

A Neural Network Training Algorithm Utilizing Multiple Sets of Linear Equations

Hung-Han Chen ^a, Michael T. Manry ^b, and Hema Chandrasekaran ^b

^a CYTEL Systems, Inc., Hudson, MA 01749

^b Department of Electrical Engineering

University of Texas at Arlington, Arlington, TX 76019

Phone: (817) 272-3483

FAX: (817) 272-2253

^a e-mail: hungchen@geocities.com

^b e-mail: manry@uta.edu

ABSTRACT

A fast algorithm is presented for the training of multilayer perceptron neural networks, which uses separate error functions for each hidden unit and solves multiple sets of linear equations. The algorithm builds upon two previously described techniques. In each training iteration, output weight optimization (OWO) solves linear equations to optimize output weights, which are those connecting to output layer net functions. The method of hidden weight optimization (HWO) develops desired hidden unit net signals from delta functions. The resulting hidden unit error functions are minimized with respect to hidden weights, which are those feeding into hidden unit net functions. An algorithm is described for calculating the learning factor for hidden weights. We show that the combined technique, OWO-HWO is superior in terms of convergence to standard OWO-BP (output weight optimization-backpropagation) which uses OWO to update output weights and backpropagation to update hidden weights. We also show that the OWO-HWO algorithm usually converges to about the same training error as the Levenberg-Marquardt algorithm in an order of magnitude less time.

KEYWORDS : Multilayer perceptron, Fast training, Hidden weight optimization, Second-order methods, Conjugate gradient method, Levenberg-Marquardt algorithm, Learning factor calculation, Backpropagation, Output weight optimization.

1. Introduction

Multilayer perceptron (MLP) neural networks have been widely applied in the fields of pattern recognition, signal processing, and remote sensing. However, a critical problem has been the long training time required. Several investigators have devised fast training techniques that require the solution of sets of linear equations [3,5,18,21,24,26]. In output weight optimization-backpropagation [18] (OWO-BP), linear equations are solved to find output weights and backpropagation is used to find hidden weights (those which feed into the hidden units). Unfortunately, backpropagation is not a very effective method for updating hidden weights [15,29]. Some researchers [11,16,17,20,31] have used the Levenberg-Marquardt(LM) method to train the multilayer perceptron. While this method has better convergence properties [4] than the conventional backpropagation method, it requires $O(N^2)$ storage and calculations of order $O(N^2)$ where N is the total number of weights in an MLP[19]. Hence training an MLP using the LM method is impractical for all but small networks.

Scalero and Tepedelenlioglu [27] have developed a non-batching approach for finding all MLP weights by minimizing separate error functions for each hidden unit. Although their technique is more effective than backpropagation, it does not use OWO to optimally find the output weights, and does not use full batching. Therefore, its convergence is unproved. In our approach we have adapted their idea of minimizing a

separate error function for each hidden unit to find the hidden weights and have termed this technique hidden weight optimization (HWO).

In this paper, we develop and analyze a training algorithm which uses HWO. In section 2, we review the OWO-BP algorithm. Methods for calculating output weight changes, hidden weight changes, and the learning factor are described. In section 3, we develop the full-batching version of HWO and recalculate the learning factor. The convergence of the new algorithm is shown. The resulting algorithm, termed OWO-HWO, is compared to backpropagation and OWO-BP in section 4. Simulation results and conclusions are given in sections 5 and 6, respectively.

2. Review of Output Weight Optimization-Backpropagation

In this section, we describe the notation and error functions of a MLP network followed by the review of the output weight optimization - backpropagation (OWO-BP) algorithm [18]. The OWO-BP technique iteratively solves linear equations for output weights and uses backpropagation with full batching to change hidden weights.

2.1 Notation and Error Functions

We are given a set of N_v training patterns $\{(\mathbf{x}_p, \mathbf{T}_p)\}$ where the p th input vector \mathbf{x}_p and the p th desired output vector \mathbf{T}_p have dimensions N and N_{out} , respectively. The activation $O_p(n)$ of the n th input unit for training pattern p is

$$O_p(n) = x_p(n) \quad (2.1)$$

where $x_p(n)$ denotes the n th element of \mathbf{x}_p . If the j th unit is a hidden unit, the net input $net_p(j)$ and the output activation $O_p(j)$ for the p th training pattern are

$$\begin{aligned} net_p(j) &= \sum_i w(j,i) \cdot O_p(i) , \\ O_p(j) &= f(net_p(j)) \end{aligned} \quad (2.2)$$

where the i th unit is in any previous layer and $w(j,i)$ denotes the weight connecting the i th unit to the j th unit. If the activation function f is sigmoidal, then

$$f(net_p(j)) = \frac{1}{1 + e^{-net_p(j)}} \quad (2.3)$$

Net function thresholds are handled by adding an extra input, $O_p(N+1)$, which is always equal to one.

For the k th output unit, the net input $net_{op}(k)$ and the output activation $O_{op}(k)$ for the p th training pattern are

$$\begin{aligned} net_{op}(k) &= \sum_i w_o(k,i) \cdot O_p(i) , \\ O_{op}(k) &= net_{op}(k) \end{aligned} \quad (2.4)$$

where $w_o(k,i)$ denotes the *output* weight connecting the i th unit to the k th output unit.

The mapping error for p th pattern is

$$E_p = \sum_{k=1}^{N_{out}} [T_p(k) - O_{op}(k)]^2 \quad (2.5)$$

where $T_p(k)$ denote the k th element of the p th desired output vector. In order to train a neural network in *batch* mode, the mapping error for the k th output unit is defined as

$$E(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} [T_p(k) - O_{op}(k)]^2 \quad (2.6)$$

The overall performance of a MLP neural network, measured as *Mean Square Error* (MSE), can be written as

$$E = \sum_{k=1}^{N_{out}} E(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} E_p \quad (2.7)$$

2.2 Output Weight Changes

Some researchers [3,5,18,21,24,26] have investigated fast training techniques to find weights of neural networks by solving a set of linear equations. With the linearity property of the output units, as in most cases, the output weights are more likely to form a set of linear equations. The Output Weight Optimization (OWO) learning algorithm [18] has been successfully used to minimize MSE via solving output weights linear equations.

Taking the gradient of $E(k)$ with respect to the output weights, we get

$$g(m) \equiv \frac{\partial E(k)}{\partial w_o(k,m)} = -2 \left(R_{TO}(m) - \sum_i w_o(k,i) R_{OO}(i,m) \right) \quad (2.8)$$

where

$$R_{TO}(m) = \sum_{p=1}^{N_v} T_p(k) O_p(m) \quad (2.9)$$

$$R_{OO}(i,m) = \sum_{p=1}^{N_v} O_p(i) O_p(m) \quad (2.10)$$

Setting $g(m)$ to zero, we get

$$\sum_{i=1} w_o(k,i) R_{OO}(i,m) = R_{TO}(m) \quad (2.11)$$

Methods for solving these linear equations include *Gaussian elimination* (GE), *Singular value decomposition* (SVD), and *LU decomposition* (LUD) [23]. However, it is also possible to use the conjugate gradient approach [6,7] to minimize $E(k)$ [10,14].

2.3 Hidden Weight Changes in Backpropagation Method

Backpropagation is a popular method for updating the hidden weights. The conceptual basis of backpropagation was introduced by Werbos [28], then independently reinvented by Parker [22], and widely presented by Rumelhart and McClelland [25]. In standard backpropagation, the hidden weights are updated as

$$w(j,i) \leftarrow w(j,i) + Z \cdot \frac{-\bar{\partial}E_p}{\partial w(j,i)} \quad (2.12)$$

where Z is a constant learning factor. By using the chain rule, the gradients can be expressed as

$$\frac{\bar{\partial}E_p}{\partial w(j,i)} = -\delta_p(j) \cdot O_p(i) \quad (2.13)$$

where

$$\delta_p(j) = \frac{-\bar{\partial}E_p}{\partial net_p(j)} \quad (2.14)$$

is called *delta* function. The calculations of the *delta* functions for output units and hidden units are respectively [25]

$$\begin{aligned} \delta_p(j) &= f'(net_j) \cdot (T_p(j) - O_p(j)), \\ \delta_p(j) &= f'(net_j) \sum_n \delta_p(n) w(n, j) \end{aligned} \quad (2.15)$$

where n is the index of units in the following layers which are connected to the j th unit

and f' is the first derivative of the activation function.

The performance of standard backpropagation sometimes is restricted by the order of training patterns since it changes weights for each pattern. In order to reduce the importance of the order of training patterns, batch mode backpropagation has often been used to improve the performance of the standard backpropagation. That is, accumulate weight changes for all the training patterns before changing weights. With full batching, the hidden weight changes are calculated as

$$w(j,i) \leftarrow w(j,i) + Z \cdot \frac{-\delta E}{\partial w(j,i)} \quad (2.16)$$

and,

$$\frac{\partial E}{\partial w(j,i)} = \sum_{p=1}^{N_p} \frac{\delta E_p}{\partial w(j,i)} \quad (2.17)$$

2.4 Learning Factor Calculation

One problem with using BP in this manner is that the proper value of the learning factor is difficult to determine. If the gradient vector has large energy, we may need to use a small learning factor to prevent the error function E from blowing up. This intuitive idea can be developed as follows.

Assume that a learning factor Z is small so that the error surface is well approximated by a hyperplane. Then the change in E due to the change in $w(j,i)$ in equation (2.16) is approximately [18]

$$\Delta E = \frac{\partial E}{\partial w(j,i)} \cdot \Delta w(j,i) = -Z \cdot \left(\frac{\partial E}{\partial w(j,i)} \right)^2 \quad (2.18)$$

Assume that we want to calculate Z so that the error function E is reduced by a factor α which is close to, but less than, 1. We then get

$$\Delta E = \alpha E - E = -Z \sum_j \left[\sum_i \left(\frac{\partial E}{\partial w(j,i)} \right)^2 \right] \quad (2.19)$$

and

$$Z = \frac{Z'E}{\sum_j \left[\sum_i \left(\frac{\partial E}{\partial w(j,i)} \right)^2 \right]} \quad (2.20)$$

$$Z' = (1 - \alpha) \quad (2.21)$$

Using these equations, the learning factor Z is automatically determined from the gradient and Z' , where Z' is a number between 0.0 and 0.1. The learning factor Z in (2.20) is then used in (2.16).

3. OWO-HWO Training Algorithm

In this section, we first describe hidden weight optimization (HWO) which is a full-batching version of the training algorithm of [27], restricted to hidden units. The learning factor for hidden weights is derived. OWO-HWO is then presented followed by a discussion of its convergence.

3.1 Hidden Weight Changes

It is desirable to optimize the hidden weights by minimizing separate error

functions for each hidden unit. By minimizing many simple error functions instead of one large one, it is hoped that the training speed and convergence can be improved. However, this requires desired hidden net functions, which are not normally available. The desired net function can be approximated by the current net function plus a designed net change. That is, for j th unit and p th pattern, a desired net function [27] can be constructed as

$$net_{pd}(j) \cong net_p(j) + Z \cdot \delta_p(j) \quad (3.1)$$

where $net_{pd}(j)$ is the desired net function and $net_p(j)$ is the actual net function for j th unit and the p th pattern. Z is the learning factor and $\delta_p(j)$ is the delta function from (2.15).

Similarly, the hidden weights can be updated as

$$w(j,i) \leftarrow w(j,i) + Z \cdot e(j,i) \quad (3.2)$$

where $e(j,i)$ is the weight change and serves the same purpose as the negative gradient element, $-\partial E/\partial w(j,i)$, in backpropagation. With the basic operations of (2.1~2.4), and (3.1~3.2), we can use the following equation to solve for the changes in the hidden weights,

$$net_p(j) + Z \cdot \delta_p(j) \cong \sum_i [w(j,i) + Z \cdot e(j,i)] \cdot O_p(i) \quad (3.3)$$

Deleting the current net function and eliminating the learning factor Z from both side:

$$\delta_p(j) \cong \sum_i e(j,i) \cdot O_p(i) \quad (3.4)$$

Before solving (3.4) in the least squares sense, we define an objective function for the j th unit as

$$E_\delta(j) = \sum_{p=1}^{N_p} \left[\delta_p(j) - \sum_i e(j,i) O_p(i) \right]^2 \quad (3.5)$$

Then, taking the gradient of $E_\delta(j)$ with respect to the weight changes, we get

$$g(m) \equiv \frac{\partial E_\delta(j)}{\partial e(j,m)} = -2 \left(R_{\delta o}(m) - \sum_i e(j,i) R_{oo}(i,m) \right) \quad (3.6)$$

where

$$\begin{aligned} R_{\delta o}(m) &= \sum_{p=1}^{N_v} \delta_p(j) O_p(m) \\ &= \frac{-\partial E}{\partial w(j,m)} \end{aligned} \quad (3.7)$$

$$R_{oo}(i,m) = \sum_{p=1}^{N_v} O_p(i) O_p(m) \quad (3.8)$$

Setting $g(m)$ to zero generates the linear equations,

$$\sum_i e(j,i) R_{oo}(i,m) = \frac{-\partial E}{\partial w(j,m)} \quad (3.9)$$

These equations are solved unit by unit as for the desired weight changes $e(j,i)$. We update the hidden weights as in (3.2)

3.2 Learning Factor Calculation

Based upon the method of learning factor calculation, discussed in section 2.4, we can reconstruct it to suit our need. From (3.10), the actual weight changes are defined as

$$\Delta w(j,i) = Z \cdot e(j,i) \quad (3.11)$$

Then the change in E due to the change in $w(j,i)$ is approximately

$$\Delta E \cong \frac{\partial E}{\partial w(j,i)} \cdot \Delta w(j,i) = Z \cdot \frac{\partial E}{\partial w(j,i)} \cdot e(j,i) \quad (3.12)$$

Assume that we want to calculate Z so that the error function E is reduced by a factor α which is close to, but less than, 1. We then get

$$\Delta E = \alpha E - E \cong Z \sum_j \left[\sum_i \frac{\partial E}{\partial w(j,i)} \cdot e(j,i) \right] \quad (3.13)$$

and we can set

$$Z = \frac{-Z'E}{\sum_j \left[\sum_i \frac{\partial E}{\partial w(j,i)} \cdot e(j,i) \right]} \quad (3.14)$$

$$Z' = (1 - \alpha) \quad (3.15)$$

Using these equations, the learning factor Z is automatically determined from the gradient elements, weight changes solved from linear equations, and Z' , where Z' is a number between 0.0 and 0.1.

3.3 New Algorithm Description

Replacing the BP component of OWO-BP by HWO, we construct the following algorithm,

- (1) Initialize all weights and thresholds as small random numbers in the usual manner.
Pick a value for the maximum number of iterations, N_{it} . Set the iteration (epoch) number i_t to 0.

- (2) Increment i_t by 1. Stop if $i_t > N_{it}$.
- (3) Pass the training data through the network. For each input vector, calculate the hidden unit outputs $O_p(i)$ and accumulate the cross- and auto-correlation $R_{TO}(m)$ and $R_{OO}(i,m)$.
- (4) Solve linear equations (2.11) for the output weights and calculate E .
- (5) If E increases, reduce the value of Z , reload the previous best hidden weights and go to Step 9.
- (6) Make a second pass through the training data. Accumulate the gradient elements $\partial E / \partial w(j,m)$, as the cross-correlation $R_{sO}(m)$, and accumulate the auto-correlation $R_{OO}(i,m)$ for hidden units.
- (7) Solve linear equations (3.9) for hidden weight changes.
- (8) Calculate the learning factor Z .
- (9) Update the hidden unit weights.
- (10) Go to Step 2.

3.4 Algorithm Convergence

To show that the new algorithm converges, we make use of the learning factor calculations. In a given iteration, the change in E for the j th unit, which is a hidden unit, is approximated as

$$\begin{aligned}
\Delta E(j) &\cong \sum_i \frac{\partial E}{\partial w(j,i)} \cdot \Delta w(j,i) = Z \cdot \sum_i \frac{\partial E}{\partial w(j,i)} \cdot e(j,i) \\
&= Z \cdot \sum_i \frac{1}{N_v} \sum_{p=1}^{N_v} \frac{\partial E_p}{\partial w(j,i)} \cdot e(j,i) \\
&= Z \cdot \sum_i \frac{1}{N_v} \sum_{p=1}^{N_v} -\delta_p(j) \cdot O_p(i) \cdot e(j,i) \\
&= -Z \cdot \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p(j) \sum_i O_p(i) \cdot e(j,i) \cong -Z \cdot \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p^2(j)
\end{aligned} \tag{3.16}$$

The total change in the error function E , due to changes in all hidden weights, becomes approximately

$$\Delta E \cong -Z \frac{1}{N_v} \sum_j \sum_{p=1}^{N_v} \delta_p^2(j) \tag{3.17}$$

First consider the case where the learning factor Z is positive and small enough to make (3.16) valid. Let E_k denotes the training error in the k th iteration. Since the ΔE sequence is nonpositive, the E_k sequence is nonincreasing. Since nonincreasing sequences of nonnegative real numbers converge, E_k converges.

When the error surface is highly curved, the approximation of (3.16) may be invalid in some iterations, resulting in increases in E_k . In such a case, step (5) of the algorithm reduces Z and reloads the best weights before trying step (9) again. This sequence of events need only be repeated a finite number of times before E_k is again decreasing, since the error surface is continuous. After removing parts of the E_k sequence which are increasing, we again have convergence.

The convergence of the global error function E is interesting, considering the fact that individual hidden unit error functions are reduced. As with other learning algorithms, it should be noted that OWO-HWO is not guaranteed to find a global minimum.

4. Comparison with Gradient Approach

4.1 Efficient Implementation

From (3.9), the relationship between the hidden weight changes of backpropagation and those of hidden weight optimization algorithm is a linear transformation through an autocorrelation matrix R_{OO} . That is

$$R_{OO} \cdot \mathbf{e}_{hwo} = \mathbf{e}_{bp} \quad (4.1)$$

where \mathbf{e}_{bp} denotes the vector of hidden weight changes obtained from backpropagation for a given hidden unit, and \mathbf{e}_{hwo} denotes the hidden weight change vector from the new HWO algorithm.

There are at least two methods for solving (4.1) for the hidden weight changes. If the conjugate gradient method [18] is used in MLP training, the number of multiplications N_{m1} for finding hidden weight changes of a hidden unit in one iteration is approximated as

$$N_{m1} \cong x_1 \cdot (2n^3 + 5n^2 + 2n) + 2(n^2 + n) \quad (4.2)$$

where n is the number of total inputs for the given hidden unit and x_1 is the number of iterations in the conjugate gradient method. Typically, the value of x_1 is 2. When used in a 3-layer MLP network, the number of extra multiplications M_1 , compared with OWO-BP, for solving hidden weights during training becomes

$$M_1 \cong N_{it} \cdot N_{hu} \cdot N_{m1} \quad (4.3)$$

where N_{it} is the number of iterations in MLP training and N_{hu} is the number of hidden units. Note that the needed multiplications for finding R_{OO} are not counted in (4.3).

It is also possible to invert R_{OO} in (4.1) and solve for \mathbf{e}_{hwo} as

$$\mathbf{e}_{hwo} = R_{OO}^{-1} \cdot \mathbf{e}_{bp} \quad (4.4)$$

The advantage of this method is that the matrix inverse operation is needed only a few times during training. For example, this operation is needed only once for the units in the first hidden layer. For a unit in the second hidden layer, this inversion is needed only once in each iteration rather than once per hidden unit.

To solve for the hidden weight changes with this second method, the number of multiplications N_{m2} for finding hidden weight changes of a hidden unit is approximately is

$$N_{m2} \cong x_2 \cdot (6n^3 + 11n^2 - 4n) + 9n^3 + 7n^2 - 3n \quad (4.5)$$

where x_2 is the maximum allowed number of iterations in the SVD. Typically, the value of x_2 is 30. When used in a 3-layer MLP network, the number of extra multiplications M_2 for solving hidden weights during training becomes

$$M_2 \cong N_{m2} + N_{it} \cdot N_{hu} \cdot n^2 \quad (4.6)$$

We can compare M_1 and M_2 with an example of having 19 inputs in each pattern. Therefore n is 20 (19 inputs plus a threshold), x_1 is 2, x_2 is 30, N_{it} is 100, and N_{hu} is 20. By (4.3) and (4.6), M_1 is approximately 73,840,000 and M_2 is approximately 2,444,340 which is 30 times less than M_1 . Clearly, the inversion approach of (4.4) is more efficient.

4.2 Weight Change Relationship

In this subsection we further investigate the relationship between the new HWO algorithm and BP.

From (4.1), the vector of hidden weight changes obtained from the hidden weight optimization algorithm is equal to that from the backpropagation algorithm when the autocorrelation matrix R_{OO} is an identity matrix. This happens when the following conditions are satisfied:

1. Each input feeding into hidden units is *zero mean*.
2. The *variances* of these inputs are all equal to 1.
3. All of the hidden unit inputs are *uncorrelated*.

Note that these conditions are usually not satisfied because:

(1) the threshold input $O_p(N+1)$ is always equal to one so conditions 1 and 2 are not met,
(2) in four layer networks, hidden unit inputs include outputs of the first hidden layer, so condition 3 is not met. However, in a three-layer network with no hidden unit thresholds, the algorithms could become equivalent.

4.3 Transformation of Inputs

In the previous subsection we see that OWO-BP and OWO-HWO are equivalent when their training data are the same and satisfy certain conditions. It is natural to wonder

whether a transformation of the input vectors followed by OWO-BP is equivalent to performing OWO-HWO on the original input. In such a case, the conditions from previous subsection are no longer necessary. Assume that our MLP has only one hidden layer, so that R_{OO} is independent of the iteration number. After the weights have been updated using HWO in (3.10), the net function of each hidden unit for each pattern can be obtained from

$$(\mathbf{w} + Z \cdot \mathbf{e}_{hwo})^T \cdot \mathbf{x}_p = net_p + Z \cdot \Delta net_p \quad (4.7)$$

where \mathbf{w} denotes the input weight vector of a given hidden unit, and net_p and Δnet_p denote the net function and net function change of the hidden unit. Since

$$\mathbf{w}^T \cdot \mathbf{x}_p = net_p \quad (4.8)$$

by the definition of net function, we then can get

$$\mathbf{e}_{hwo}^T \cdot \mathbf{x}_p = \Delta net_p \quad (4.9)$$

With (4.4), equation (4.9) can be rewritten as

$$\left(R_{OO}^{-1} \cdot \mathbf{e}_{bp} \right)^T \cdot \mathbf{x}_p = \Delta net_p \quad (4.10)$$

or

$$\mathbf{e}_{bp}^T \cdot \left(R_{OO}^{-1} \cdot \mathbf{x}_p \right) = \Delta net_p \quad (4.11)$$

The net function change from backpropagation is

$$\mathbf{e}_{bp}^T \cdot \mathbf{x}_p = \Delta net_p \quad (4.12)$$

Comparing (4.11) and (4.12) we can linearly transform our training data and perform OWO-BP, which is equivalent to performing OWO-HWO on the original data. Note that the linear transformation, $R_{OO}^{-1} \cdot \mathbf{x}_p$, is not equivalent to a principal components transform.

The procedure for this method includes:

- (1) Transform training data once.
- (2) Train the MLP network with OWO-BP.
- (3) Absorb transformation into input weights so that new input patterns don't require transformation.

Note that the idea of transforming training data will work as long as the inverse of the auto-correlation matrix can be found. When used in a 3-layer MLP network, the number of extra multiplications M_3 for solving hidden weights becomes

$$M_3 \cong N_{m2} + (N_v + N_{hu}) \cdot n^2 \quad (4.13)$$

We can see that the calculation of M_3 strongly depends on the total number of training patterns, therefore this method may not be as efficient as the matrix inversion approach for using on a large training data set.

5. Performance Comparison

In this section, examples with four mapping data sets are used to illustrate the performance of the new training algorithm. All our simulations were carried out on a Pentium II , 300 Mhz Windows NT workstation using the Microsoft Visual C++ 5.0 compiler. The comparisons between the two-data-pass OWO-BP algorithm, the Levenberg-Marquardt (LM) algorithm [11,16,17,19,20,31] and the new training algorithm (OWO-HWO) are shown in figures 1 through 4.

Example 1: The data set *Power14* obtained from TU Electric Company in Texas has 14 inputs and one output. The first ten input features are the last ten minutes' average

power load in megawatts for the entire TU Electric utility, which covers a large part of north Texas. The output is the forecast power load fifteen minutes from the current time. All powers are originally sampled every fraction of a second, and averaged over 1 minute to reduce noise. The original data file is split into a training file and a testing file by random assignment of patterns. The training file has 1048 patterns and the testing file contains 366 patterns.

We chose the MLP structure 14-10-1 and trained the network for 50 iterations using the OWO-BP, OWO-HWO and LM algorithms. We subsequently tested the trained networks using the testing data file. The results are shown in Figure 1 and Table 5.1. We see that OWO-HWO outperforms both OWO-BP and LM in terms of training error. From the table, we see that OWO-HWO generalizes as well as LM. Because the separation of the training and testing errors is greater for OWO-HWO than for LM, we should be able to use a smaller network for the OWO-HWO case.

Example 2 : The data set *Single2* has 16 inputs and 3 outputs, and represents the training set for inversion of surface permittivity ϵ , the normalized surface rms roughness $k\sigma$, and the surface correlation length kL found in backscattering models from randomly rough dielectric surfaces [12,13]. The first eight of the sixteen inputs represent the simulated backscattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarizations. The remaining eight inputs are various combinations of ratios of the original eight values. These ratios correspond to those used in several empirical retrieval algorithms. The training and testing sets are obtained by random assignment of the original patterns to each of these sets. The training set contains 5992

patterns and the testing set has 4008 patterns.

We chose the MLP structure 16-20-3 and trained the network for 50 iterations of the OWO-HWO and LM algorithms and for 300 iterations of OWO-BP. We then tested the trained networks using the testing data file. The results are shown in figure 2 and Table 5.1. Again we see that OWO-HWO outperforms OWO-BP and LM for both training and testing.

Example 3: The third example is the data set *F17*, which contains 2823 training patterns and 1922 testing patterns for onboard *Flight Load Synthesis* (FLS) in helicopters. In FLS, we estimate mechanical loads on critical parts, using measurements available in the cockpit. The accumulated loads can then be used to determine component retirement times. There are 17 inputs and 9 outputs for each pattern. In this approach, signals available on an aircraft, such as airspeed, control attitudes, accelerations, altitude, and rates of pitch, roll, and yaw, are processed into desired output loads such as fore/aft cyclic boost tub oscillatory axial load (OAL), lateral cyclic boost tube OAL, collective boost tube OAL, main rotor pitch link OAL, etc. This data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995 by Bell Helicopter of Fort Worth.

We chose the MLP structure 17-20-9 and trained the network using OWO-BP for 100 iterations, using OWO-HWO for 300 iterations and then using the LM algorithm for 50 iterations. We then tested the trained networks using the testing data file. The results are shown in figure 3 and Table 5.1. Here we note that OWO-HWO reaches almost the same MSE as LM in an order of magnitude less time and easily outperforms

OWO-BP. We want to mention that the target output values in this data set are large and hence the resulting MSE is large.

Example 4: The data set *Twod* contains simulated data based on models from backscattering measurements. The data set has 8 inputs and 7 outputs, 1768 training patterns and 1000 testing patterns. The inputs consisted of eight theoretical values of the backscattering coefficient parameters σ° at V and H polarizations and four incident angles (10°, 30°, 50°, 70°). The outputs were the corresponding values of ϵ , $k\sigma_1$, $k\sigma_2$, kL_1 , kL_2 , τ , and ω , which had a jointly uniform probability density. Here ϵ is the effective permittivity of the surface, $k\sigma$ is the normalized rms height (upper surface $k\sigma_1$, lower surface $k\sigma_2$), kL is the normalized surface correlation length (upper surface kL_1 , lower surface kL_2), k is the wavenumber, τ is the optical depth, and ω is the single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space [8,9].

We chose the MLP structures 8-10-7 and trained the network using the OWO-BP, LM and OWO-HWO algorithms for 50 iterations. The results are shown in Table 5.1 and in Figure 4. We see that the OWO-HWO algorithm outperforms OWO-BP easily and performs significantly better than LM in terms of training speed, MSE and generalization capability.

Table 5.1

Training and Testing Results For OWO-BP, LM and OWO-HWO algorithms

Data : <i>Power14</i> MLP (14-10-1)	Training MSE	Testing MSE
OWO-BP	10469.4	10661.5
LM	5941.4	7875.4
OWO-HWO	5144.6	7889.6
Data : <i>Single2</i> MLP (16-20-3)	Training MSE	Testing MSE
OWO-BP	0.64211	0.89131
LM	0.20379	0.33019
OWO-HWO	0.10881	0.18279
Data : <i>F17</i> MLP (17-20-9)	Training MSE	Testing MSE
OWO-BP	133223657.0	139572289.8
LM	20021846.0	21499275.8
OWO-HWO	22158499.8	22284084.9
Data : <i>Twod</i> MLP (8-10-7)	Training MSE	Testing MSE
OWO-BP	0.257819	0.283201
LM	0.172562	0.195689
OWO-HWO	0.159601	0.174393

6. Conclusions

In this paper we have developed a new MLP training method, termed OWO-HWO, and have shown the convergence of its training error. We have demonstrated the training and generalization capabilities of OWO-HWO using several examples. There are several equivalent methods for generating HWO weight changes. The matrix inversion approach seems to be the most efficient for large training data sets. Although the HWO

component of the algorithm utilizes separate error functions for each hidden unit, we have shown that OWO-HWO is equivalent to linearly transforming the training data and then performing OWO-BP. Simulation results tell us that OWO-HWO is more effective than the OWO-BP and Levenberg-Marquardt methods for training MLP networks.

Acknowledgements

This work was supported by the state of Texas through the Advanced Technology Program under grant number 003656-063. Also, we thank the reviewers for their helpful comments and suggestions.

7. References

- [1] J.A. Anderson, *An Introduction to Neural Networks* (The MIT Press, Cambridge, MA, 1986).
- [2] P. Baldi and K. Hornik, Neural Networks and Principal Component Analysis: Learning from examples without local minima, *Neural Networks*, Vol. 2, (1989) 53-58.
- [3] S.A. Barton, A matrix method for optimizing a neural network, *Neural Computation*, Vol. 3, No. 3, (1991) 450-459.
- [4] R. Battiti, First- and Second – Order Methods for Learning : Between Steepest Descent and Newton’s Method, *Neural Computation*, Vol. 4, No.2, (1992), 141-166.
- [5] M.S. Chen and M. T. Manry, Back-propagation representation theorem using power series, *Proceedings of International Joint Conference on Neural Networks*, San Diego, Vol. 1, (1990) 643-648.
- [6] M.S. Chen and M.T. Manry, Nonlinear Modeling of Back-Propagation Neural Networks, *Proceedings of International Joint Conference on Neural Networks*, Seattle, (1991) A-899.
- [7] M.S. Chen and M.T. Manry, Conventional Modeling of the Multi-Layer Perceptron Using Polynomial

- Basis Function, *IEEE Transactions on Neural Networks*, Vol. 4, No. 1, (1993) 164-166.
- [8] M.S. Dawson, *et al*, Inversion of surface parameters using fast learning neural networks, *Proceedings of International Geoscience and Remote Sensing Symposium*, Houston, Texas, Vol. 2, (1992) 910-912.
- [9] M.S. Dawson, A.K. Fung and M.T. Manry, Surface parameter retrieval using fast learning neural networks, *Remote Sensing Reviews*, Vol. 7(1), (1993) 1-18.
- [10] R. Fletcher, Conjugate Direction Methods, in: W. Murray, ed., *Numerical Methods for Unconstrained Optimization* (Academic Press, New York, 1972).
- [11] M.H. Fun and M.T. Hagan, Levenberg-Marquardt Training for Modular Networks, *The 1996 IEEE International Conference on Neural Networks*, Vol. 1, (1996) 468-473.
- [12] A.K. Fung, Z. Li, and K.S. Chen, Backscattering from a Randomly Rough Dielectric Surface, *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 30, No. 2, (1992) 356-369.
- [13] A.K. Fung, *Microwave Scattering and Emission Models and Their Applications* (Artech House, 1994).
- [14] P.E. Gill, W.Murray and M.H.Wright, *Practical Optimization* (Academic Press, New York, 1981).
- [15] A. Gopalakrishnan, *et al*, Constructive Proof of Efficient Pattern storage in the Multilayer Perceptron, *Conference Record of the Twenty-seventh Annual Asilomar Conference on Signals, Systems, and Computers*, Vol. 1, (1993) 386-390.
- [16] M.T. Hagan and M.B. Menhaj, Training Feedforward Networks with the Marquardt Algorithm, *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, (1994) 989-993.
- [17] S. Kollias and D. Anastassiou, An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks, *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 8, (1989) 1092-1101.
- [18] M.T. Manry, *et al*, Fast Training of Neural Networks for Remote Sensing, *Remote Sensing Reviews*, Vol. 9, (1994) 77-96.
- [19] T. Masters, *Neural, Novel & Hybrid Algorithms for Time Series Prediction* (John Wiley & Sons, Inc., 1995).

- [20] S. McLoone, M.D. Brown, G. Irwin and G. Lightbody, A Hybrid Linear/Nonlinear training Algorithm for Feedforward Neural Networks, *IEEE Transactions on Neural Networks*, Vol. 9, No. 9, (1998) 669-683.
- [21] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, New York, 1989).
- [22] D.B. Parker, "Learning Logic," *Invention Report S81-64*, File 1, Office of Technology Licensing, Stanford University, 1982.
- [23] W.H. Press, *et al*, *Numerical Recipes* (Cambridge University Press, New York, 1986).
- [24] K. Rohani, M.S. Chen and M.T. Manry, Neural subnet design by direct polynomial mapping, *IEEE Transactions on Neural Networks*, Vol. 3, No. 6, (1992) 1024-1026.
- [25] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart and J.L. McClelland ed., *Parallel Distributed Processing*, Vol. 1, (The MIT Press, Cambridge, MA, 1986).
- [26] M.A. Sartori and P.J. Antsaklis, A simple method to derive bounds on the size and to train multilayer neural networks, *IEEE Transactions on Neural Networks*, Vol. 2, No. 4, (1991) 467-471.
- [27] R.S. Scalerio and N. Tepedelenlioglu, A Fast New Algorithm for Training Feedforward Neural Networks, *IEEE Transactions on Signal Processing*, Vol. 40, No. 1, (1992) 202-210.
- [28] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. dissertation, Committee on Applied Mathematics, Harvard University, Cambridge, MA, Nov. 1974.
- [29] P. Werbos, Backpropagation: Past and future, the *Proceedings of the IEEE International Conference on Neural Networks*, (1988) 343-353.
- [30] B. Widrow and S.D. Stearns, *Adaptive Signal Processing*, (Prentice-Hall, Englewood Cliffs, NJ, 1985).
- [31] J.Y.F. Yam and T. W. S. Chow, Extended Least Squares Based Algorithm for Training Feedforward Networks, *IEEE Transactions on Neural Networks*, Vol. 8, No. 3, (1997) 803-810

Figure 1 - Simulation Results for example 1
Data: POWER14.TRA, Structure : 14-10-1

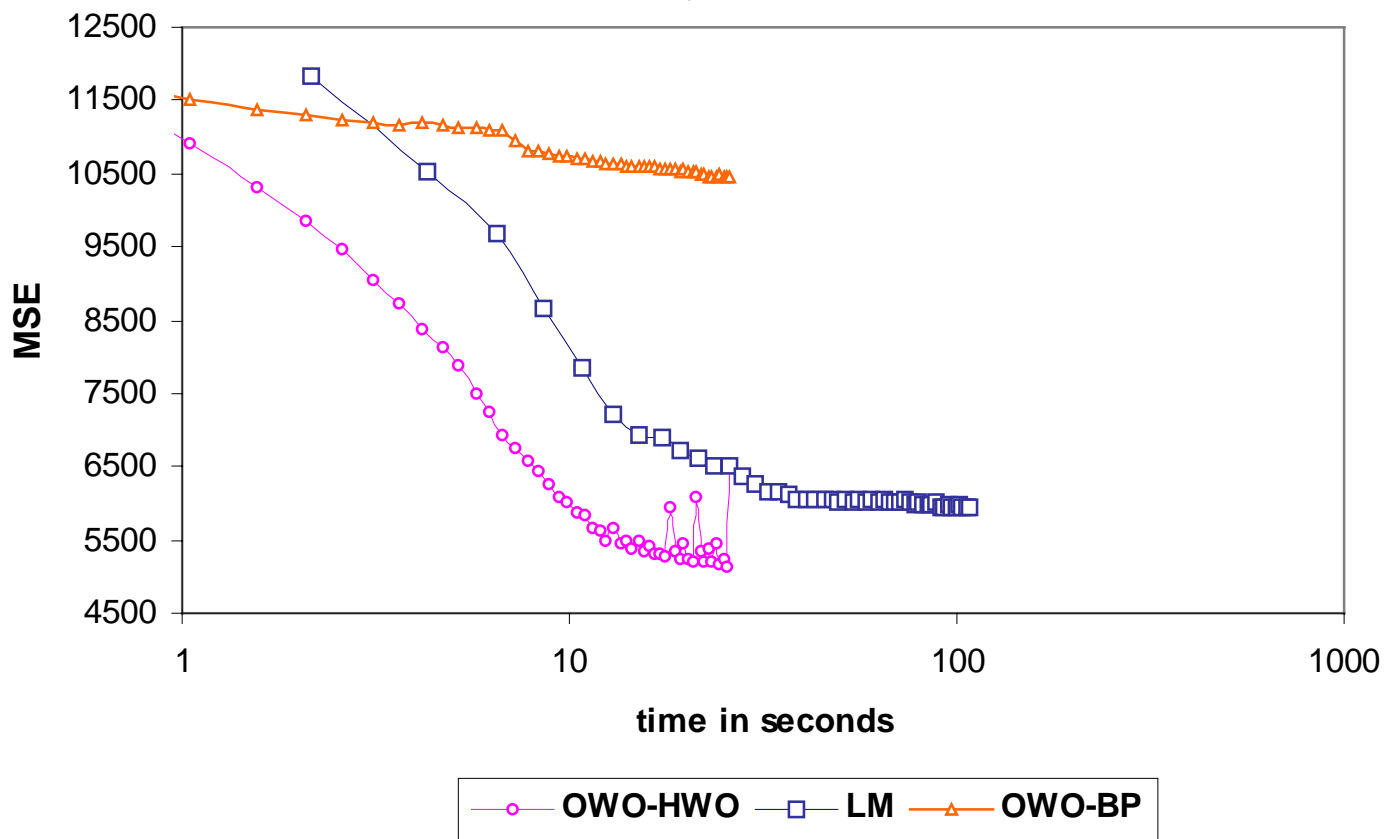


Figure 2 - Simulation Results for example 2
Data: SINGLE2.TRA, Structure : 16-20-3

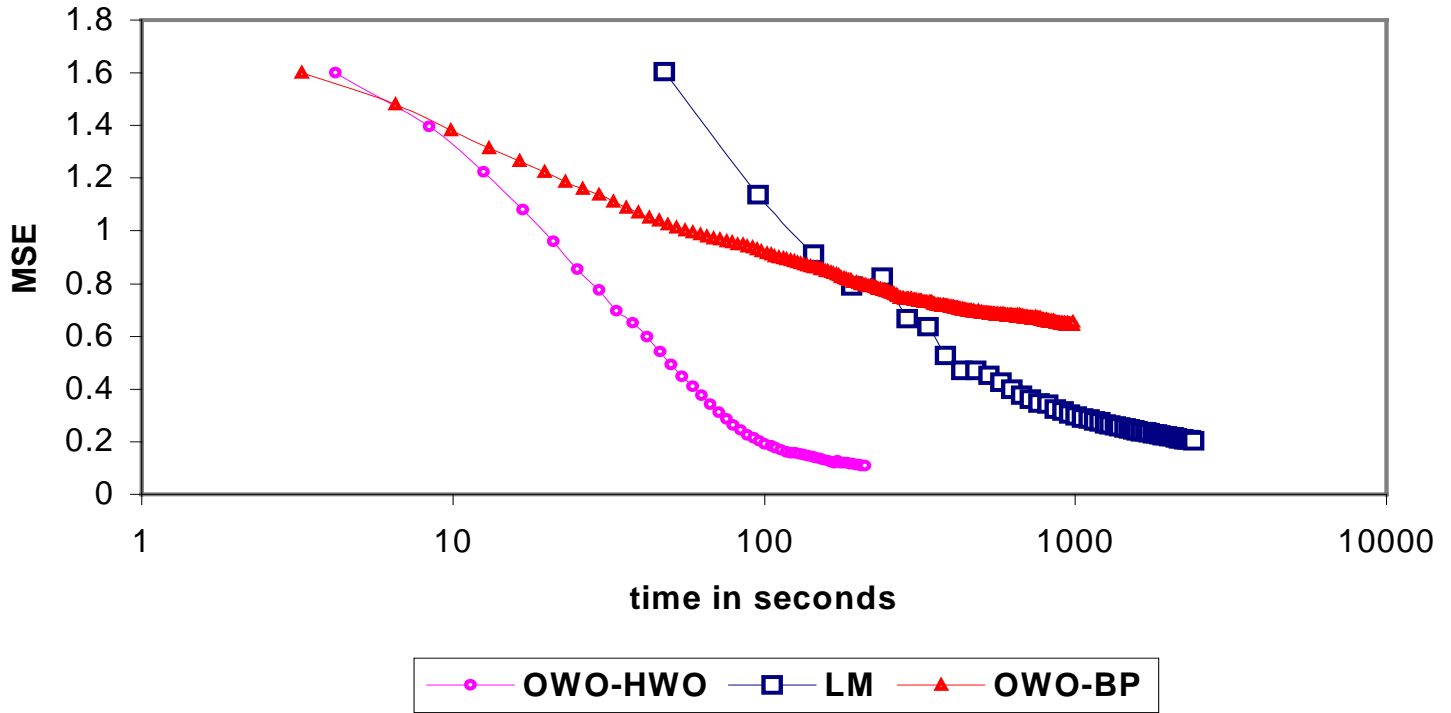


Figure 3 - Simulation Results for example 3
Data: F17.TRA Structure: 17- 20 - 9

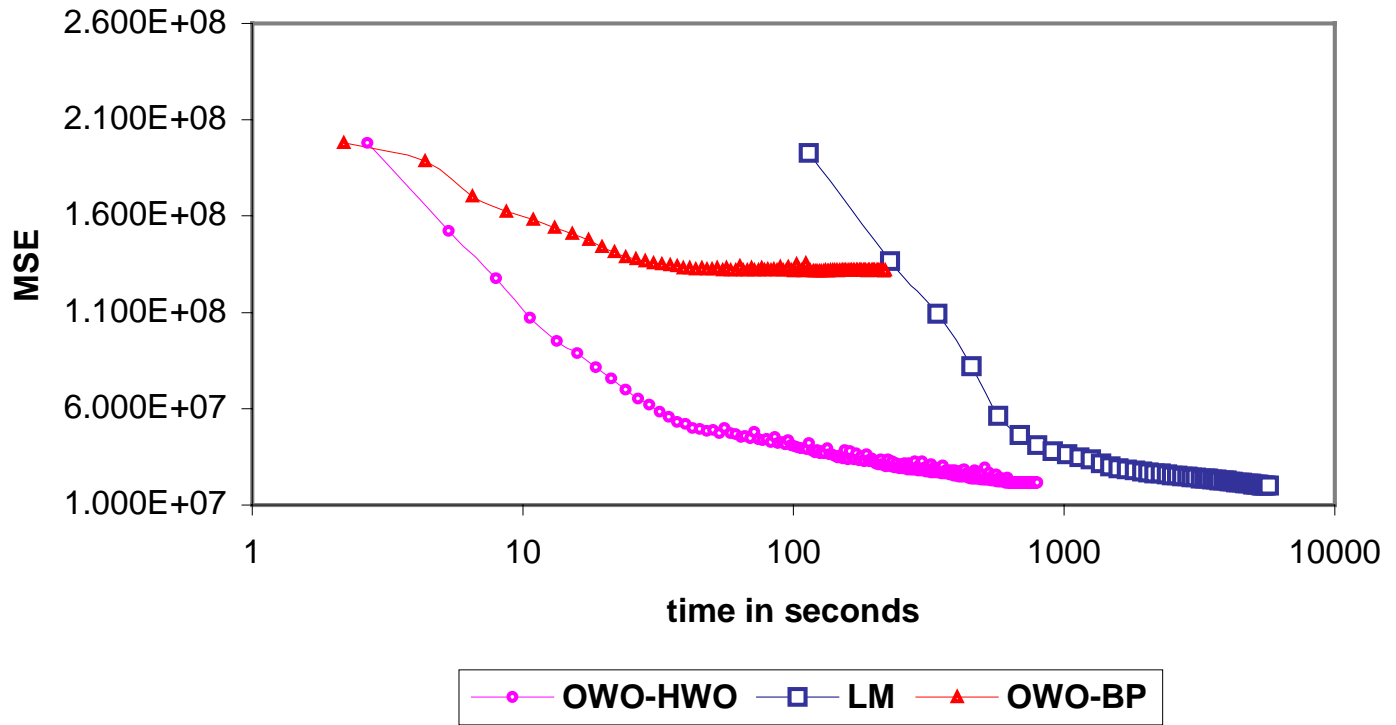
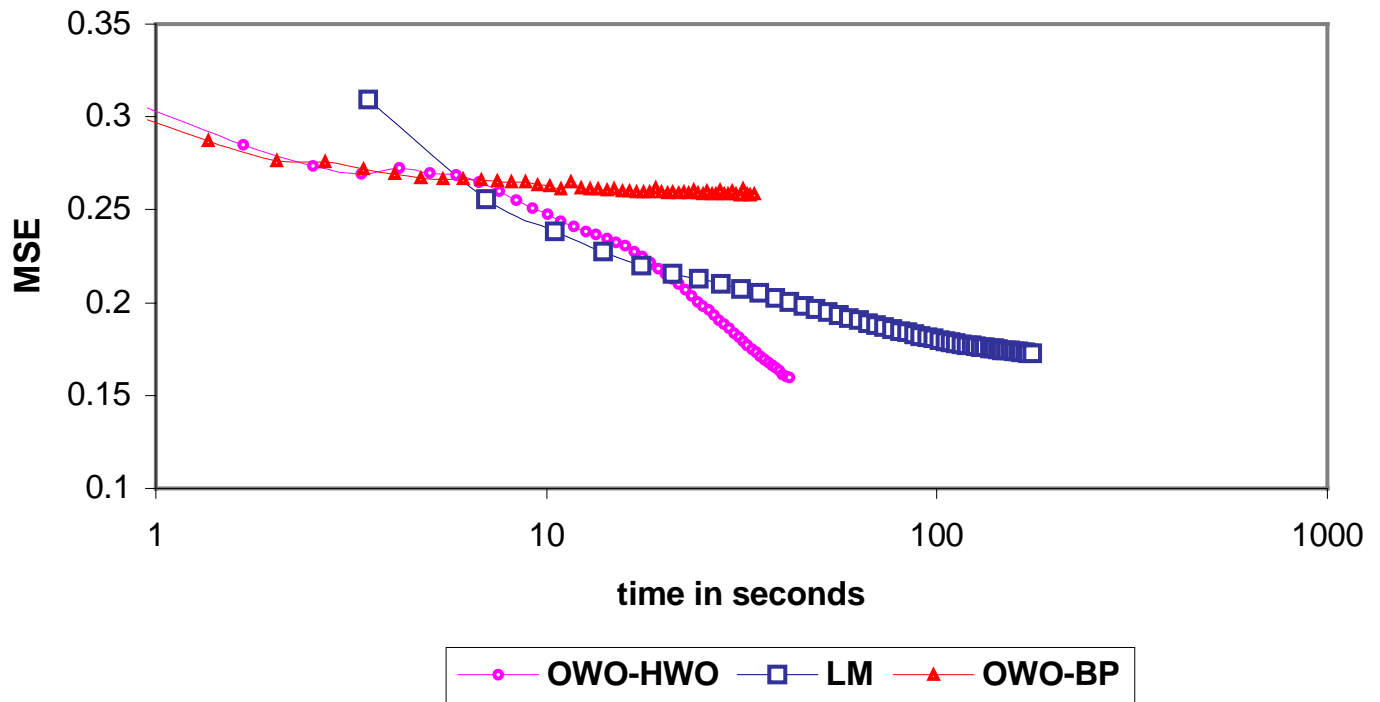
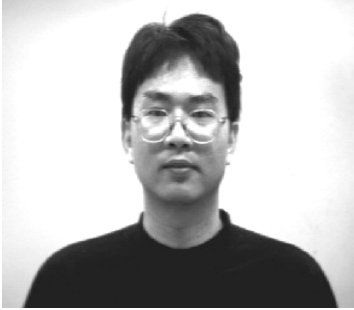


Figure 4 - Simulation Results for example 4
Data: TWOD.TRA, Structure : 8-10-7





Hung-Han Chen received his B.S. in Electrical Engineering in 1988 from National Cheng Kung University, Taiwan, and M.S. in Electrical Engineering in 1993 from West Coast University, L.A. He joined the Neural Networks and Image Processing Lab in the EE department as a graduate research assistant in 1994. There, he investigated fast algorithms for training feedforward neural networks and applied this research to power load forecasting. In 1997, he received his Ph.D. degree in Electrical Engineering from University of Texas at Arlington. His areas of interest include Neural Networks, Digital Communication, Robotic Control, and Artificial Intelligence. Currently, Dr. Chen works at CYTEL Systems, Inc., Hudson, Massachusetts, where he develops neural network based software for vehicle identification and other applications.



Michael T. Manry was born in Houston, Texas in 1949. He received the B.S., M.S., and Ph.D. in Electrical Engineering in 1971, 1973, and 1976 respectively, from The University of Texas at Austin. After working there for two years as an Assistant Professor, he joined Schlumberger Well Services in Houston where he developed signal processing algorithms for magnetic resonance well logging and sonic well logging. He joined the Department of Electrical Engineering at the University of Texas at Arlington in 1982, and has held the rank of Professor since 1993. In Summer 1989, Dr. Manry developed neural networks for the Image Processing Laboratory of Texas Instruments in Dallas. His recent work, sponsored by the Advanced Technology Program of the state of Texas, E-Systems, Mobil Research, and NASA has involved the development of techniques for the analysis and fast design of neural networks for image processing, parameter estimation, and pattern classification. Dr. Manry has served as a consultant for the Office of Missile Electronic Warfare at White Sands Missile Range, MICOM (Missile Command) at Redstone Arsenal, NSF, Texas Instruments, Geophysics International, Halliburton Logging Services, Mobil Research and Verity Instruments. He is a Senior Member of the IEEE.



Hema Chandrasekaran received her B.Sc degree in Physics in 1981 from the University of Madras, B.Tech in Electronics from the Madras Institute of Technology in 1985, and M.S. in Electrical Engineering from the University of Texas at Arlington in 1994. She is currently pursuing her Ph.D in Electrical Engineering at the University of Texas at Arlington. Her specific research interests include neural networks, image and speech coding, signal processing algorithms for communications. She is currently a Graduate Research Assistant in the Image Processing and Neural Networks Laboratory at UT Arlington.