Recursive Dynamic Node Creation in Multilayer Neural Networks

Mahmood R. Azimi-Sadjadi, *Senior Member, IEEE*, Sassan Sheedvash, and Frank O. Trujillo

Abstract—**This paper presents the derivations of a novel approach for simultaneous recursive weight adaptation and node creation in multilayer back-propagation neural networks. The method uses time and order update formulations in the orthogonal projection method to derive a recursive weight updating procedure for the training process of the neural network and a recursive node creation algorithm for weight adjustment of a layer with added nodes during the training process. The proposed approach allows optimal dynamic node creation in the sense that the mean-squared error is minimized for each new topology. The effectiveness of the algorithm is demonstrated on several benchmark problems, namely, the multiplexer and the decoder problems as well as a real world application for detection and classification of buried dielectric anomalies using a microwave sensor.**

I. Introduction

In a supervised neural network such as the back-propagation network the choices of the training algorithm, the network architecture, the input signal representation, and the training set, play dominant roles for optimal training and generalization capability of the network.

The choice of the training algorithm determines the rate of convergence to a solution and the optimality of the training. The training process of the back-propagation network is based upon the least-squares criterion. If enough training samples and internal parameters are used, the input–output mapping may be approximated to within an arbitrary accuracy [1]. In this case, the performance of the network can approach to that of Bayes estimator which is optimal [2]. Fast adaptation or learning is one of the main issues in these neural networks. In [3], [4], a new algorithm for expediting the learning process of multilayer neural networks is introduced using a recursive least squares (RLS) based method.

The choice of the network architecture is another important consideration for optimal training and generalization characteristics. It is proved [5] that a three-layer neural network with Sigmoidal type nonlinearity at nodes can approximate any arbitrary nonlinear function and generate any complex decision region needed for classification and recognition tasks. Neural network architectures with hidden layers bottlenecks have been shown to generalize better than networks which contain larger hidden layer nodes than their previous layer [6].

However, the selection of an architecture with appropriate size has been mainly empirical. For the most part, different configurations are tried, and if they do not yield an acceptable solution, they are discarded. Another topology is then defined and the whole training process is repeated. As a result, the possible benefits of training the original network architecture are lost and the computational costs of retraining become prohibitive. Another approach involves using a larger than needed topology and training it until a convergent solution is found. After that, the weights of the network are pruned off, if their values are negligible and have no influence on the performance of the network [7]. Since the pruning approach starts with a large network, the training time is larger than necessary and the method is computationally inefficient. It may also get trapped in one of the intermediately sized solutions because of the shape of the error surface and hence never finds the smallest network solution. Additionally, the relative importance of the nodes and weights depend on the particular mapping problem which the network is attempting to approximate and the pruning method makes it difficult to come up with a general cost function that would yield small networks for an arbitrary mapping. In the procedure suggested in [8], the error curve is monitored during the training process and a node is created when the ratio of the drop in the mean squared error (MSE) over a fixed number of trials falls below *a priori* chosen threshold slope. This procedure then uses the conventional, LMS-type, back-propagation algorithm to train the new architecture.

In this paper a new recursive procedure for node creation in multilayer back-propagation neural networks is introduced. The derivations of the methodology are based upon the application of the Orthogonal Projection Theorem [12]. Simulation results on various examples are presented which indicate the effectiveness of the node creation scheme developed in this paper when used in conjunction with the RLS based learning method.

II. Training Process of Multilayer Neural Network

In this section the problem of weight updating in multilayer neural networks is formulated in the context of the geometric orthogonal projection [11], [12]. The sum of the squared error is viewed as the squared length (or norm) of an error vector which is minimized using the geometric approach. It will be shown that the solution of the time updating leads to the RLS adaptation [9], [10], and the solution to the order updating allows us to recursively add nodes to the hidden layers during the training process.

Consider an M-layer network as shown in Fig. 1. This network has N_0 inputs, N_l hidden layer nodes in layer $l, l \in$

Manuscript received December 23, 1991; revised July 3, 1992.
M. R. Azimi-Sadjadi and F. O. Trujillo are with the Department of Electrical Engineering, Colorado State University, Fort Collins, CO.
S. Sheedvash is with IBM Corp., Austin, TX 78712.
IEEE Log Number 9203736.

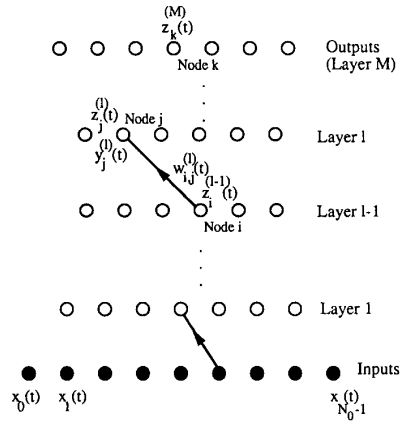


Fig. 1. A multilayer neural network.

$[1, M-1]$ and N_M output nodes. At training sample t, we denote the inputs to the first layer by $x_i(t)$'s, $i \in [0, N_0 - 1]$, the inputs to other layers, say layer l, by $y_j^{(l)}(t), j \in [0, N_l - 1]$ and the outputs of this layer by $z_j^{(l)}(t)$. Then the total input to node j in layer l can be expressed as

$$y_j^{(l)}(t) = \sum_{i=0}^{N_{l-1}-1} w_{i,j}^{(l)}(t)\, z_i^{(l-1)}(t) \qquad \forall j \in [0, N_l - 1], \text{ and } \forall l \in [1, M] \tag{1}$$

where $w_{i,j}^{(l)}(t)$ represents the weight connecting node $i, i \in [0, N_{l-1} - 1]$ in layer $l-1$ to node $j, j \in [0, N_l - 1]$ in layer $l, l \in [1, M]$. Alternatively, in vector form we can write

$$y_j^{(l)}(t) = \vec{Z}^{(l-1)^T}(t)\, \vec{W}_j^{(l)}(t), \qquad \forall j \in [0, N_l - 1], \text{ and } \forall l \in [1, M] \tag{2a}$$

where

$$\vec{W}_j^{(l)}(t) := \left[w_{0,j}^{(l)}(t), w_{1,j}^{(l)}(t), \ldots, w_{N_{l-1}-1,j}^{(l)}(t)\right]^T \tag{2b}$$

$$\vec{Z}^{(0)}(t) := [x_0(t), x_1(t), \ldots, x_{N_0-1}(t)]^T \tag{2c}$$

$$\vec{Z}^{(l)}(t) := \left[z_0^{(l)}(t), z_1^{(l)}(t), \ldots, z_{N_l-1}^{(l)}(t)\right]^T, \qquad \forall l \in [1, M] \tag{2d}$$

where $\vec{W}_j^{(l)}(t)$ is the weight vector associated with node j in layer l and $Z^{(l-1)}(t)$ represents the input vector to layer l (coming from layer $l-1$). The output of node j, $z_j^{(l)}(t)$, has a real value that is normally a nonlinear function of the total input, $y_j^{(l)}(t)$. If the standard threshold nonlinear activation function, as in Fig. 2 is used, then this output is given by

$$z_j^{(l)}(t) = \begin{cases} 0 & y_j^{(l)}(t) \le 0, \\ \frac{1}{a_l} y_j^{(l)}(t) & 0 < y_j^{(l)}(t) \le a_l, \\ 1 & y_j^{(l)}(t) > a_l \end{cases} \tag{3}$$

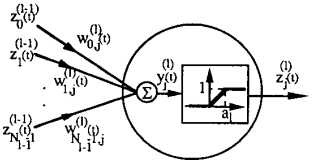


Fig. 2. A model for neurons (nodes) in the network.

where $1/a_l$ determines the slope of the ramp region. Note that similar input/output relationship can be used for the piecewise linearized sigmoidal function. In this case $1/a_l(t)$ should be determined at every iteration. However, in this paper for simplicity in derivations and without loss of generality we have considered the threshold logic nonlinear activation function.

Now, the aim of weight updating is to find an appropriate set of weight vectors $\vec{W}_j^{(l)}(t)$ for $\forall l \in [1, M]$, and for $\forall j \in [0, N_l - 1]$, so that the global weighted sum of the squared errors between the actual output, i.e., $z_i^{(M)}(t)$ and the desired output (externally specified), $d_i^{(M)}(t)$, is minimized over the entire training set, $t \in [1, n]$, and all output nodes, $i \in [0, N_M - 1]$. This index of performance is given by

$$J(n) = (1/2) \sum_{i=0}^{N_M-1} \sum_{t=1}^{n} \lambda^{n-t} \left[\varepsilon_i^{(M)}(t)\right]^2 \tag{4a}$$

where λ is a positive number less than but close to one which is referred to as "forgetting factor"; and the error $\epsilon_i^{(M)}(t)$ for the ith node in the output of layer M at training time t is defined as

$$\varepsilon_i^{(M)}(t) := d_i^{(M)}(t) - z_i^{(M)}(t), \qquad \text{for } i \in [0, N_M - 1]. \tag{4b}$$

The index of performance in (4) can be minimized for $\vec{W}_j^{(l)}(n)$ by taking the partial derivative of $J(n)$ with respect to $\vec{W}_j^{(l)}(n)$ and setting it equal to zero. It is shown [4] that the problem of weight adjustment using the RLS approach leads to a normal equation for each layer. For example, for layer l we get the following normal equation:

$$\frac{\partial J(n)}{\partial \vec{W}_j^{(l)}(n)} = 0 \Rightarrow \sum_{t=1}^{n} \lambda^{n-t} \frac{1}{a_l} \vec{Z}^{(l-1)}(t) \varepsilon_j^{(l)}(t) = 0, \qquad \forall l \epsilon [1, M], \quad \forall j \epsilon [0, N_l - 1] \tag{5a}$$

where

$$\varepsilon_j^{(l)}(t) := d_j^{(l)}(t) - \frac{1}{a_l} \vec{Z}^{(l-1)T}(t)\, \hat{\vec{W}}_j^{(l)}(n). \tag{5b}$$

Here "$\wedge$" represents the estimate of the relevant quantity, and $d_j^{(l)}(t)$ is the desired output for node j in the output of layer l which is determined through the error back-propagation procedure as described in [5].
That is

$$d_j^{(l)}(t) := z_j^{(l)}(t) + \varepsilon_j^{(l)}(t) \tag{6a}$$

where the error $\epsilon_j^{(l)}(t)$ is determined by

$$\varepsilon_j^{(l)}(t) = \frac{1}{a_{l+1}} \sum_{k=0}^{N_{l+1}-1} \hat{w}_{j,k}^{(l+1)}(n) \varepsilon_k^{(l+1)}(t) \tag{6b}$$

where $\epsilon_k^{(l+1)}(t)$ represents the error at node k in the output of layer $(l+1)$ which is back-propagated through the weights of this layer, i.e., $\hat{w}_{j,k}^{(l+1)}(n)$, to generate $\epsilon_j^{(l)}(t)$, the error at node j in the output of previous layer l.

Note that when the total input to node j is off the ramp region of the threshold logic nonlinearity function, the derivative in (5a) is always zero since $(\partial z_j^{(l)}(t))/(\partial \vec{W}_j^{(l)}(n)) = 0$, (see [4, eqs. (15) and (16)]). This implies that the normal equation in (5a) will only be solved when the input to the relevant node lies within the ramp region and in this case the error is given by (5b).

Since each layer can be treated independently we can, for simplicity in notation, drop the layer index, l, and assume that the slope of the nonlinear activation function, i.e., $a_l = 1$, for $\forall l \in [1, M]$. The latter assumption is equivalent to normalizing the input vector $\vec{Z}^{(l-1)}(t)$ by dividing its elements by a_l. The normal (5a) can then be rewritten in a more compact form as

$$\underline{Y}_N^T(n)\vec{E}_j(n) = 0, \qquad \text{for } \forall j \in [0, N-1] \tag{7a}$$

where $\vec{E}_j(n)$ is the n-dimensional error vector given by

$$\vec{E}_j(n) := \underline{\Lambda}^{1/2}(n)[\varepsilon_j(1), \varepsilon_j(2), \ldots, \varepsilon_j(n)]^T \tag{7b}$$

$\underline{Y}_N(n)$ is the input matrix defined by

$$\underline{Y}_N(n) := \underline{\Lambda}^{1/2}(n)\Big[\vec{Z}(1), \vec{Z}(2), \ldots, \vec{Z}(n)\Big]^T \tag{7c}$$

and

$$\underline{\Lambda}^{1/2}(n) := \mathrm{Diag}\Big[\lambda^{(n-1)/2}, \lambda^{(n-2)/2}, \ldots, \lambda^{1/2}, 1\Big]. \tag{7d}$$

Let us also define the desired output vector $\vec{D}_j(n)$ as

$$\vec{D}_j(n) := \underline{\Lambda}^{1/2}(n)[d_j(1), d_j(2), \ldots, d_j(n)]^T. \tag{8}$$

Now using (5b), (7a) can alternatively be written as

$$\underline{Y}_N^T(n)\Big(\vec{D}_j(n) - \underline{Y}_N(n)\hat{\vec{W}}_j(n)\Big) = 0, \quad \text{for } \forall j \in [0, N-1]. \tag{9}$$

This normal equation can be solved for $\hat{\vec{W}}_j(n)$ to give the standard least squares (LS) solution as

$$\hat{\vec{W}}_j(n) = \big(\underline{Y}_N^T(n)\underline{Y}_N(n)\big)^{-1}\underline{Y}_N^T(n)\vec{D}_j(n) \tag{10}$$

which is the optimum weight vector for node j in a particular layer. This vector yields the minimum error vector,

$$\vec{E}_j(n) = \Big[\underline{I} - \underline{Y}_N(n)\big(\underline{Y}_N^T(n)\underline{Y}_N(n)\big)^{-1}\underline{Y}_N^T(n)\Big]\vec{D}_j(n). \tag{11}$$

Let us introduce the following projection operators [11], [12]

$$\underline{P}_Y(n) := \underline{Y}_N(n)\big(\underline{Y}_N^T(n)\underline{Y}_N(n)\big)^{-1}\underline{Y}_N^T(n), \quad \text{and } \underline{P}_Y^{\perp}(n) := \underline{I} - \underline{P}_Y(n) \tag{12}$$

where $\underline{P}_Y(n)$ is the projection matrix on the column space of $\underline{Y}_N(n)$ and $\underline{P}_Y^{\perp}(n)$ is its orthogonal complement. Thus, using the definitions in (12) the minimum error vector in (11) can be represented as

$$\vec{E}_j(n) = \underline{P}_Y^{\perp}(n)\vec{D}_j(n) \tag{13a}$$

and the normal equation in (9) can be represented as

$$\underline{Y}_N^T(n)\underline{P}_Y^{\perp}(n)\vec{D}_j(n) = 0. \tag{13b}$$

Let us also introduce another matrix $\underline{Q}_Y(n)$ as

$$\underline{Q}_Y(n) := \underline{Y}_N(n)\big(\underline{Y}_N^T(n)\underline{Y}_N(n)\big)^{-1} \tag{14}$$

which is the left generalized inverse of the input data matrix $\underline{Y}_N(n)$. Then the transpose of the optimum weight vector $\hat{\vec{W}}_j(n)$ in (10) can be represented as

$$\hat{\vec{W}}_j^T(n) = \vec{D}_j^T(n)\underline{Q}_Y(n). \tag{15}$$

From definitions $\underline{P}_Y(n)$ and $\underline{Q}_Y(n)$ in (12) and (14), it can readily be verified that the following properties hold for these matrices

$$\underline{P}_Y^T(n) = \underline{P}_Y(n) \tag{16a}$$

$$\underline{P}_Y^m(n) = \underline{P}_Y(n) \tag{16b}$$

$$\underline{Q}_Y(n)\underline{Y}_N^T(n) = \underline{P}_Y(n) \tag{16c}$$

$$\underline{P}_Y(n)\underline{Q}_Y(n) = \underline{Q}_Y(n) \tag{16d}$$

and

$$\underline{Y}_N^T(n)\underline{Q}_Y(n) = \underline{I}_N \tag{16e}$$

where $\underline{I}_N$ is an identity matrix of size $N \times N$.

III. Time and Order Update Processes

Let $\underline{Y}, \underline{Z}, \underline{U}$, and $\underline{V}$ be arbitrary matrices of the same column length, then the following orthogonal updating equations hold [11]

$$\underline{P}_{Y:Z} = \underline{P}_Y + \underline{P}_Y^{\perp}\underline{Z}\Big(\underline{Z}^T\underline{P}_Y^{\perp}\underline{Z}\Big)^{-1}\underline{Z}^T\underline{P}_Y^{\perp} \tag{17a}$$

$$\underline{P}_{Y:Z}^{\perp} = \underline{P}_Y^{\perp} + \underline{P}_Y^{\perp}\underline{Z}\Big(\underline{Z}^T\underline{P}_Y^{\perp}\underline{Z}\Big)^{-1}\underline{Z}^T\underline{P}_Y^{\perp} \tag{17b}$$

and

$$\underline{Q}_{Y:Z} = \Big[\underline{Q}_Y \quad 0\Big] + \underline{P}_Y^{\perp}\underline{Z}\Big(\underline{Z}^T\underline{P}_Y^{\perp}\underline{Z}\Big)^{-1}\Big[-\underline{Z}^T\underline{Q}_Y \quad 1\Big] \tag{17c}$$

where $\underline{P}_{Y:Z}$ represents the projection operator for the combined space spanned by $\underline{Y}$ and $\underline{Z}$; and $Y : Z$ denotes the appending of vector $\vec{Z}$ to column space of $\underline{Y}$. Pre- and post-multiplying (17b) by $\underline{U}^T$ and $\underline{V}$, respectively, yields

$$\underline{U}^T\underline{P}_{Y:Z}^{\perp}\underline{V} = \underline{U}^T\underline{P}_Y^{\perp}\underline{V} - \Big(\underline{U}^T\underline{P}_Y^{\perp}\underline{Z}\Big)\Big(\underline{Z}^T\underline{P}_Y^{\perp}\underline{Z}\Big)^{-1}\Big(\underline{Z}^T\underline{P}_Y^{\perp}\underline{V}\Big). \tag{18a}$$

Also from (17c) we have

$$\underline{U}^T\underline{Q}_{Y:Z} = \left[\underline{U}^T\underline{Q}_Y \quad 0\right] + \left(\underline{U}^T\underline{P}_Y^{\perp}\underline{Z}\right)\left(\underline{Z}^T\underline{P}_Y^{\perp}\underline{Z}\right)^{-1} \cdot \left[-\underline{Z}^T\underline{Q}_Y \quad 1\right]. \tag{18b}$$

In the next section we shall derive the "time" and "order" update equations for the optimum weight vector in (15) using the projection update formula.

A. Time Updating and Weight Adaptation

In this section, the identities in (17) are used to derive the time update relationships. To achieve time updating and hence weight adaptation let $\vec{Z} = \vec{\sigma}(n) := [0, \ldots, 1]^T$ and $\underline{Y} = \underline{Y}_N(n)$. Note that the dimension of vector $\vec{\sigma}(n)$ has to be equal to the dimension (length) of the columns of the data matrix $\underline{Y}_N(n)$, i.e., the current time index n. Then substituting for $\vec{Z} = \vec{\sigma}(n)$ and $\underline{Y} = \underline{Y}_N(n)$ into (17a) and (17c), respectively, yields

$$\underline{P}_{Y_N:\sigma}(n) = \underline{P}_{Y_N}(n) + \underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n)\left(\vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n)\right)^{-1} \cdot \vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n) \tag{19a}$$

and

$$\underline{Q}_{Y_N:\sigma}(n) = \left[\underline{Q}_{Y_N}(n) \quad 0\right] + \underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n) \cdot \left(\vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n)\right)^{-1} \cdot \left[-\vec{\sigma}^T(n)\underline{Q}_{Y_N}(n) \quad 1\right]. \tag{19b}$$

However, invoking the property $\underline{P}_{Y_N:\sigma}(n) = \underline{P}_{\sigma:Y_N}(n)$ and using (17a) we can write

$$\underline{P}_{Y_N:\sigma}(n) = \underline{P}_{\sigma:Y_N}(n) = \underline{P}_\sigma + \underline{P}_\sigma^{\perp}\underline{Y}_N(n) \cdot \left(\underline{Y}_N^T(n)\underline{P}_\sigma^{\perp}\underline{Y}_N(n)\right)^{-1}\underline{Y}_N^T(n)\underline{P}_\sigma^{\perp} \tag{20}$$

where the projection operator $\underline{P}_\sigma$ is

$$\underline{P}_\sigma = \vec{\sigma}(n)\left(\vec{\sigma}^T(n)\vec{\sigma}(n)\right)^{-1}\vec{\sigma}^T(n) = \begin{bmatrix} \underline{0} & \vec{0} \\ \vec{0} & 1 \end{bmatrix} \tag{21a}$$

and

$$\underline{P}_\sigma^{\perp} = \underline{I} - \underline{P}_\sigma = \begin{bmatrix} \underline{I}_{n-1} & \vec{0} \\ \vec{0} & \vec{0} \end{bmatrix}. \tag{21b}$$

The interesting property of the vector $\vec{\sigma}(n)$ which is called the pinning vector or the time annihilator [11] is that the projection of $\underline{Y}_N(n)$ onto the space spanned by $\vec{\sigma}(n)$, i.e., $\underline{P}_\sigma\underline{Y}_N(n)$ is

$$\underline{P}_\sigma\underline{Y}_N(n) = \begin{bmatrix} 0 \\ \vec{Z}(n) \end{bmatrix}, \tag{41}$$

i.e., the current data vector. Thus

$$\underline{P}_\sigma^{\perp}\underline{Y}_N(n) := \overline{Y}_N(n-1) = \begin{bmatrix} \underline{Y}_N(n-1) \\ 0 \end{bmatrix}$$

is the data matrix with its current data vector (last row) removed and replaced by a zero row. Since the column space of $\bar{Y}_N(n-1)$ is orthogonal to $\vec{\sigma}(n)$, we have

$$\underline{P}_{\sigma:Y_N}(n) = \underline{P}_{\overline{Y}_N}(n-1) + \underline{P}_\sigma = \begin{bmatrix} \underline{P}_{Y_N}(n-1) & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \underline{P}_{Y_N}(n-1) & 0 \\ 0 & 1 \end{bmatrix} \tag{22}$$

and obviously

$$\underline{P}_{\sigma:Y_N}^{\perp}(n) = \begin{bmatrix} \underline{P}_{Y_N}^{\perp}(n-1) & 0 \\ 0 & 0 \end{bmatrix}. \tag{23}$$

Now using (23) and the property in (16d) we obtain

$$\begin{bmatrix} \underline{P}_{Y_N}(n-1) & 0 \\ 0 & 1 \end{bmatrix}\underline{Q}_{Y_N:\sigma}(n) = \underline{Q}_{Y_N:\sigma}(n). \tag{24}$$

From (24) and the property in (16e), one can easily deduce that

$$\underline{Q}_{Y_N:\sigma}(n) = \begin{bmatrix} \lambda^{-1/2}\underline{Q}_{Y_N}(n-1) & * \\ 0 & 1 \end{bmatrix} \tag{25}$$

where "$*$" denotes a don't care block. Using (25) in (19b) we obtain

$$\begin{bmatrix} \lambda^{-1/2}\underline{Q}_{Y_N}(n-1) \\ 0 \end{bmatrix} = \underline{Q}_{Y_N}(n) + \underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n) \cdot \left(\vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n)\right)^{-1}\left[-\vec{\sigma}(n)^T\underline{Q}_{Y_N}(n)\right]. \tag{26}$$

Now, in order to arrive at the weight updating equations we premultiply both sides of the "time update" equation (26) by $\vec{D}_j^T(n)$, i.e.,

$$\vec{D}_j^T(n)\begin{bmatrix} \lambda^{-1/2}\underline{Q}_{Y_N}(n-1) \\ 0 \end{bmatrix} = \vec{D}_j^T(n)\underline{Q}_{Y_N}(n) - \vec{D}_j^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n) \cdot \left(\vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{\sigma}(n)\right)^{-1}\left(\vec{\sigma}(n)^T\underline{Q}_{Y_N}(n)\right). \tag{27}$$

But using (8) and (15), we can write

$$\hat{\vec{W}}_j(n) = \hat{\vec{W}}_j(n-1) + \frac{\varepsilon_j(n)}{\gamma_N(n)}\vec{C}_N(n) \tag{28}$$

where

$$\varepsilon_j(n) := \vec{\sigma}^T(n)\underline{P}_{Y_N}^{\perp}(n)\vec{D}_j(n) = \vec{\sigma}^T(n)\vec{E}_j(n) = d_j(n) - \vec{Z}^T(n)\hat{\vec{W}}_j(n) \tag{29a}$$

$$\vec{C}_N(n) := \underline{Q}_{Y_N}^T(n)\vec{\sigma}(n) = \left(\underline{Y}_N^T(n)\underline{Y}_N(n)\right)^{-1} \cdot \underline{Y}_N^T(n)\vec{\sigma}(n) = \underline{R}_N^{-1}(n)\vec{Z}(n) \tag{29b}$$

$$\underline{R}_N(n) := \underline{Y}_N^T(n)\underline{Y}_N(n) \tag{29c}$$

and

$$\gamma_N(n) := \vec{\sigma}^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{\sigma}(n) = \vec{\sigma}^T(n)\left[\underline{I} - \underline{P}_{Y_N}(n)\right]\vec{\sigma}(n)$$
$$= 1 - \vec{\sigma}^T(n)\underline{P}_{Y_N}(n)\vec{\sigma}(n) = 1 - \vec{Z}^T(n)\underline{R}_N^{-1}(n)\vec{Z}(n). \tag{29d}$$

In most of the signal processing applications, one can exploit the shifting or the serial property of the data sequence and obtain recursive relationships for computing $\vec{C}_N(n)$ and $\gamma_N(n)$. This would lead to the fast transversal filters (FTF) formulations [11] which require only $O(N)$ operations as opposed to $O(N^2)$ operations for the standard RLS approach. For neural networks, however, the seriality property can not be assumed for the input and output sequences in each layer. Thus it is inevitable to use a RLS-based scheme [3], [4] for the weight adaptation procedure. It can be shown that the updating equation (28) is equivalent to that of the standard RLS by expressing the terms in $\varepsilon_j(n)/\gamma_N(n)$ and $\vec{C}_N(n)$ using the RLS equations [13]; namely

$$\underline{R}_N^{-1}(n) = \lambda^{-1}\underline{R}_N^{-1}(n-1) - \frac{\lambda^{-2}\underline{R}_N^{-1}(n-1)\vec{Z}(n)\vec{Z}^T(n)\underline{R}_N^{-1}(n-1)}{1+\lambda^{-1}\vec{Z}^T(n)\underline{R}_N^{-1}(n-1)\vec{Z}(n)}. \tag{30}$$

This gives

$$\varepsilon_j^p(n) := \frac{\varepsilon_j(n)}{\gamma_N(n)} = d_j(n) - \vec{Z}^T(n)\hat{\vec{W}}_j(n-1) \tag{31a}$$

and

$$\vec{C}_N(n) = \underline{R}_N^{-1}(n)\vec{Z}(n) = \frac{\lambda^{-1}\underline{R}_N^{-1}(n-1)\vec{Z}(n)}{1+\lambda^{-1}\vec{Z}^T(n)\underline{R}_N^{-1}(n-1)\vec{Z}(n)}. \tag{31b}$$

Equations (28), (31a), and (31b) represent the RLS algorithm developed in [3], [4] and $\vec{C}_N(n)$ is the relevant gain matrix. Note that $\varepsilon_j^p(n)$ is the *a priori* error (i.e., before updating) whereas $\varepsilon_j(n)$ represents the *a posteriori* error (i.e., after updating).

B. Order Updating and Recursive Node Creation

The order update equation and the node addition formulation can be obtained by appending a column vector, $\vec{y}_N(n)$, to the column space of $\underline{Y}_N(n)$. When a new node N is added, assuming that the weights coming to this node are randomly selected, the corresponding vector is $\vec{y}_N(n) = [0, 0, \ldots, z_N(n)]^T = z_N(n)\vec{\sigma}(n)$ where $z_N(n)$ is the output of the added node.

Fig. 3 demonstrates this node addition process. The expanded $\underline{Y}(n)$ matrix is then given by

$$\underline{Y}_{N+1}(n) = [\underline{Y}_N(n) \quad \vec{y}_N(n)] = [\underline{Y}_N(n) \quad z_N(n)\vec{\sigma}(n)]. \tag{32}$$

Now using (17c) for $\underline{Y} = \underline{Y}_N(n)$ and $\vec{Z} = \vec{y}_N(n)$ and pre-multiplying the result by $\vec{D}_j^T(n)$ gives

$$\vec{D}_j^T(n)\underline{Q}_{Y_{N+1}}(n) = \left[\vec{D}_j^T(n)\underline{Q}_{Y_N}(n) \quad 0\right] + \vec{D}_j^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{y}_N(n)$$
$$\cdot \left(\vec{y}_N^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{y}_N(n)\right)^{-1}\left[-\vec{y}_N^T(n)\underline{Q}_{Y_N}(n) \quad 1\right]. \tag{33}$$

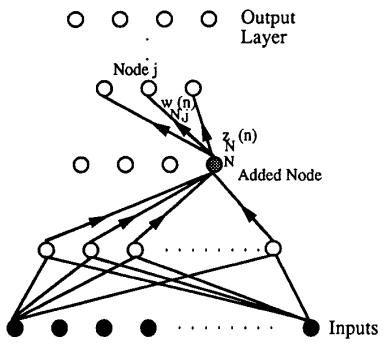


Fig. 3. A new node, N, is added to a hidden layer in the network.

Note that according to (15) the term on the left side of (33) is the new weight vector associated with node j after a node is added to its input (previous) layer. Thus we can write

$$\hat{\vec{W}}_j(N+1,n) = \begin{bmatrix}\hat{\vec{W}}_j(N,n)\\ 0\end{bmatrix} + \frac{\beta_N(n)}{\theta_N(n)}\vec{B}_N(n) \tag{34}$$

where $\vec{B}_N(n)$, $\beta_N(n)$ and $\theta_N(n)$ can be expressed, using the definitions in (29a)–(29d), in terms of time update (RLS) parameters as

$$\vec{B}_N(n) := \begin{bmatrix}-\underline{Q}^T_{Y_N}(n)\vec{y}_N(n)\\ 1\end{bmatrix} = \begin{bmatrix}-z_N(n)\underline{Q}^T_{Y_N}(n)\vec{\sigma}(n)\\ 1\end{bmatrix} = \begin{bmatrix}-z_N(n)\vec{C}_N(n)\\ 1\end{bmatrix} \tag{35a}$$

$$\beta_N(n) := \vec{y}_N^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{D}_j(n) = z_N(n)\vec{\sigma}^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{D}_j(n) = z_N(n)\varepsilon_j(n) \tag{35b}$$

and

$$\theta_N(n) := \vec{y}_N^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{y}_N(n) = z_N^2(n)\vec{\sigma}^T(n)\underline{P}^{\perp}_{Y_N}(n)\vec{\sigma}(n) = z_N^2(n)\gamma_N(n) \tag{35c}$$

where $\varepsilon_j(n)$ is the *a posteriori* error at the output of node j at iteration n as was defined in (29a). Note that $\hat{\vec{W}}_j(N,n)$ is the same as $\hat{\vec{W}}_j(n)$ used before, i.e., the weight vector associated with node j which receives N inputs prior to node addition and hence $\hat{\vec{W}}_j(N+1,n)$ is the weight vector associated with the same node after an additional node is added to its input layer. The incoming weights to the added node are updated using the standard RLS training algorithm.

C. Time–Order Update Interface

To proceed with the training using the RLS equations in Section III-A for the layer with added node (at its input), the relationship between the order-update (34) and the time-update (28) must be derived.

Comparing (34) and (28), and considering definitions in (29a)-(29d), indicate that we only need to establish a relationship between $\underline{R}_{N+1}^{-1}(n)$ and $\underline{R}_N^{-1}(n)$, in terms of the parameters $\vec{C}_N(n)$ and $\gamma_N(n)$. This would take into account for the effects of added dimension in the RLS equations as a result of node addition. Let us first partition matrix $\underline{R}_{N+1}(n)$ as follows

$$\underline{R}_{N+1}(n) = \begin{bmatrix} \underline{Y}_N^T(n) \\ \vec{Z}_N^T(n) \end{bmatrix} \begin{bmatrix} \underline{Y}_N(n) & \vec{Z}_N(n) \end{bmatrix} = \begin{bmatrix} \underline{Y}_N^T(n)\underline{Y}_N(n) & \underline{Y}_N^T(n)\vec{Z}_N(n) \\ \vec{Z}_N^T(n)\underline{Y}_N(n) & \vec{Z}_N^T(n)\vec{Z}_N(n) \end{bmatrix} \tag{36a}$$

where $\vec{Z}_N(n) = [0, 0, \ldots, z_N(n)]^T = z_N(n)\vec{\sigma}(n)$, so that

$$\underline{R}_{N+1}(n) = \begin{bmatrix} \underline{Y}_N^T(n)\underline{Y}_N(n) & z_N(n)\underline{Y}_N^T(n)\vec{\sigma}(n) \\ \vec{\sigma}^T(n)\underline{Y}_N(n)z_N(n) & z_N^2(n) \end{bmatrix} = \begin{bmatrix} \underline{R}_N(n) & z_N(n)\vec{Z}_N(n) \\ \vec{Z}_N^T(n)z_N(n) & z_N^2(n) \end{bmatrix}. \tag{36b}$$

Now using the inverse of partitioned matrices we obtain

$$\underline{R}_{N+1}^{-1}(n) = \begin{bmatrix} \underline{R}_N^{-1}(n) + \frac{\vec{C}_N(n)\vec{C}_N^T(n)}{\gamma_N(n)} & \frac{-\vec{C}_N(n)}{z_N(n)\gamma_N(n)} \\ \frac{-\vec{C}_N^T(n)}{z_N(n)\gamma_N(n)} & \frac{1}{z_N^2(n)\gamma_N(n)} \end{bmatrix}. \tag{37}$$

This time-order update interface equation allows continuous weigh adaptation after a node is added to the network.

Summary of the Algorithm

1. *Initial Architecture:* Construct a neural network architecture with a small number of hidden layer nodes.
2. *Time Update – Weight Adaptation:* Present the training data and iterate the standard RLS weight adaptation equations in conjunction with the analog of backpropagation method using (28)–(31). Monitor the Average Mean Squared Error (AMSE) at the output.
3. *Order Update – Node Creation:* If the rate of change of AMSE is not acceptable, increase the number of hidden layer nodes by one and update the outgoing weights[1] using the order update (34) and (35).
4. *Order – Time Updates Interface:* To proceed with subsequent weight updating after node creation use (37) to generate the required $\underline{R}_{N+1}^{-1}(n)$ and then switch back to the time updating process.

IV. Simulation Results

This section presents the simulation results for the multiplexer, decoder, and a target detection and classification problem. The objectives are: to compare the convergence rates as well as the detection, classification, and false-alarm rates for the target detection problem for both the fixed and the variable topologies and to study the effects of adding nodes to the first and second hidden layers on the learning behavior. For all the examples and all the cases, the networks are considered to have converged when the actual outputs are within 10% of the desired outputs for all the training data.

A. Multiplexer Network

The architecture of the multiplexer network problem consisted of three-layers with six inputs and a single output node. The numbers of first and second hidden layer nodes were varied. The two address lines activate one of the four data lines using a binary code. The desired output was the input to the activated data line. The index of performance used to determine the training of the network was the averaged mean squared error (AMSE) at the output averaged over all the output nodes. The AMSE was monitored every 25 and 40 iterations when the nodes were added to the first and the second layers, respectively, and the threshold value for the rate of change of AMSE was set to 0.001. Table I shows the convergence results for both the fixed and the variable topologies. Figs. (4a)–(4d) show the learning behavior for cases 1 and 2. Figs. (5a)–(5d) represent the corresponding learning curves for cases 3 and 4, and Figs. (6a)–(6e) show the corresponding learning curves for cases 5 through 7.

In Table I, comparing the convergence rates for the fixed and the corresponding variable architectures in cases 1 through 4 indicates a significant improvement when the nodes were added to the first hidden layer. This improvement is more obvious in cases 1 and 2 with a small initial fixed architecture than those of cases 3 and 4 with a larger initial topology. As seen from the learning curves, the addition of nodes during the early stages of the training process caused a significant reduction in the AMSE value and a sharp decline of the error. On the other hand, addition of nodes at final stages of training process caused only slight changes in the AMSE. This trend in the learning behavior seems to be consistent for all the cases 1 through 4. When the nodes were added to the second hidden layer, as in cases 5 and 6 in Table I, the improvement in the convergence rate was not so obvious. In case 5, for the fixed network architecture, the AMSE was "stuck" to a value near 0.5 and never converged to a solution. Although the variation of the topology caused the AMSE to drop to about 0.25 during the first 620 iterations, the network did not converge after adding 15 nodes during the allowed 3500 iterations. Nonetheless, the dynamic architecture showed substantially better learning behavior than its fixed counterpart. Case 6 represents an interesting situation; the AMSE for the fixed architecture was oscillating around 0.25 during the first 450 iterations and started to decline at a reasonable rate to converge in 1256 iterations. In the corresponding variable architecture, the AMSE increased to a value around 0.32 during the first 660 iterations and then started to drop slowly after 16 nodes were added to the second layer. The error continued to decrease to a value below 0.25 after 2500 iterations, but then it started to

[1]The incoming weights to the added node are updated using the standard RLS.

TABLE I
CONVERGENCE RESULTS FOR THE MULTIPLEXER PROBLEM USING FIXED AND VARIABLE TOPOLOGIES

Case	Fixed Topology		Variable Topology	
	Initial	Iterations	Final	Iterations
Note: The AMSE was monitored every 25 iterations for cases 1 through 4.				
1	6-1-5-1	4000 >	6-19-5-1	946
2	6-3-5-1	4000 >	6-24-5-1	1115
3	6-5-5-1	1964	6-19-5-1	1291
4	6-7-5-1	1256	6-24-5-1	869
Note: The AMSE was monitored every 40 iterations for cases 5 and 6.				
5	6-9-1-1	Did not Converge	6-9-16-1	3500 >
6	6-9-3-1	1256	6-9-28-1	Did not Converge
Note: The AMSE was monitored every 200 iterations for case 7.				
7	6-9-3-1	1256	6-9-6-1	903

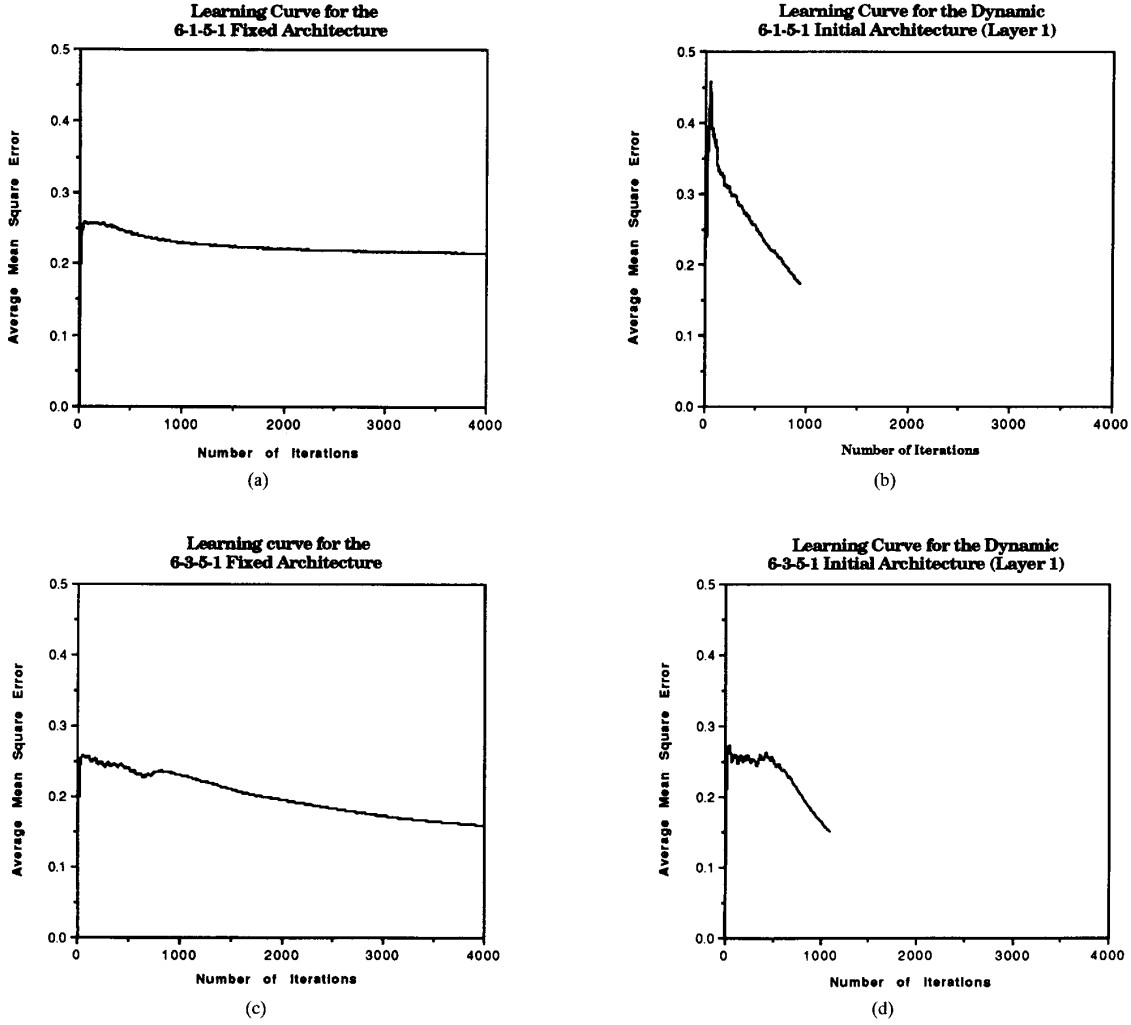


Fig. 4. The AMSE for the multiplexer problem, cases 1 and 2.

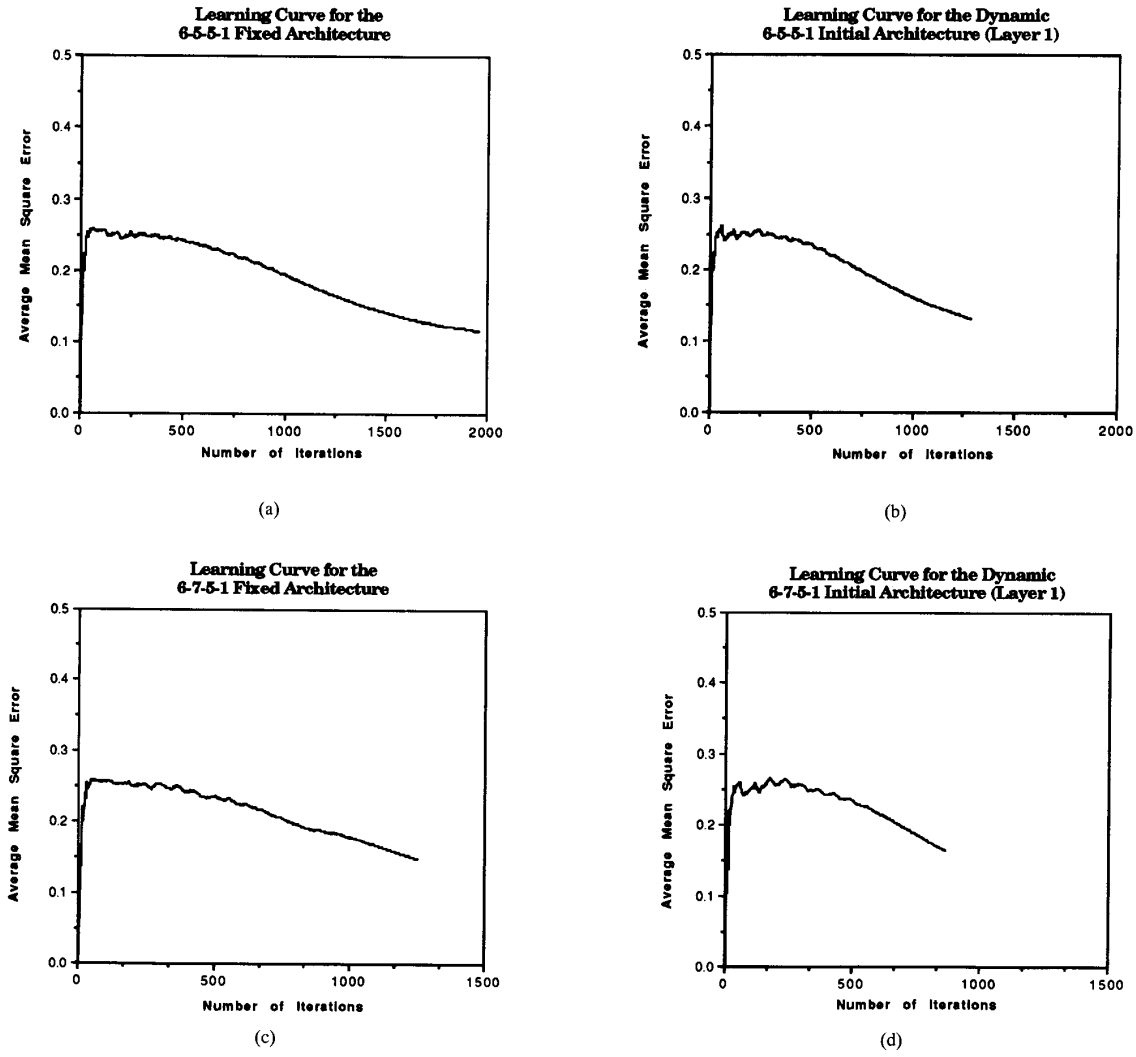
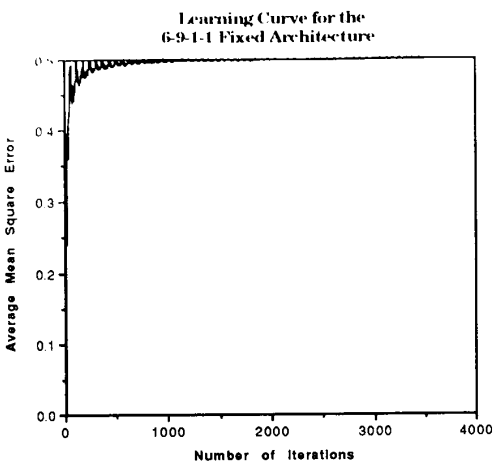


Fig. 5. The AMSE for the multiplexer problem, cases 3 and 4.

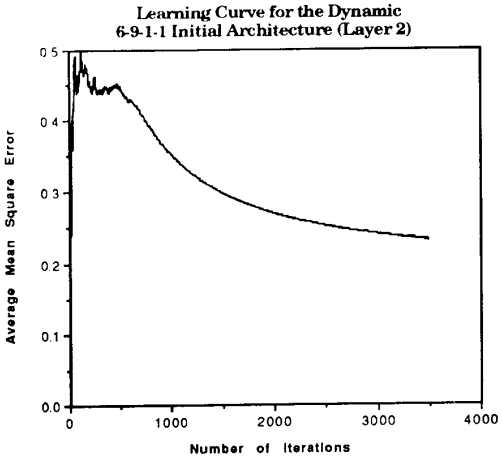
increase and consequently the network never converged. This implies that the addition of nodes to the upper (closer to the outputs) hidden layer affects the AMSE at the output layer more quickly than when the nodes are added to the lower (closer to the inputs) hidden layer. This behavior is expected since the nodes in the upper hidden layer have a filtering effect for the nodes added to the lower hidden layers, whereas there is no buffering layer for the case when the nodes are added to the upper hidden layer. To overcome this condition, it is useful to increase the iteration span for which the AMSE is monitored in order to reduce the total number of nodes created and to give the network enough time to "relax" into a more stable state before a new node is added. Otherwise, the network may end up with more added neurons than needed. Fig. (6e) shows the learning curve for case 7 which is the same dynamic architecture as in case 6 with the exception that the iteration span is chosen to be 200 instead of 40. As can be seen the network converged in only 903 iterations without exhibiting the same difficulty as in case 6. It appears that a good rule of thumb for determining a reasonable iteration span is to multiply the number of training samples by three. Another approach would be to alternate the addition of nodes amongst different hidden layers or to perform node deletion and addition at different layers.

B. Decoder Network

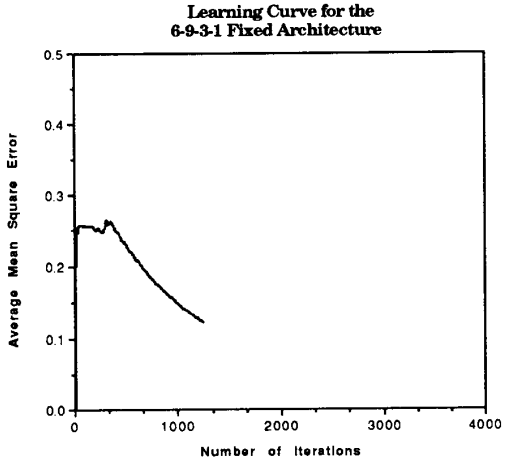
The decoder network was a two-layer neural network with three inputs and eight output nodes. The number of hidden layer nodes was varied. The three input lines activate one of the eight output lines using a binary code. The desired output is the binary decoded value of the three input lines. Again, the index of performance used was the AMSE at the output averaged over all the output nodes and the threshold value for the rate of change of AMSE was set to 0.001. Table II presents the results for the fixed and the variable



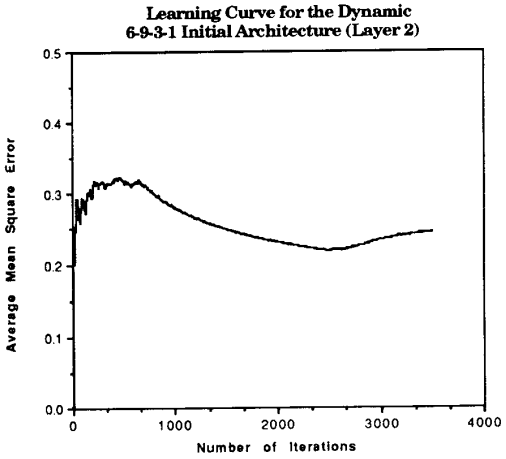
(a)



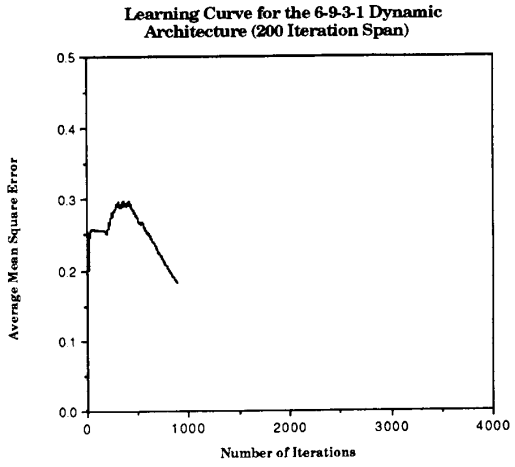
(b)



(c)

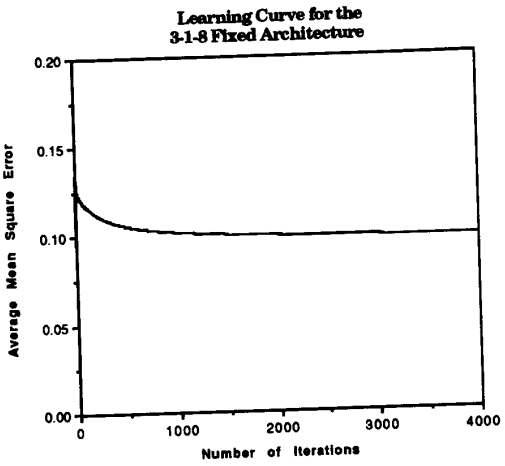


(d)

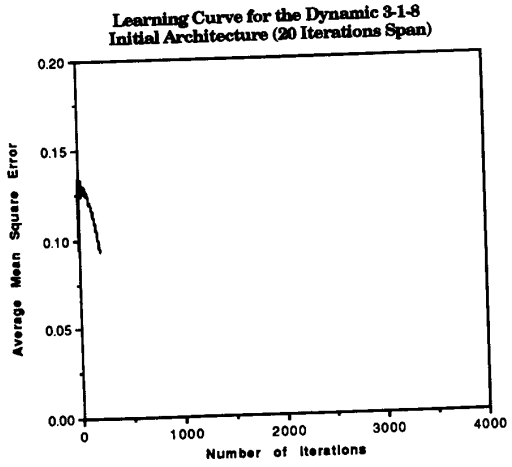


(e)

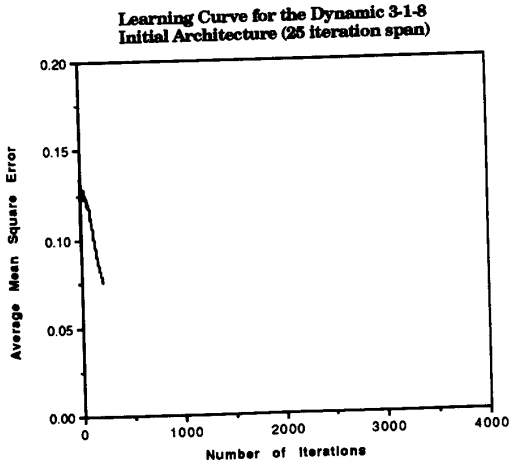
Fig. 6. The AMSE for the multiplexer problem, cases 5 through 7.



(a)



(b)



(c)

Fig. 7. The AMSE for the decoder problem, cases 1 and 2.

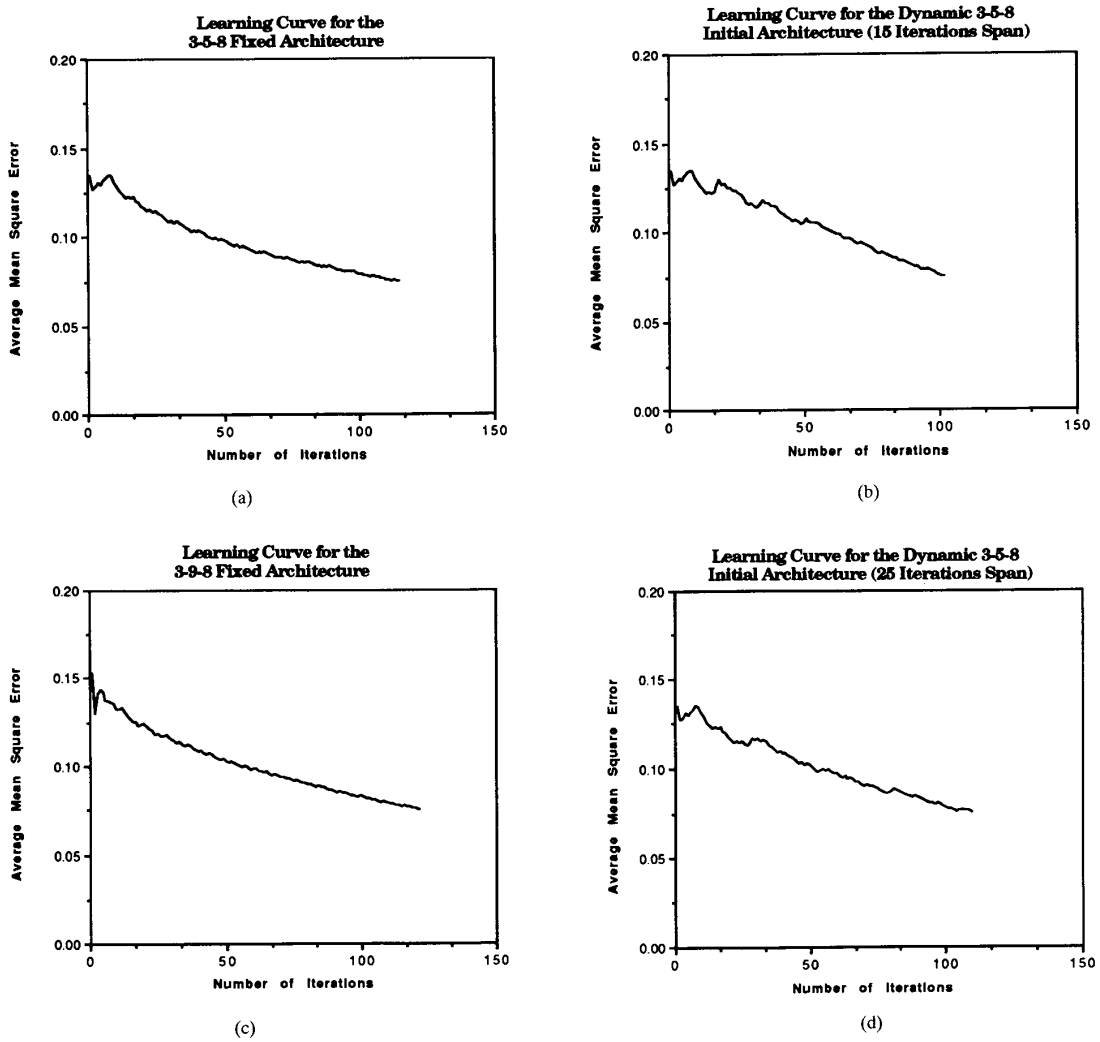


Fig. 8. The AMSE for the decoder problem, cases 3 through 5.

TABLE II
CONVERGENCE RESULTS FOR THE DECODER PROBLEM USING FIXED AND VARIABLE TOPOLOGIES

Case	Fixed Topology		Variable Topology	
	Initial	Iterations	Final	Iterations
1	3-1-8	5000 >	3-10-8 AMSE @ 20 Iterations	208
2	3-1-8	5000 >	3-9-8 AMSE @ 25 Iterations	209
3	3-5-8	115	3-11-8 AMSE @ 15 Iterations	102
4	3-5-8	115	3-9-8 AMSE @ 25 Iterations	110
5	3-9-8	122		

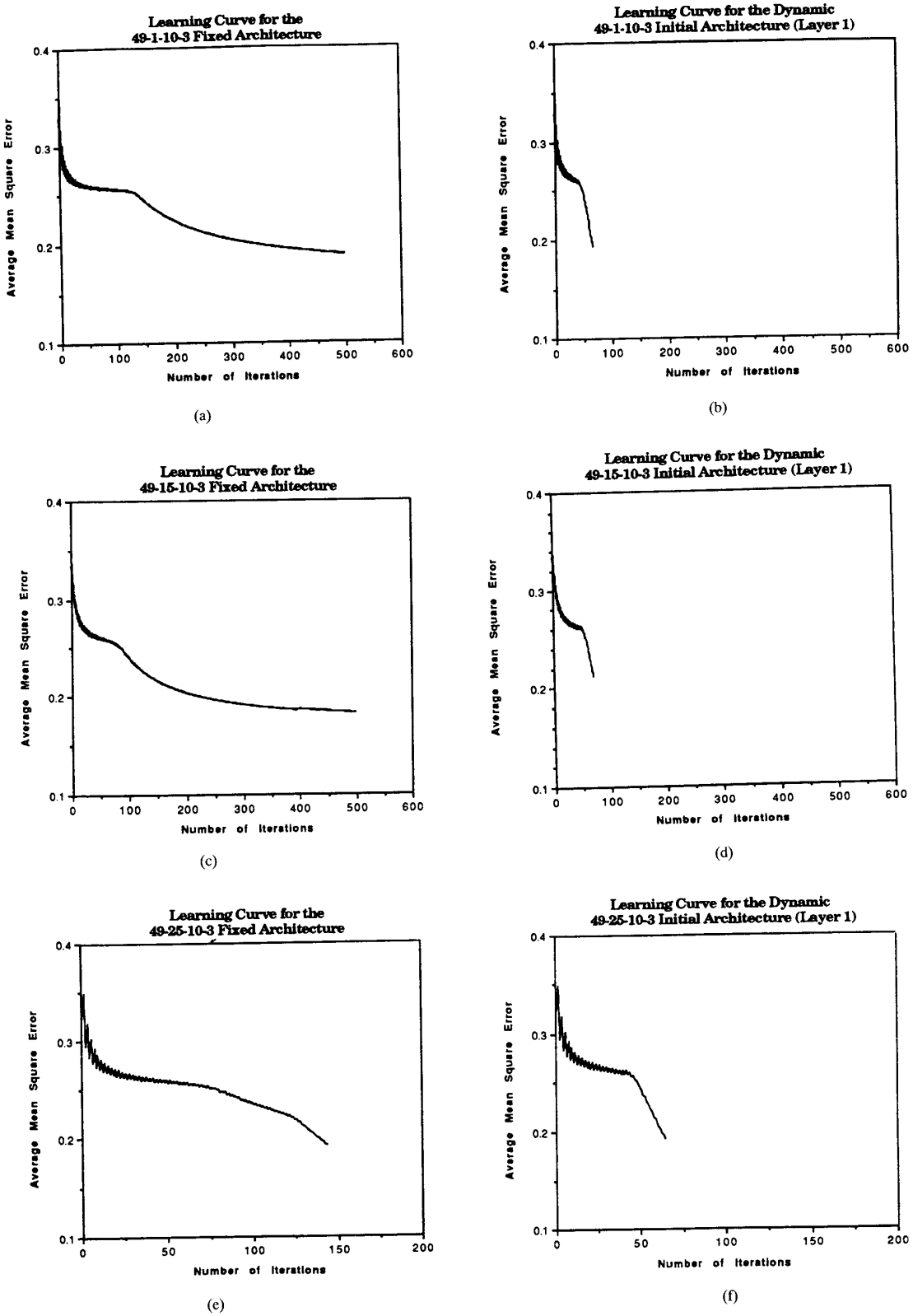


Fig. 9. The AMSE for target detection and classification problem, cases 1 through 3.

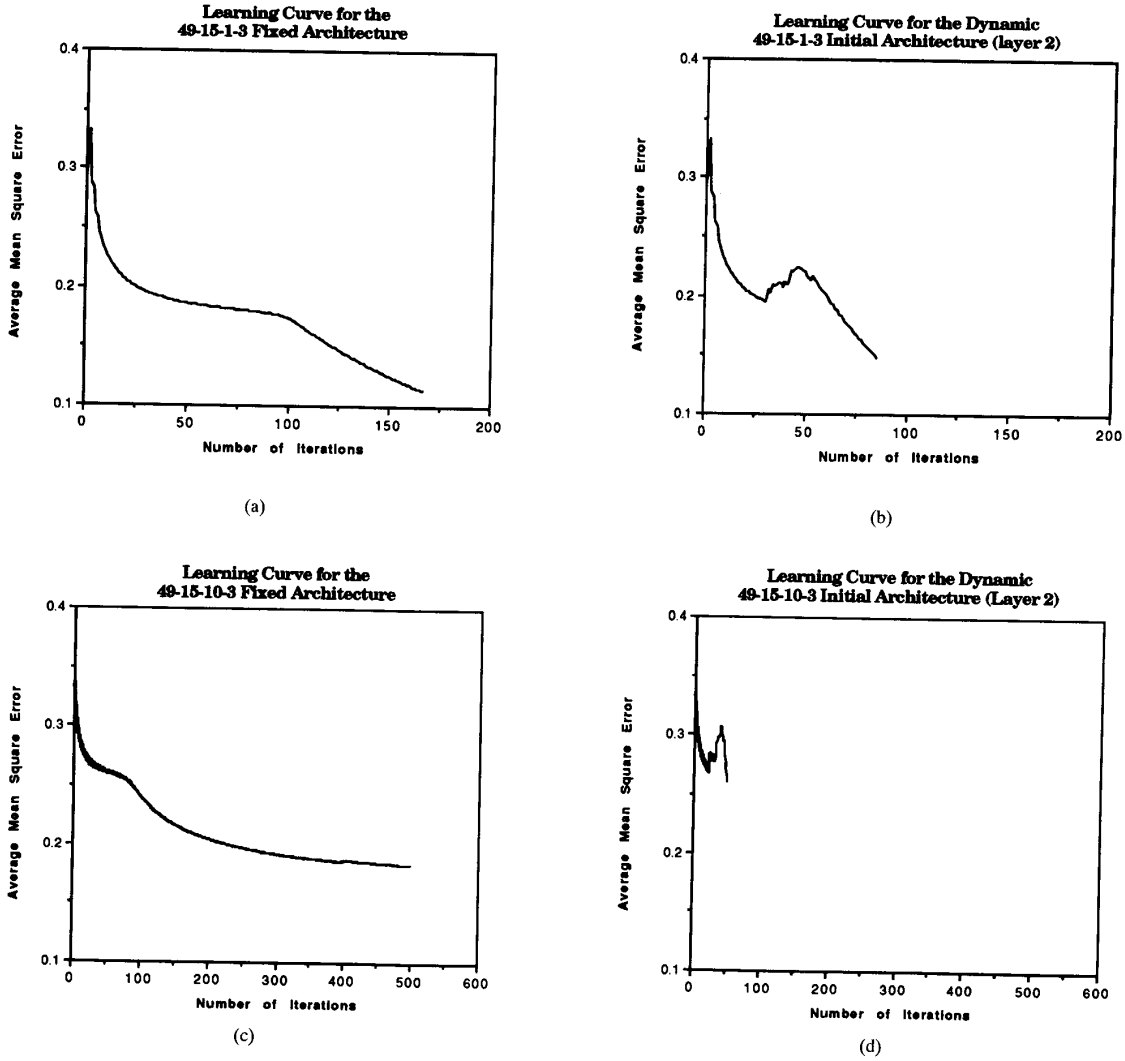


Fig. 10. The AMSE for target detection and classification problem, cases 4 and 5.

topologies. The corresponding learning curves for cases 1 and 2 are presented in Figs. (7a)–(7c) and those of cases 3 through 5 are presented in Figs. (8a)–(8d).

Comparing the convergence rates for the fixed and the corresponding variable architecture in cases 1 and 2 in Table II, indicates a drastic improvement when the nodes are dynamically added to the hidden layer. However, the results in cases 3 and 4 indicate only a minor improvement when the dynamic node creation method is employed. As discussed previously, this is due to the fact that in cases 3 and 4, the training process starts with a larger initial topology than the previous two cases and possibly closer to an "optimal" topology already, hence fewer additional nodes are needed. Comparing the results of the fixed topology in case 5 to the variable topology in case 4, which had the same final topology as the fixed architecture in case 5, indicates an interesting situation. That is, although the training process starts with a small network, the varied network topology finds a solution faster than the fixed network with identical final topology. This indicates the impact of changing the dimensionality and the form of the error space on both the training and the convergence rate of the network.

C. Target Detection and Classification

In this section we present the results of applying the proposed method for detection and classification of two types of targets, namely of Nylon and Wood compositions, from microwave data [15]. These targets were buried in dry loamy soil. The networks were trained for six targets and six backgrounds from each type of data. The input signal consisted of the amplitude of sensor data in windows of size 15×15. The method of principal components or Karhunen-Loeve (KL) transform [16] was used to reduce the amount of data without losing the accuracy in the representation. This was achieved by evaluating the two-sided covariances of the 2-D amplitude data in 7×7 grids within each 15×15 window. The covariances were used to form the doubly block Toeplitz covariance matrix

TABLE III
CONVERGENCE RESULTS FOR THE TARGET DETECTION AND CLASSIFICATION PROBLEM USING FIXED AND VARIABLE TOPOLOGIES

Case	Fixed Topology			Variable Topology		
	Initial	Iterations	Performance	Final	Iterations	Performance
Note: In cases 1 through 3 the nodes were added to the first hidden layer						
1	49-1-10-3	500 >	93/93/35	49-4-10-3	67	93/87/9
2	49-15-10-3	500 >	100/93/30	49-19-10-3	71	100/100/35
3	49-25-10-3	144	87/93/9	49-28-10-3	65	100/100/13
Note: In cases 4 and 5 the nodes were added to the second hidden layer						
4	49-15-1-3	167	100/93/13	49-15-4-3	86	100/93/35
5	49-15-10-3	500 >	100/93/30	49-15-13-3	51	87/100/9
Note: All entries under the performance are of the form: Detection Rate (%) / Classification Rate (%) / False-Alarm Rate (%)						

of the process. This matrix was then diagonalized using a unitary transformation to yield the eigenvalues associated with the data within each window. This resulted in 49 eigenvalues representing the most significant energy components of the data in an orthonormal signal space. These are used for both training and testing procedures of the neural networks.

Once the network is trained, the generalization of the network is tested by using the training data and the testing data (the data from other parts of the lanes which the network had not seen before). The testing data at each frequency consisted of 15 targets and 9 background windows for nylon data and 12 targets and 8 background windows for wood data. The initial architecture used was a 49–25–10–3 network architecture, i.e., 49 inputs, 25 first layer nodes, 10 second layer nodes, and the 3 output nodes which was determined empirically. The desired output sequences for nylon targets, wood targets and backgrounds were (1,0,0), (0,1,0), and (0,0,1), respectively.

The index of performance used to determine the trainability of the network was the AMSE at the output nodes. Table III summarizes the convergence results for the fixed and the variable topologies. The initial fixed network architectures were heuristically chosen based on our empirical studies. The architecture was held fixed and RLS-based algorithm in Section III-A was applied to train the network. In the variable topology cases, the network architecture was dynamically changed during the training process. The criterion for adding a node to a given layer was based upon monitoring the slope of the AMSE at the output layer after each 6 to 10 iterations. If the slope was below a selected threshold of 0.001, a new node was created; otherwise the RLS updating of the weights continued without interruption.

Figs. 9(a)–9(f) represent the learning curves for cases 1 through 3 and Figs. 10(a)–10(d) show the corresponding learning curves for cases 4 and 5 in Table III. As evident from these results, the dynamic node creation algorithm shows a drastic improvement in the convergence rate. It can be seen from these graphs that while the AMSE is oscillating around a flat portion of the error surface, upon addition of nodes, the error drastically decreases to its final value for convergence in a few iterations. Note that when the nodes were added to the first hidden layer only, no prominent overshoot was observed in the monitored AMSE at the output. However, when the nodes are added to the second hidden layer, as in cases 4 and 5, the overshoot is more observable. This is mainly attributed to the filtering operation performed at the upper layer as described before.

In addition, the performance of the network, in terms of the detection and classification rates and generalization properties, was relatively unchanged compared to the heuristically chosen fixed architectures. In particular, the performance in terms of detection and classification rates, was improved in cases 2 and 3 and was unchanged in case 4 when the variable topology method was employed. In case 1 a slight degradation in the classification rate was observed while the false-alarm rate was substantially reduced; and in case 5 a degredation in detection rate was seen while both classification and false-alarm rates were improved. This indicates the fact that the AMSE criterion alone may not necessarily lead to an optimum final architecture of the network as far as the detection, classification, and false-alarm rates are concerned. More research will be needed to address these issues. The proposed algorithm exhibits much better convergence rate than the standard LMS learning method [17].

V. CONCLUSION

The problem of simultaneous weight adaptation and node creation is considered in this paper. The projection updating method is utilized to arrive at recursive equations for both the weight updating and dynamic node creation during the training process. The vector-space interpretation for the RLS type algorithm allows the variation of the number of hidden layer nodes, and provides an optimal weight vector solution without requiring the prohibitive cost of retraining process. In this approach, it is possible to gain more insight into the internal behavior of the learning and the convergence

characteristics, since the error is monitored at each stage of the algorithm. The effectiveness of the algorithm is demonstrated on a real world application for detecting and classifying buried dielectric anomalies as well as the standard multiplexer and the decoder problems. Comparison of the simulation results indicates a significant improvement when compared to the LMS-based approach suggested in [8].

Also, the proposed scheme provides an optimal or near optimal topology in the least squares sense. It is also plausible that changing the shape of the error surface during the training process, will in effect, reduce the possibility of getting stuck in a local minima which is a prominent problem with all the LMS-based training algorithms. The simulation results demonstrate how the node creation method impacts the convergence rate especially when the nodes are added to the lower layers as the upper hidden layer has a filtering effect and suppresses the large transients which may cause instability in the learning process. The addition of nodes to the upper (closer to the outputs) impacts the convergence rate more quickly, however, this may cause oscillation and divergence. This can be overcome by increasing the iteration span and allowing the network to relax into a stable mode. The improvement in the convergence rate is more obvious when the training process starts with a small network topology, and the method is particularly effective for problems requiring complex decision regions (*hard learning*). There are a few open issues that need to be mentioned at this time. First, it seems that the rate of AMSE alone is not enough to determine exactly *when* and *where* to add the new nodes in order to obtain an optimal network topology while preserving its generalization property. Second, one can not be certain that the variable architecture method reaches the *same* solution as its fixed architecture counterpart.

REFERENCES

[1] K. Hornik and M. Stinchcombe, "Multilayer feedforward networks are universal approximators," manuscript, Dept. of Economics, University of California at San Diego, June 1988.

[2] D. E. Rumelhart and J. L. McClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge MA: MIT Press, vol. I, 1986.

[3] M. R. Azimi-Sadjadi and S. Citrin "Fast learning process of multi-layer neural nets using recursive least squares," presented at the Int. Joint Conf. Neural Networks (IJCNN '89), Washington, DC, June, 1989.

[4] M. R. Azimi-Sadjadi and R. J. Liou, "Learning process of multi-layer perceptron neural networks using recursive least squares technique," *IEEE Trans. Signal Processing*, vol. 40, pp. 446–450, Feb. 1992.

[5] R. Hetcht-Nielsen, "Theory of the backpropagation neural network," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, pp. 593–611. New York: IEEE Press, June 1989.

[6] J. K. Kruschke, "Improving generalization in back-propagation networks with distributed bottlenecks," in *Proc. IEEE Conf. Neural Networks*, San Diego, CA, June 1989, pp. I-443–447.

[7] E. D. Karnin "A simple procedure for pruning back-propagation trained neural network," *IEEE Trans. Neural Networks*, vol. 1, pp. 239–244, June 1990.

[8] T. Ash, "Dynamic node creation in back propagation networks," ICS Report 9802, Institute for Cognitive Science, University of California, La Jolla, CA, Feb. 1989.

[9] M. R. Azimi-Sadjadi, and S. Sheedvash "Recursive node creation in back-propagation neural networks using orthogonal projection method," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing, (ICASSP' 91)*, Toronto, Canada, May 1991, pp. 432–439.

[10] M. R. Azimi-Sadjadi, S. Sheedvash, and F. O. Trujillo, "A new approach for dynamic node creation in multi-layer neural networks," in *Proc. 1991 Int. Conf. Neural Networks (IJCNN '91)*, Singapore, Nov. 1991, pp. 2631–2638.

[11] J. M. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 304–337, Apr. 1984.

[12] S. T. Alexander, *Adaptive Signal Processing, Theory and Applications*. New York: Springer-Verlag, 1986.

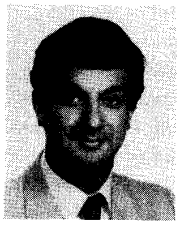
[13] S. Haykin, *Adaptive Filter Theory*. New York: Prentice Hall, 1986.

[14] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[15] M. R. Azimi-Sadjadi *et al.*, "Detection and classification of burried dielectric anomalies using a separated aperture sensor and a neural network discriminator," *IEEE Trans. Instrum. Meas.*, vol. 41, pp. 137–143, Feb. 1992.

[16] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

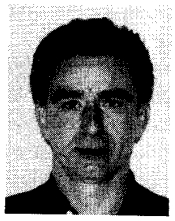
[17] S. Sheedvash "New approaches for fast learning and architectural variation in multi-layer neural networks," Ph.D. dissertation, Department of Electrical Engineering, Colorado State University, Fort Collins, CO, Dec. 1991.



Mahmood R. Azimi-Sadjadi (S'81–M'81–SM'89) was born in Tehran, Iran in 1952. He received the B.S. degree from the University of Tehran, Iran in 1977, the M.Sc. and the Ph.D. degrees from the Imperial College, University of London, England, in 1978 and 1982 respectively, all in electrical engineering.

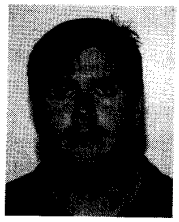
He served as an Assistant Professor in the Department of Electrical and Computer Engineering, University of Michigan-Dearborn. Since July 1986 he has been with the Department of Electrical Engineering, Colorado State University, where he is now an Associate Professor. His areas of interest are in digital signal/image processing, multidimensional system theory and analysis, adaptive filtering, system identification and neural networks. He is a coauthor of the book, *Digital Filtering in One and Two Dimensions*, Plenum Press, 1989.

Dr. Azimi-Sadjadi is the recipient of the 1990 BATTELLE Faculty Fellowship Award and the 1984 DOW Chemical Outstanding Young Faculty Award of the American Society for Engineering Education.



Sassan Sheedvash was born in Tehran, Iran, in 1950. He received the B.S. degree from the University of Utah, Salt Lake City, in 1974, the M.S. degree from the University of Arkansas, Fayetteville, AR, in 1975, and the Ph.D. degree from Colorado State University, Ft. Collins, in 1992, all in electrical engineering.

From 1975 to 1977 he worked at Bell Helicopter International as a development manager for the Logistics Department of the Ministry of Defense of Iran. He joined IBM in 1977 as Senior Associate Engineer. His current research interests involve neural networks, signal/image processing, pattern recognition, and artificial intelligence.



Frank O. Trujillo received the B.S. degree in electrical engineering with computer engineering concentration from Colorado State University, Fort Collins, CO, in 1991. He is currently working toward the M.S. degree at Colorado State University in computer and electrical engineering.

He has received a three year fellowship at CSU which covers his education from 1991 to 1994. He is a McNair Scholar, a member of SHPE, and became a member of Eta Kappa Nu and Tau Beta Pi in 1989 and 1990. His interests include neural network architectures, fast learning algorithms, stochastic processes, and object-oriented programming.