leigh fading channels using soft-decision decoding. The overall advantage of this system is that it allows the implementation of rate-selectable transmission systems that plays an important role in packet switching of speech where bit rate reduction is necessary due to heavy traffic. Since channel rate is constant, transmission becomes less complicated and regular modems can be used.

ACKNOWLEDGMENT

The authors would like to thank the reviewers of this correspondence for their many insightful comments and suggestions.

REFERENCES

[1] D. J. Goodman and C. E. Sundberg, "Combined source and channel coding for variable-bit-rate speech transmission," *Bell Syst. Tech. J.*, vol. 62, no. 7, pp. 2017-2036, Sept. 1983.
[2] P. J. Lee, "Construction of rate $(n-1)/n$ punctured convolutional codes with minimum required SNR criterion," *IEEE Trans. Commun.*, vol. COM-36, no. 10, pp. 1171-1174, Oct. 1988.
[3] D. J. Goodman and C. E. Sundberg, "Transmission errors and forward error correction in embedded differential pulse code modulation," *Bell Syst. Tech. J.*, vol. 62, no. 6, pp. 2735-2764, Nov. 1983.

Fast Learning Process of Multilayer Neural Networks Using Recursive Least Squares Method

Mahmood R. Azimi-Sadjadi and Ren-Jean Liou

***Abstract*—In this correspondence a new approach for the learning process of multilayer perceptron neural networks using the recursive least squares (RLS) type algorithm is proposed. This method minimizes the global sum of the squared errors between the actual and the desired output values iteratively. The weights in the network are updated upon the arrival of a new training sample and by solving a system of normal equations recursively. To determine the desired target in the hidden layers an analog of back-propagation (BP) strategy used in the conventional learning algorithms is developed. This permits the application of the learning procedure to all the layers. Simulation results on the 4-b parity checker and multiplexer networks are obtained which indicate significant reduction in the total number of iterations when compared with those of the conventional and accelerated back-propagation (ABP) algorithms.**

I. INTRODUCTION

In a supervised neural network such as the multilayer perceptron network [1]-[3], the choice of training algorithm, network architecture, and input signal representation plays a dominant role in the generalization and trainability characteristics. The choice of input signal representation determines the size of the network, the dimensionality of the weight space, and the transient behavior of the learning. A method must typically be employed to extract the principal components of the signal in order to reduce the volume of the data and also to orthogonalize (or decorrelate) the training data. The network architecture is another important consideration for optimal trainability and generalization. It is shown [4] that a three-layer perceptron neural network with sigmoidal-type nonlinearity at cells can approximate any arbitrary nonlinear function and generate any complex decision region needed for classification and recognition tasks.

The choice of the training algorithm, on the other hand, determines the rate of convergence to a solution, time required to reach the solution, and the optimality of the solution. The training process of the multilayer perceptron network is based upon the gradient descent method which minimizes the global sum of the squared errors between the actual and desired output values. If enough training samples and internal parameters are used, the input-output transformation may be defined to within an arbitrary accuracy. In this case, the performance of the network can approach to that of Bayes estimator which is optimal. Although the gradient descent algorithm in conjunction with the BP strategy is capable of providing reasonable estimates of the weights even for sigmoidal nonlinearities at nodes, the speed of the convergence of the process is relatively slow. Additionally, in multilayer networks this method may lead to local minima and thus nonoptimal solutions.

The development of a training algorithm using the RLS method for neural networks was first introduced in [5] and then later extended in [6]. The simulation results in these references indicated significant reduction in the total number of iterations owing mainly to the fast convergence characteristic of the RLS method which is independent of the signal statistics or the eigenvalue spread of the signal covariance matrix. Recently, several other new algorithms have been developed [7]-[9] for supervised training of multilayer perceptron networks. In this correspondence, a new fast recursive algorithm for the learning process of these networks is developed using an RLS-type algorithm. The weights are updated after each presentation of the training data. To apply the algorithm to the hidden layers, an analog of the BP scheme is developed and used to generate the desired values for the outputs of the nodes in these layers. Simulation results on the 4-b parity checker and the multiplexer networks are presented which indicate very fast convergence behavior of the proposed algorithm.

II. INPUT/OUTPUT RELATIONSHIPS FOR A MULTILAYER PERCEPTRON NEURAL NETWORK

Consider an M-layer perceptron neural network [1]-[4] as shown in Fig. 1. This network has N_0 inputs, N_r hidden layer nodes in layer $r \in [1, M-1]$, 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 r, by $y_s^{(r)}(t)$, $s \in [0, N_r - 1]$, and the outputs of this layer by $z_s^{(r)}(t)$. Then the total input to node s in layer r can be expressed as

$$y_s^{(r)}(t) = \sum_{p=0}^{N_{r-1}-1} w_{p,s}^{(r)}(t) z_p^{(r-1)}(t) = Z^{(r-1)^T}(t) W_s^{(r)}(t), \quad s \in [0, N_r - 1], \quad \forall r \in [1, M] \tag{1a}$$

where

$$W_s^{(r)}(t) := [w_{0,s}^{(r)}(t) w_{1,s}^{(r)}(t) \cdots w_{N_{r-1}-1,s}^{(r)}(t)]^T$$
$$Z^{(0)}(t) = [x_0(t) x_1(t) \cdots x_{N_0-1}(t)]^T$$
$$Z^{(r)}(t) := [z_0^{(r)}(t) z_1^{(r)}(t) \cdots z_{N_r-1}^{(r)}(t)]^T, \quad \forall r \in [1, M] \tag{1b}$$

where $w_{p,s}^{(r)}$ is the weight connecting node $p \in [0, N_{r-1} - 1]$ in layer $r-1$ to node $s \in [0, N_r - 1]$ in layer r, $r \in [1, M]$. The output of

Manuscript received December 24, 1990; revised May 24, 1991.
The authors are with the Department of Electrical Engineering, Colorado State University, Fort Collins, CO 80523.
IEEE Log Number 9104866.

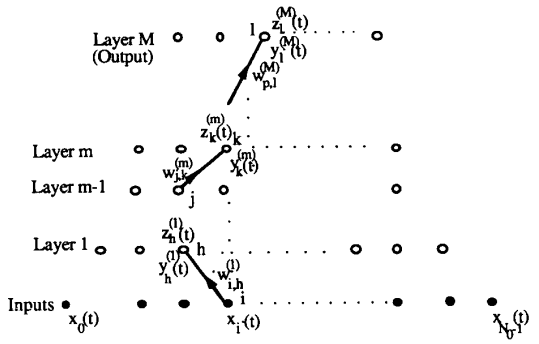

Fig. 1. A three-layer perceptron neural network.

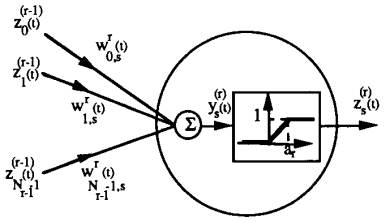

Fig. 2. A model for neurons (nodes) in the network.

this node, $z_s^{(r)}(t)$, has a real value that is normally a nonlinear function of the total input, $y_s^{(r)}(t)$. If the standard threshold logic-type nonlinearity shown in Fig. 2 is used, this output is given by

$$z_s^{(r)}(t) = \begin{cases} 0 & y_s^{(r)}(t) \le 0 \\ \dfrac{1}{a_r} Z^{(r-1)^T}(t) W_s^{(r)}(t) & 0 < y_s^{(r)}(t) < a_r \\ 1 & y_s^{(r)}(t) \ge a_r \end{cases} \tag{2}$$

where $1/a_r$ determines the slope of the ramp region. Note that similar results can be obtained when first-order approximated sigmoidal-type nonlinearity is used.

The aim of the learning procedure is to find an appropriate set of values for the weight vectors $W_s^{(r)}(t)$'s so that the global sum of the squared errors between the actual output, i.e., $z_l^{(M)}(t)$ and the desired output, $d_l^{(M)}(t)$, is minimized over the entire training set and all the output nodes, $l \in [0, N_M - 1]$. This index is given by

$$J = \tfrac{1}{2} \sum_t \sum_l \epsilon_l^{(M)^2}(t) \tag{3a}$$

where

$$\epsilon_l^{(M)}(t) := d_l^{(M)}(t) - z_l^{(M)}(t), \qquad l \in [0, N_M - 1]. \tag{3b}$$

The conventional learning algorithms [1]–[4] minimizes this index over the entire training set using the gradient descent method. Using this method the partial derivatives of J w.r.t. the weights in each layer are evaluated and used as corrections required to adjust the weights during the training process. This is accomplished once the entire set of the training samples is available. In this correspondence, due to the recursive nature of the process, the updating is accomplished at every training sample. The learning process using an RLS-type algorithm is derived in the following section.

III. Learning Procedure Using a Weighted RLS Type Algorithm

Consider a performance index J at iteration n which incorporates a limited memory, i.e.,

$$J(n) = \tfrac{1}{2} \sum_{t=1}^{n} \rho(n, t) \sum_{l=0}^{N_M - 1} \epsilon_l^{(M)^2}(t) \tag{4a}$$

where $\rho(n, t)$ is the variable weighting sequence which satisfies

$$\rho(n, t) = \lambda(n)\rho(n-1, t)$$
$$\rho(n, n) = 1. \tag{4b}$$

This means that we may write

$$\rho(n, t) = \prod_{j=t+1}^{n} \lambda(j). \tag{5}$$

In this correspondence we shall use a constant exponential weighting, i.e.,

$$\rho(n, t) = \lambda^{n-t} \tag{6}$$

where λ is a positive number less than but close to one which is called "forgetting factor."

For the weight vectors, $W_l^{(M)}(n)$'s, $l \in [0, N_M - 1]$, in the final (output) layer since the desired outputs are specified, the performance index can be minimized for $W_l^{(M)}(n)$ by taking the partial derivative of $J(n)$ w.r.t. $W_l^{(M)}(n)$ and setting it equal to zero. This gives

$$\frac{\partial J(n)}{\partial \hat{W}_l^{(M)}(n)} = 0 \Rightarrow \sum_{t=1}^{n} \lambda^{n-t} \left[\frac{\partial z_l^{(M)}(t)}{\partial \hat{W}_l^{(M)}(n)} \right] \cdot \epsilon_l^{(M)}(t) = 0 \tag{7}$$

where "$\hat{}$" represents the estimate of the relevant quantity. Note that it is assumed that the current estimate $\hat{W}_l^{(M)}(n)$ is used for $\forall\, t \in [1, n]$. Using (2) and (7) we obtain the following normal equation:

$$\sum_{t=1}^{n} \lambda^{n-t} (1/a_M) Z^{(M-1)}(t) \epsilon_l^{(M)}(t) = 0 \tag{8a}$$

or

$$\sum_{t=1}^{n} \lambda^{n-t} (1/a_M) Z^{(M-1)}(t) [d_l^{(M)}(t) - 1/a_M Z^{(M-1)^T}(t) \hat{W}_l^{(M)}(n)] = 0 \tag{8b}$$

which can be solved efficiently using the weighted RLS algorithm [11], [12]. Note that when the total input to node l is off the ramp region of the threshold logic nonlinearity function, the derivative in (7) is always zero. This implies that the normal equation in (8) will only be solved when the input to the relevant node lies within the ramp region; otherwise no updating is required. This is the case for all the other layers as will be shown later.

Similar normal equations for the other layers can be obtained by taking the partial derivative of $J(n)$ w.r.t. the weight vectors in these layers and then setting the results equal to zero. For the weight vector in layer m we have

$$\frac{\partial J(n)}{\partial \hat{W}_k^{(m)}(n)} = 0 \Rightarrow \sum_{t=1}^{n} \lambda^{n-t} \sum_{l=0}^{N_M - 1} \left[\frac{\partial \epsilon_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)} \right] \epsilon_l^{(M)}(t) = 0 \tag{9}$$

where $\hat{W}_k^{(m)}(n)$ is the estimate of the weight vector associated with node $k \in [0, N_m - 1]$ in the hidden layer m. If the current estimate, $\hat{W}_k^{(m)}(n)$, is used in place of $\hat{W}_k^{(m)}(t)$'s, $\forall t \in [1, n]$, using the chain rule we obtain

$$\frac{\partial \epsilon_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)} = -\frac{\partial z_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)}$$

$$= -\frac{\partial y_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)} \cdot \frac{dz_l^{(M)}(t)}{dy_l^{(M)}(t)} = -\frac{1}{a_M} \cdot \frac{\partial y_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)}. \quad (10)$$

Using (1a) we can write

$$\frac{\partial y_l^{(M)}(t)}{\partial \hat{W}_k^{(m)}(n)} = \sum_{p=0}^{N_{M-1}-1} \hat{w}_{p,l}^{(M)}(n) \frac{\partial z_p^{(M-1)}(t)}{\partial \hat{W}_k^{(m)}(n)} \quad (11)$$

which gives

$$\sum_{t=1}^{n} \lambda^{n-t} \sum_{p=0}^{N_{M-1}-1} \frac{\partial z_p^{(M-1)}(t)}{\partial \hat{W}_k^{(m)}(n)} \sum_{l=0}^{N_M-1} (1/a_M) \hat{w}_{p,l}^{(M)}(n) \epsilon_l^{(M)}(t) = 0. \quad (12)$$

Define

$$\epsilon_p^{(M-1)}(t) := \frac{1}{a_M} \sum_{l=0}^{N_M-1} \hat{w}_{p,l}^{(M)}(n) \epsilon_l^{(M)}(t) \quad (13)$$

which is the error propagated backward from the output to the nodes of hidden layer $M-1$ through the updated weights of layer M, i.e., $\hat{w}_{p,l}^{(M)}(n)$. This error is used to generate the desired outputs for the hidden layer $M-1$. Now, (12) becomes

$$\sum_{t=1}^{n} \lambda^{n-t} \sum_{p=0}^{N_{M-1}-1} \frac{\partial z_p^{(M-1)}(t)}{\partial \hat{W}_k^{(m)}(n)} \cdot \epsilon_p^{(M-1)}(t) = 0. \quad (14)$$

The procedure in (10)–(13) can be repeated for the term $\partial z_p^{(M-1)}(t)/\partial \hat{W}_k^{(m)}(n)$. Continuing this procedure up to layer m and back-propagating the errors to the lower layers yields

$$\sum_{t=1}^{n} \lambda^{n-t} \sum_{q=0}^{N_m-1} \frac{\partial z_q^{(m)}(t)}{\partial \hat{W}_k^{(m)}(n)} \cdot \epsilon_q^{(m)}(t)$$

$$= \sum_{t=1}^{n} \lambda^{n-t} \frac{\partial z_k^{(m)}(t)}{\partial \hat{W}_k^{(m)}(n)} \cdot \epsilon_k^{(m)}(t) = 0 \quad (15a)$$

where

$$\epsilon_k^{(m)}(t) := \frac{1}{a_{m+1}} \sum_{l=0}^{N_{m+1}-1} \hat{w}_{k,l}^{(m+1)}(n) \epsilon_l^{(m+1)}(t),$$

$$\forall m \in [1, M-1]. \quad (15b)$$

Here $\epsilon_l^{(m+1)}(t)$ represents the error sequence at the output of layer $(m+1)$ which is back-propagated through the updated weights of this layer, i.e., $\hat{w}_{k,l}^{(m+1)}(n)$, to generate $\epsilon_k^{(m)}(t)$, the error sequence at layer m. Note that (15a) is similar, in form, to (7). Again using (2) in (15a) the following normal equation can be obtained for this layer:

$$\sum_{t=1}^{n} \lambda^{n-t} (1/a_m) Z^{(m-1)}(t) \epsilon_k^{(m)}(t) = 0 \quad (16a)$$

or

$$\sum_{t=1}^{n} \lambda^{n-t} (1/a_m) Z^{(m-1)}(t) [d_k^{(m)}(t) - (1/a_m) Z^{(m-1)^T}(t) \hat{W}_k^{(m)}(n)] = 0 \quad (16b)$$

where

$$d_k^{(m)}(t) := z_k^{(m)}(t) + \epsilon_k^{(m)}(t). \quad (16c)$$

As a result, the weight updating in each layer leads to a normal equation that can be solved using the weighted RLS method [11], [12]. Let us now define the following correlation and cross-correlation matrices for the layer $m \in [1, M]$:

$$R^{(m)}(n) := \sum_{t=1}^{n} \lambda^{n-t} (1/a_m)^2 Z^{(m-1)}(t) Z^{(m-1)^T}(t)$$

$$\Delta_k^{(m)}(n) := \sum_{t=1}^{n} \lambda^{n-t} (1/a_m) Z^{(m-1)}(t) d_k^{(m)}(t). \quad (17)$$

Then (16b) can be written as

$$R^{(m)}(n) \hat{W}_k^{(m)}(n) = \Delta_k^{(m)}(n), \qquad \forall m \in [1, M]$$

$$\text{and } \forall k \in [0, N_m - 1]. \quad (18)$$

This normal equation can be solved without performing any matrix inversion operation using the weighted RLS method. The equations for updating the weight vectors $\hat{W}_k^{(m)}(n)$'s, $k \in [0, N_m - 1]$, can be derived using the matrix inversion lemma [11], [12] as

$$K^{(m)}(n) := P^{(m)}(n-1) Z^{(m-1)}(n) / [\lambda + Z^{(m-1)^T}(n) \cdot P^{(m)}(n-1) Z^{(m-1)}(n)] \quad (19a)$$

$$P^{(m)}(n) = \lambda^{-1} [I - K^{(m)}(n) Z^{(m-1)^T}(n)] P^{(m)}(n-1), \qquad \forall m \in [1, M] \quad (19b)$$

$$\hat{W}_k^{(m)}(n) = \hat{W}_k^{(m)}(n-1) + K^{(m)}(n) [d_k^{(m)}(n) - (1/a_m) \cdot Z^{(m-1)^T}(n) \hat{W}_k^{(m)}(n-1)], \qquad \forall k \in [0, N_m - 1] \quad (19c)$$

where $P^{(m)}(n) := R^{(m)^{-1}}(n)$, and $K^{(m)}(n)$ is the gain matrix for layer m.

Summary of the Algorithm:

The updating procedure at every iteration, i.e., at each training sample involves the following steps.

Step 1: Propagate the input signal through the network in the forward direction to obtain the actual outputs for each training signal using (1) and (2). At each layer, form the vector $Z^{(m-1)}(t)$, $m \in [1, M]$.

Step 2: Generate the error signal at the output of each layer and for each node. At the output layer this error is simply formed by comparing the actual outputs with the desired signal using (3b). For the other layers the error signals at the output layer are propagated backward, using (15b), through those layers with updated weights until the errors at the outputs of the lower layer (say layer m) with weights to be updated are generated.

Step 3: Compute matrices $P^{(m)}(n)$ and $K^{(m)}(n)$ for layer m using (19a,b).

Step 4: Determine the state of the node k, $\forall k \in [0, N_m - 1]$, in this layer. If the input to this node is within the ramp region proceed; otherwise there is no need for weight updating; thus increment k and examine the next node.

Step 5: Update the weight vector $\hat{W}_k^{(m)}(n)$ using the recursive equation (19c). Increment k and repeat steps 4 and 5 for the next node until all the weight vectors in this layer are updated.

These steps are performed for all the layers several times for a given training set until the error converges to within an acceptable range.

IV. Implementation and Results

The performance of the proposed algorithm was tested for training of the 4-b parity checker and multiplexer networks. The results are given in the following.

A. Example 1: 4-b Parity Checker

This network, which determines the parity of 4-b sequences [2], has 4 input nodes, 4 hidden nodes, and a single output node. There is one additional hidden layer node and one input node whose input values are always equal to 1. The weights emanating from these nodes thus act as threshold values for their respective superior layers. There were 16 possible training samples for which the network would have to determine the proper parity which would be the value at the output. The values of a_1 and a_2 were chosen to be 1. For each run, the first three weights in the lower and upper layers were initialized randomly in the range of -0.5 to -0.1 and 0.1 to 0.5. The fourth weight was set equal to the negative of the sum of the prior three weights. The threshold weights were initialized to 0.75. The initial condition for matrices $P^{(2)}$ and $P^{(1)}$ are $P^{(2)}(0) = P^{(1)}(0) = 0.5\ \mathrm{I}$.

Table I summarizes the results for this network and compares it with those of the standard BP learning algorithm [2]. The performance is highly dependent on how the weights are initialized. If the weights were too small to start with, it would take an excessive number of iterations for the weights to get large enough. A constant exponential weighting with $\lambda = 0.985$ was used. If λ was too small, $z_k^{(1)}$ and $z_1^{(2)}$ would tend to move rapidly off the ramp. This was especially prevalent in the hidden layer. If λ was too large, updating was too slow and the system tended to get "stuck" in the local minima with output values all close to 0.5.

In standard BP case 1, initialization was done in a procedure analogous to that described above. In case 2, the weights were initialized to small values in region $[-0.05, 0.05]$ as suggested in [2]. The step size used was 0.5 and the momentum factor was 0.9 in both cases. As can be noted in the table, the mean number of iterations for convergence and the percent of runs that did converge for 100 runs using different weights for initialization is significantly better for the RLS case. Fig. 3 shows the convergence of the error to zero for a typical run for RLS case and Figs. 4 and 5 show those of two standard cases. From the graphs one can clearly observe the very fast convergence behavior of the proposed method.

In the above experiments the behavior of the network was studied for a fixed number of hidden layer nodes and the initial value of $P^{(m)}(0)$. An experiment was also conducted where different values for these parameters were chosen. The results for one hundred runs are reflected in Table II. In this study the initial weights were in the range of $[-0.5, 0.5]$ and the allowed error was 0.03. As can be seen, the best convergent behavior was achieved when the parameters were equal to those numbers in the last column of this table.

It was observed that the choice of $P^{(m)}(0)$ does not have a great impact on the results. However, smaller values, say 0.5, appear to provide slightly better results. In some situations, it was noted that after certain iterations of the training process, the elements of $P^{(m)}(n)$ matrices may either increase or decrease rapidly. The problem can be remedied by periodic resetting of matrix $P^{(m)}(n)$ every certain iterations. Unlike the signal processing and control applications [12] where resetting of matrix $P^{(m)}(n)$ is normally done when abrupt variations in the signal or the system are detected, in this case it is practically impossible to determine when the resetting process is required. However, our experiments indicate that resetting matrix $P^{(m)}(n)$ when the outputs are approaching rather rapidly to the desired values can improve the speed of convergence and also prevent the occurrence of large misadjustments.

B. Example 2: Multiplexer Network

The architecture for the multiplexer [10] is a two-layer network with 6 inputs, 6 hidden layer nodes, and a single output node. The

TABLE I
PERFORMANCE COMPARISON OF THE BP AND RLS ALGORITHMS

	RLS	Standard Case 1	Standard Case 2
No. of runs	100	100	100
% Convergence	30	11	18
Iterations (μ)	85.2	474.2	1033
Iterations (σ)	31.6	195.8	377.4
Max. allowable No. of iterations	200	2000	2000
CPU time/run (sec.)	12.5	25	25

μ: Mean of number of iterations.
σ: Standard deviation of number of iterations.

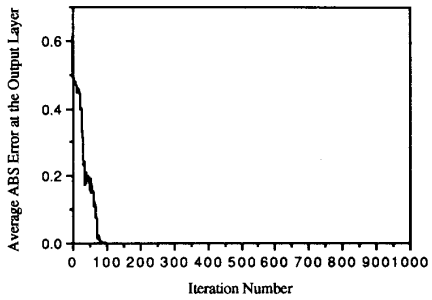

Fig. 3. Learning curve for RLS, 4-b parity checker. Mean absolute value of error over all training samples versus the number of iterations, RLS method.

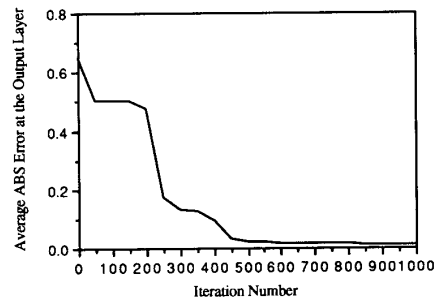

Fig. 4. Learning curve, back-propagation case 1, 4-b parity checker. Mean absolute value of error over all training samples versus the number of iterations, standard case 1.

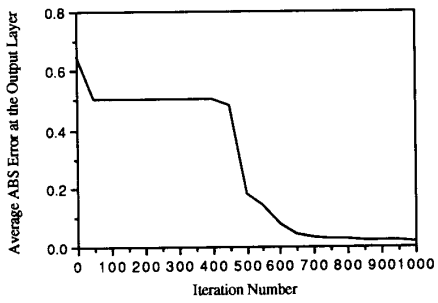

Fig. 5. Learning curve, back-propagation case 2, 4-b parity checker. Mean absolute value of error over all training samples versus the number of iterations, standard case 2.

TABLE II
EFFECTS OF PARAMETER VARIATIONS ON THE ALGORITHM PERFORMANCE

No. of hidden layer nodes	4	4	4	6	8
$P^{(m)}(0)$	0.5	0.5	0.2	0.2	0.2
% Convergence	45	31	40	91	100
Iteration (μ)	152.3	110.6	113.4	75.1	49.63
Iteration (σ)	76.48	43.28	40.62	28.67	17.73
Max. iterations	400	200	200	200	200

TABLE III
PERFORMANCE COMPARISON OF THE BP, ABP, AND RLS ALGORITHMS

	BP	ABP	RLS
No. of runs	25	25	25
% Convergence	100	100	100
Iterations (μ)	344.3	105.6	36.7
Iterations (σ)	485.2	20.3	11.2

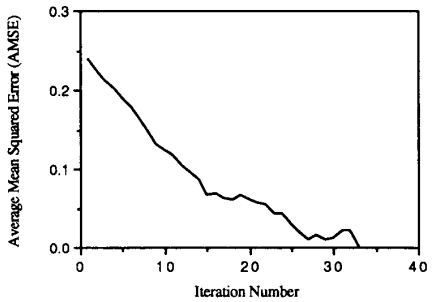

Fig. 6. Learning curve for RLS, multiplexer. Average mean squared error (AMSE) over all training samples versus the number of iterations, RLS method.

input to the network is four data lines and two address lines. The address lines activate one of the data lines using a binary code. The desired output is the input to the activated data line. The index of performance used to determine the trainability of the network is the average mean squared error (AMSE) at the output row averaged over all the output units. Table III shows the results of the RLS algorithm and compares them with those of the standard BP and the ABP in [10] for twenty-five trials. The fast convergence behavior of the RLS algorithm is demonstrated in Fig. 6 for one typical trial. It is interesting to note that the network trained using the proposed RLS converged to a smaller AMSE after only 34 iterations which took less than 50 s of CPU time on a MicroVAX Station 3600 computer environment.

V. CONCLUSION

In this correspondence a new scheme for fast learning of multilayer neural networks is proposed using the RLS method. A recursive procedure for updating the weights in these networks is derived when a threshold logic nonlinear activation function is employed at the nodes. The weights are updated upon the arrival of a new training sample. An analog of the BP scheme is developed which permits the application of the algorithm to the hidden layers. The effectiveness of the proposed algorithm is demonstrated on the 4-b parity checker and also the multiplexer networks. A comparison with the standard BP and the ABP learning algorithms is made which indicates substantial reduction in the number of iterations and also the computational time when the proposed training scheme is used.

REFERENCES

[1] D. E. Rumelhart and J. L. McClelland, "Parallel distributed processing: Explorations in the microstructure of cognition," vol. 1. Cambridge, MA: M.I.T. Press.
[2] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning by back-propagation," Int. Rep., Comput. Sci. Dep., Carnegie-Mellon Univ., June 1986.
[3] R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, pp. 4-22, Apr. 1987.
[4] R. Hecht-Nielsen, "Theory of back-propagation neural networks," in *Proc. IEEE Int. Conf. Neural Networks* (Washington DC), Jan. 1989, pp. I-593-605.
[5] M. R. Azimi-Sadjadi and S. Citrin, "Fast learning process of multilayer neural nets using recursive least squares technique," in *Proc. IEEE Int. Conf. Neural Networks* (Washington DC), May 1989.
[6] M. R. Azimi-Sadjadi, S. Citrin, and S. Sheedvash, "Supervised learning process of multilayer perceptron neural networks using fast recursive least squares," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing (ICASSP'90)* (New Mexico), Apr. 1990, pp. 1381-1384.
[7] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended Kalman algorithm," in *Advances in Neural Information Processing Systems I.* Morgan Kaufman, 1989.
[8] D. R. Hush and J. M. Salas, "Improving the learning rate of back-propagation with the gradient reuse algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, 1988, pp. I441-I447.
[9] F. Palmieri and S. Shah, "Fast training of multilayer perceptron using multilinear parametrization," in *Proc. IEEE Int. Joint Conf. Neural Networks* (Washington DC), Jan. 15-19, 1990.
[10] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.
[11] S. Haykin, *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
[12] G. C. Goodwin and K. S. Sin, *Adaptive Filtering, Prediction, and Control.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

Fast Calculation of the Choi-Williams Time-Frequency Distribution

Daniel T. Barry

***Abstract*—The Choi-Williams distribution (CWD) uses an exponential kernel in the generalized class of bilinear time-frequency distributions to achieve a reduction in the cross-term components of the distribution. Matrix manipulations provide an intuitive approach and, when combined with parallel processing, improve the processing speed to allow real-time calculations of the CWD.**

I. INTRODUCTION

Time-frequency distributions (TFD) are used to represent nonstationary signals. A common example from the generalized Cohen's class of distributions is the spectrogram. However, in the spectrogram, the window length determines the frequency space

Manuscript received January 25, 1990; revised June 27, 1991. This work was supported by NIH Grant NS-01710 and NSF Grant BCS-9000257.
The author is with the Department of PM&R, University of Michigan, Ann Arbor, MI 48109-0042.
IEEE Log Number 9104859.