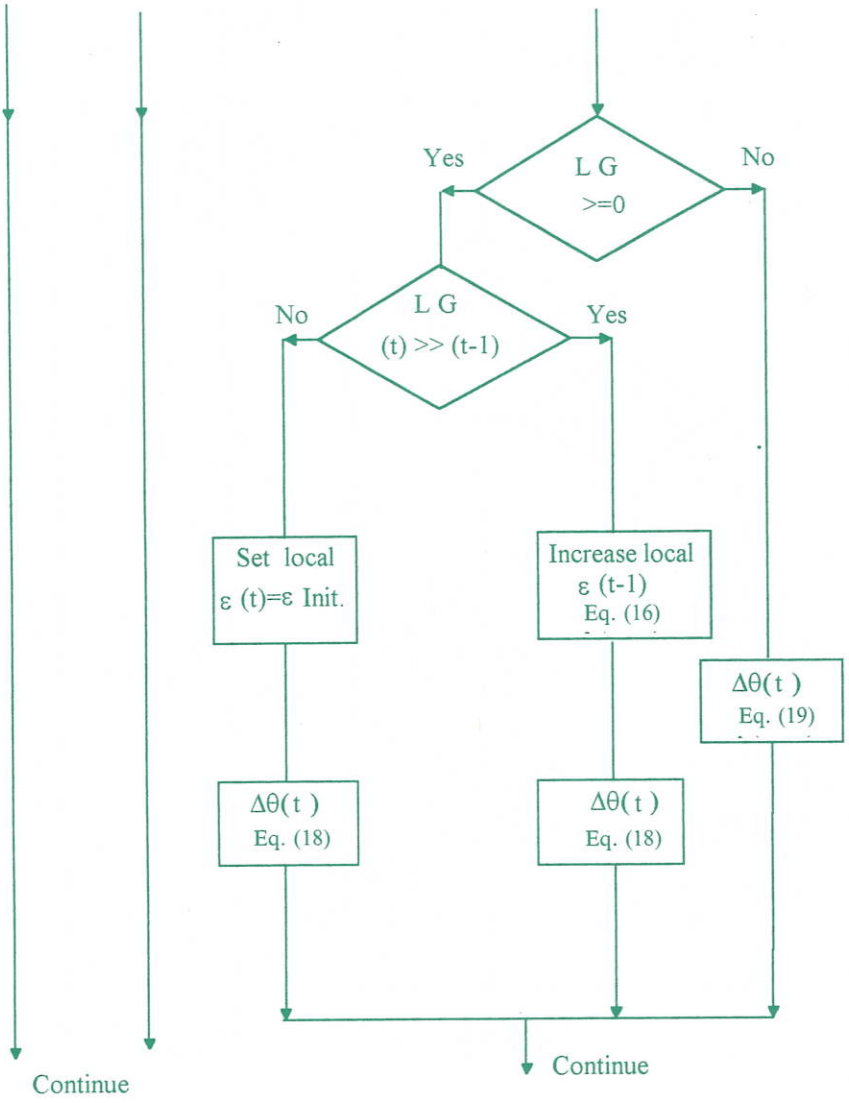


Figure 2 Flowchart for the new self-tuning back-propagation. (SELF - PROP)


Key:
LG=Local Gradient
Start
Δθ (t-1)>0
No
Yes
L G
>=0
Yes
No
L G
(t-1) >> (t-2)
No
Yes
Set local
ε(t)= ε Init.
Increase local
ε (t-1)
Eq. (16)
Δθ(t)=
Eq. (18)
Δθ(t)
Eq. (18)
Δθ(t)
Eq. (19)
Continue
Continue
C


Yes
L G
>=0
No
No
L G
(t) >> (t-1)
Yes
Set local
ε (t)=ε Init.
Increase local
ε (t-1)
Eq. (16)
Δθ(t)
Eq. (19)
Δθ(t)
Eq. (18)
Δθ(t)
Eq. (18)
Continue
Continue

Cont.

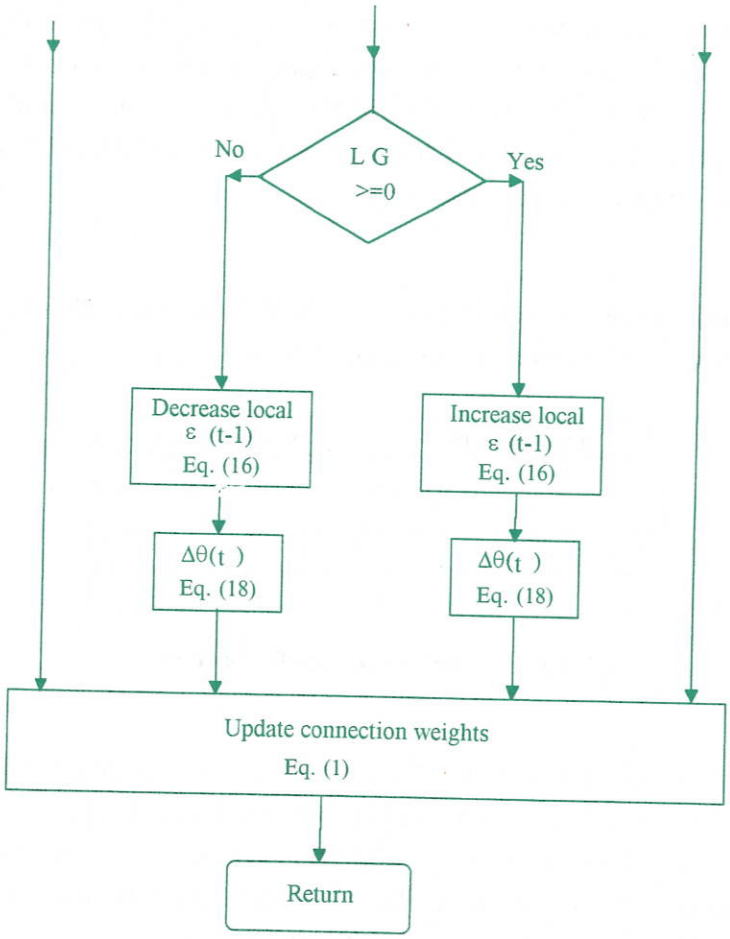


Figure 3 Flowchart illustrating the updating strategy for the connection weights in the new back-propagation algorithm. (SELF - PROP)

6) PERFORMANCE STUDIES ON THE EFFECTS OF TUNING PARAMETERS ON THE LEARNING CURVE OF SELF-PROP

The task undertaken is to compare by a simulation study how the new tuned learning rate back-propagation rule is better than the well known back-propagation algorithms. The comparative study, has been made with the standard back-propagation [24] and the quick-prop back-propagation [5], algorithms.

Consider the classic Exclusive OR (XOR) problem, table 1, which is commonly used in back-propagation function testing.

Input Patterns	Output Patterns
0 0	0
0 1	1
1 0	1
1 1	0

Table 1 Exclusive-OR (XOR) function.

The neural network model selected for this non-linear binary function was NN (2-2-1), which is of two input, two hidden, and one-output nodes. The output layer node function is linear. However, to solve this binary problem with numerical data structure, the zeros and ones in table 1, were set to -0.5 and 0.5 respectively.

Figure 4 depicts the first 500 trials of the comparative results obtained from the standard back-propagation (SBP), the quick-prop back-propagation (QPBP) and the new self-tuning back-propagation (SELF-PROP), algorithms. For all the above learning algorithms, the learning parameters were set as follows: the learning rate parameter $\varepsilon = 0.02$, the momentum learning factor $\mu = 0.5$ and the initial-weights $\rho = 1.0$.

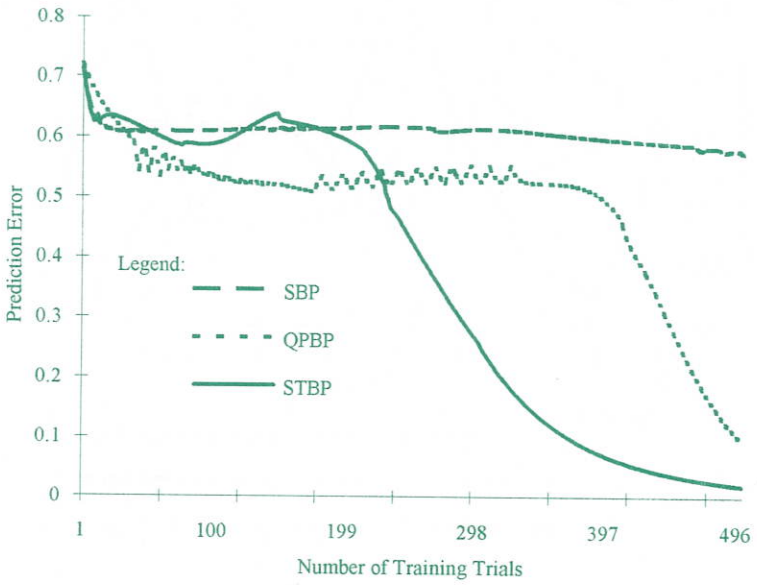


Figure 4, Comparative study between the learning curve of SBP, QPBP, and STBP. (SELF-PROP)

It is very clear, from figure 4 that the new SELF-PROB algorithm converges much faster when compared to SBP and QPBP.

In the above XOR problem the learning rate parameter ε, was automatically tuned in the SELF-PROB according to the tuning strategy in equation (16). The first 10 trials of the learning rate variation for each node of the NN (2-2-1) neural model is shown in figure 5.

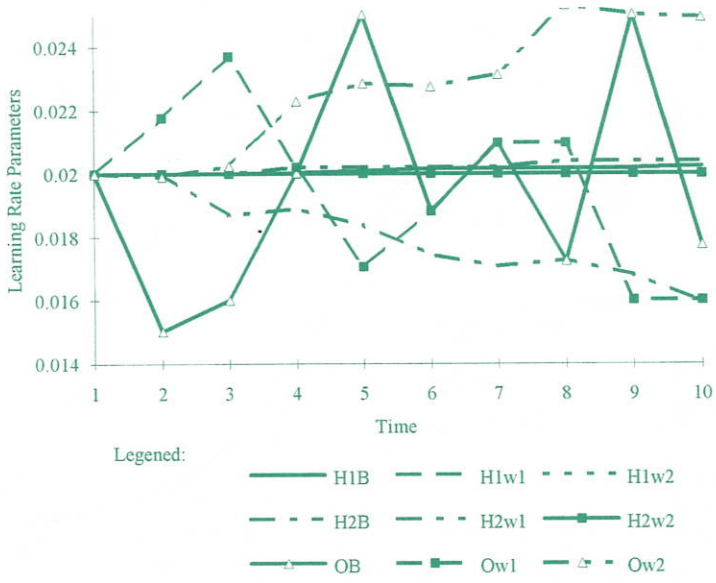


Figure 5: Pictorial representation of the variations of the learning rate parameter, ε.

The annotations for the learning rate parameter, ε, on figure 5 are briefly explained as follows: H1B, H1w1 and H1w2 are the variations of ε for node 1 in the hidden layer, respectively. H2B, H2w1 and H2w2 are the variations of ε in hidden node 2. OB, Ow1 and Ow2 are the variations of ϵ in output layer.

To illustrate the effects of the jump factor, γ, on the convergence speed, small changes were made to γ, between 0.009 to 0.04. The objective was to record the number of trials before convergence for each jump factor, γ. The algorithm was started with $\gamma = 0.009$ and the learning parameters were set as in table 2.

N(2-2-1) XOR			
Initial Value ε	**Jump factor** (γ)	**Initial weights** (ρ)	**Trials before convergence**
0.02	0.009	1.0	800
0.02	0.01	1.0	750
0.02	0.015	1.0	648
0.02	0.02	1.0	577
0.02	0.025	1.0	533
0.02	0.03	1.0	490
0.02	0.035	1.0	467
0.02	0.04	1.0	444

Table 2 Effects of jump factor γ on convergence rate.

The results obtained in table 2 may be best understood if it is pictorially represented as in figure 6

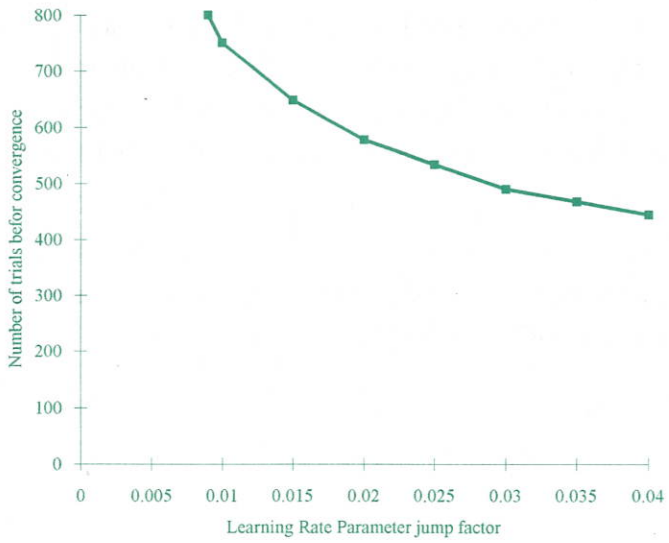


Figure 6: Effects of incrementing the learning rate parameter factor (γ) on system convergence.

This may conclude that, by increasing the learning rate jump factor the convergence speed is effectively improved, i.e. the learning jump factor is directly proportional to convergence speed.

7) CONCLUSION

In real systems identification and control [2, 3, 4, 6, 8, 9], the time available for the estimation of the system parameters in the parametric model is usually small and hence fast convergence algorithms are required. In adaptive control techniques, fast convergence estimator plays a central role for the performance, and the accuracy of a controller especially when fast reference point changes are involved. In non-linear control, although the artificial neural network approach is inherently non-linear in nature and may be a very effective tool for non-linear system identification and control, it has been rejected by plant managers in many real applications due to its slow rate of convergence.

The objective of the work outlined in this paper has been to improve the convergence speed of the back-propagation algorithm through an empirical study, which could cope with the fore, mentioned circumstances. The developed algorithm exhibits good adaptation properties and this with its low complexity makes them ideally suited for practical purposes.

It can be observed from the first test, figure 4, that the new proposed algorithm behaved well compared to some counter part algorithms such as the SBP and QPBP.

REFERENCES

[1] Abdulaziz, A.A., and M. Farsi, (1993), "Non-linear System Identification and Control based on Neural and Self-tuning Control", International Journal of Adaptive Control and Signal Processing, Vol. 7, pp. 297-307.

[2] Abdulaziz, A.A., and M. Farsi, (1993), "Dynamic Modelling and Control for a class of Non-linear Systems using Neural Nets", Conf. Proc. IEEE International Symposium on Industrial Electronics (ISIE' 93), Budapest, Hungary, pp. 543-548.

[3] Abdulaziz, A.A., and M. Farsi, (1993), "A Comparative Study between Self-tuning and Neural-based Controllers", Colloquium on Neural Networks and Fuzzy Logic: in Measurement & Control, Control Systems Research Group, John Mores University, Liverpool, UK.

[4] Abdulaziz, A.A. and M. Farsi, (1994), "An On-line Neural-Based Predictive Controller", ICSE, 10th International Conference On System Engineering, Coventry, UK, Vol. 1, pp. 9-16.

[5] Fahlman, S.E., (1988), "An Empirical Study of Learning Speed in Back-Propagation Networks", CMU Technical Report, CMU-CS-88-162.

[6] Farsi, M., and A.A. Abdulaziz, (1994), "Self-tuning Controller of Bi-linear Systems", The Third IEEE Conference On Control Applications, Glasgow, UK. Vol. 3, pp. 417-422.

[7] Fukuda, T., and T. Shibata, (1992), "Theory and Applications of Neural Networks for Industrial Control Systems", IEEE Trans.

on Ind. Electronics, Vol. 39, No. 6, pp. 472-489.

[8] Funahashi, K.I., (1989), "On the Approximate Realisation of Continuous Mappings by Neural Networks", Neural Networks, vol. 2, pp. 183-192.

[9] Grant, P.M., (1989), "Artificial Neural Networks and Conventional Approaches to Filtering and Pattern Recognition", IEE J. Elect and Comms. Eng., September, pp. 225-232.

[10] Haykin, S., (1994), "Neural Networks: A Comprehensive Foundation", Macmillan College Publishing Company.

[11] Hebb, D.O., (1949), "The Organisation of Behaviour: A Neuropsychological Theory", Wiley: New York.

[12] Hecht-Nielsen, R., (1989), "Theory of back propagation neural networks", Proc. IJCNN, Washington DC, pp. 593-608.

[13] Irwin, G.W, G. Lightbody and S. McLoone, (1994), "Comparison of Training Algorithms for Multi-layer Perceptrons", IEE Coll. on Advances in Neural Networks for Control and Systems, Digest No. 136, Berlin, pp. 11/1-11/6.

[14] Jacobs, R.A., (1988), "Increased Rates of Convergence Through Learning Rate Adaptation", Neural Networks, Vol. 1, pp. 295-307.

[15] McLoone, S. and G. Irwin, (1995), "Fast Gradient Based Off-Line Training of Multi-layer Perceptrons", To appear in 'A Chanced in Neural Networks for Control Systems', (Hunt, Irwin and Warwick, eds.), Sperings-Vealag.

[16] Miller, W.T., F.H. Glanz and L.G. Kraft, (1990), "CAMAC: An Associative Neural Network Alternative to Back-Propagation" Proc. IEEE, Vol. 78, No. 10, pp. 1561-1567.

[17] Moody, J.E and C.J. Darken, (1989), "Fast Learning in Networks of Locally-Tuned Processing Units", Neural Computation 1, pp. 281-294.

[18] Poggio, T,. and F. Girsi, (1990), "Networks for Approximation

and Learning", Proc. IEEE, Vol. 78, No. 9, pp. 1481-1497.

[19] Poggio, T. and F Girosi, (1990), "Regularization Algorithms for Learning that are be equivalent to Multi-layer Networks", Science 247, pp. 978-982.

[20] Powell, M.J., (1985), "Radial Basis functions for Multivariable Interpolation: A review", In IMA Conference on Algorithms for the Applications if Functions and Data., pp. 143-167, RMCS, Shrivenham, UK.

[21] Renals, S., (1989), "Radial Basis Function Networks for Speech Pattern Classification", Electronic Letters 25, pp. 437-439.

[22] Robbins, H. and S. Moroe, (1951), "A Stochastic Approximation Method", Annals of Mathematical Statistics, No. 22, pp. 400-407.

[23] Rosenblatt, F., (1958), "The Perceptron: A probabilistic model for information storage and organisation in the brain", Psychological Rev., vol. 65, pp. 368-408.

[24] Rumelhart, D.E., G.E. Hinton, and R. J. Williams, (1986), "Parallel Distributed Processing", MA, Cambridge: The MIT press.

[25] Saarinen, S., R.B. Bramley and G. Cybenko, (1992), "Neural Networks, Back-Propagation, and Automatic Differentiation", In Automatic Differentiation of Algorithms: Theory, Implementation, and Application, (A. Griewank and G.F Corliss, eds.), pp. 31-42, Philadelphia, PA: SIAM.

[26] Samad, T., (1990), "Back-propagation Improvements Based on Heuristic Arguments, IJCNN-90, Wash-DC, Vol. 1, pp. 565-568.

[27] Sutton, R.S., (1986), "Two Problems with Back-Propagation and other Steepest Descent Learning Procedures for Networks" Proc. of the 8th Annual Conference of the Cognitive Science Society, pp. 823-831.

[28] Werbos, P.J., (1974), "Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences", Ph.D. Dissertation, Harvard University, MA, Cambridge.

[29] Werbos, P.J. , (1990), "Neurocontrol and Related Techniques", Handbook of Neural Computing Applications, by Maren A. J., C. T. Harston and R.M. Pap, Academic Press, Inc.

[30] Widrow, B., and M.A. Lehr, (1990), "Thirty years of adaptive neural networks: Perceptron, Madaline, and Back-Propagation", Proc. IEEE, Vol. 78, No. 9, pp. 1415-1441.

[31] Wiener, N., (1958), "Non-linear Problems in Random Theory", J. Wiley, New York.

[32] Willis, M.j., C.D. Massimo, G.A. Montague, M.T. Tham and A.J. Morris, (1991), "Artificial Neural Networks in Process Engineering", IEE Proceedings-D, Vol. 138, No. 3, pp. 256-266.

[33] Ogata, K., (1990), "Modern Control Engineering", Prentice-Hall, Inc.