A New Recurrent Neural Network Learning Algorithm for Time Series Prediction

P.G. Madhavan
Electrical Engineering & Computer Science Department
University of Michigan, Ann Arbor
1301 Beal Avenue, Ann Arbor, MI 48109, U.S.A.
e-mail: pgmadhav@eecs.umich.edu

ABSTRACT

A new recurrent neural network topology for the prediction of time series is developed. The back-propagation algorithm to train this network is derived. Such a network is called the Prediction Recurrent Artificial Neural Network (PRANN). The performance of the PRANN network is analyzed for linear and nonlinear time series. Performance comparison to time-delay neural networks shows a two-fold increase in performance for the PRANN.

KEYWORDS

neural networks, recurrent network topology, learning rule, time series prediction, nonlinear time series.

I. INTRODUCTION

In recent years, new artificial neural networks (ANNs) for temporal processing have been proposed including "recurrent" neural networks that have feedback connections. As distinct from the recurrent networks such as Hopfield networks which settle into stable states leading to associative memory networks, we consider neural networks with recurrent connections which allow the processing of time series (Werbos, 1990; Rumelhart *et al.*, 1986; Williams and Zipser, 1989).

One of the celebrated problems in time series analysis is that of prediction of future values from the knowledge of present and past values. A large body of work exists in the case of linear prediction (Makhoul, 1975). Neural networks have been employed in the prediction of linear time series

(Ukrainec *et al.*, 1989; Connor and Atlas, 1991; Weigend *et al.*, 1992; Madhavan *et al.*, 1996). In this article, we propose a multilayer perceptron network with recurrent connections and a complete back-propagation learning rule for updating the weights of the network. This network is developed based on the recurrent network of William & Zipser (1989), explicitly incorporating features which allow processing of time series. The network has arbitrary dynamics and the back-propagation algorithm is derived to minimize the output error using the approach in Haykin (1993).

II. DYNAMICS

The architecture of the PRANN is shown in figure 1 in the training mode. There is 1 output node, (1+N) hidden nodes and (1+M+N) input nodes of which M are external inputs. The nodes are numbered (p,q) such that p is the layer number and q is the node number in layer, p. Nodes with q=0 and 2 are linear nodes; nodes in the hidden layer are non-linear except the first. "D" is a delay unit where $D[h_j(n)] = h_j(n-1)$. The weights, w^{pq}_{ab} connect from layer a, node b to layer p, node q.

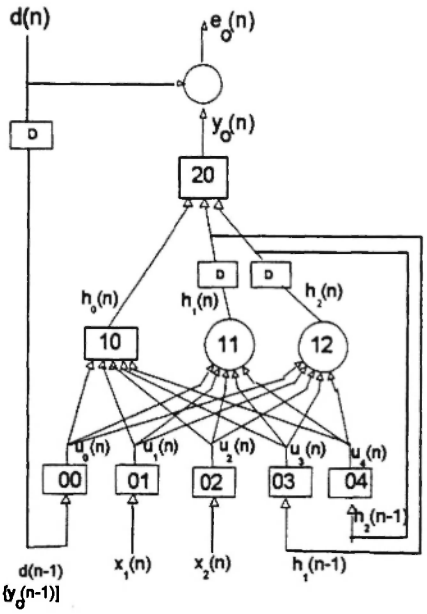

Fig. 1: Recurrent ANN; d(n-1) replaced by $y_0(n-1)$ during testing.

$$y_0(n) = w_{10}^{20}(n)h_0(n) + \sum_{j=1}^{N} w_{1j}^{20}(n)h_j(n-1) \quad (1)$$

$$e_0(n) = d(n) - y_0(n) \quad (2)$$

$$h_j(n) = \phi_j[v_j(n)] \quad (3)$$

where ϕ is the sigmoidal operator and $\phi'_j=1$ if j=0.

$$v_j(n) = \sum_{k=0}^{P-1} w_{ok}^{1j}(n)u_k(n) \quad (4)$$

where P=1+M+N

$$u_k(n) = \begin{cases} d(n-1) & for\ k=0;\ u_0(n)=y_0(n-1)\ during\ testing \\ x(n) & for\ k\in[external\ inputs, j=1,...,M] \\ h_j(n-1) & for\ k\in[hidden\ layer\ feedback, j=1,...,N] \end{cases} \quad (5)$$

Equations (1), (3), (4) and (5) define the entire dynamics of our recurrent ANN.

III. LEARNING RULE

The objective of our recurrent ANN is to minimize the *total error function*,

$$E_{tot} = \sum_n E(n); \quad where\ E(n) = \frac{1}{2}e_0^2(n)\ and\ e_0(n) = d(n) - y_0(n)$$

We utilize the method of steepest descent, which requires the knowledge of the following gradient with respect to synaptic weights, w.

$$\nabla_w E_{tot} = \frac{\partial E_{tot}}{\partial w} = \sum_n \frac{\partial E(n)}{\partial w} = \sum_n \nabla_w E(n) \quad (6)$$

To develop an algorithm that can learn to minimize E_{tot} in real time, we use $\nabla_w E(n)$ instead of $\nabla_w E_{tot}$ as an approximation to the method of steepest descent. Therefore, using this instantaneous estimate, the incremental weight change for a weight $w^{ij}_{kl}(n)$ can be written as

$$\frac{\partial E(n)}{\partial w_{kl}^{ij}(n)} = \frac{\partial}{\partial w_{kl}^{ij}(n)}\left[\frac{1}{2}e_0^2(n)\right] = -e_0(n)\,\frac{\partial y_0(n)}{\partial w_{kl}^{ij}(n)} \tag{7}$$

$$\Delta w_{kl}^{ij}(n) = -\eta\,\nabla_w E(n) = -\eta\frac{\partial E(n)}{\partial w_{kl}^{ij}(n)} \tag{8}$$

where equation (2) was used.

We will develop the update equations for output node weights and hidden node weights separately. First consider the output node weights, $w^{20}{}_{1j}$.

Output Node Weight Update

From equation (8), we have to develop an expression for the partial derivative of y0(n) with respect to w201j(n). From equation (1),

$$\frac{\partial y_0(n)}{\partial w_{1j}^{20}(n)} = \frac{\partial}{\partial w_{1j}^{20}(n)}\left[w_{10}^{20}(n)\;h_0(n) + \sum_{k=1}^{N} w_{1k}^{20}(n)\;h_k(n-1)\right] \tag{9}$$

Consider the 2 terms on the RHS of equation (9) separately. The first term,

$$\frac{\partial}{\partial w_{1j}^{20}(n)}\left[\,w_{10}^{20}(n)\;h_0(n)\,\right] = h_0(n)\frac{\partial w_{10}^{20}(n)}{\partial w_{1j}^{20}(n)} + w_{10}^{20}(n)\;\frac{\partial h_0(n)}{\partial w_{1j}^{20}(n)} = \delta_{j0}\;h_0(n) + w_{10}^{20}(n)\frac{\partial h_0(n)}{\partial w_{1j}^{20}(n)} \tag{10}$$

The term, $\partial w^{20}{}_{10}(n)/\partial w^{20}{}_{1j}(n) = 1$ when $j = 0$ and equal to 0 for $j \neq 0$. This is indicated by the Kronecker delta, δ_{j0} which is equal to 1 for $j = 0$ and zero otherwise. The second term on the RHS of equation (9) can be simplified using the derivative of product rule as before to give the following expression.

$$\frac{\partial}{\partial w_{1j}^{20}(n)}\left[\sum_{k=1}^{N} w_{1k}^{20}(n)\;h_k(n-1)\right] = \sum_{k=1}^{N}\left[\delta_{jk}h_k(n-1) + w_{1k}^{20}(n)\frac{\partial h_k(n-1)}{\partial w_{1j}^{20}(n)}\right] \tag{11}$$

Combining equations (9), (10) and (11), we get

$$\frac{\partial y_0(n)}{\partial w_{1j}^{20}(n)} = \delta_{j0}\, h_0(n) + w_{10}^{20}(n)\, \frac{\partial h_0(n)}{\partial w_{1j}^{20}(n)} + \sum_{k=1}^{N} \left[\delta_{jk} h_k(n) + w_{1k}^{20}(n) \frac{\partial h_k(n-1)}{\partial w_{1j}^{20}(n)} \right] \tag{12}$$

We can substitute for the partial derivatives of $h_k(n-1)$ by applying the chain rule of derivatives to equation (3). Rewriting equation (3), $h_j(n) = \phi_j[v_j(n)]$ where $\phi'_j = 1$ if $j = 0$, the partial derivative

$$\frac{\partial h_j(n)}{\partial w_{1j}^{20}(n)} = \frac{\partial h_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{1j}^{20}(n)} = \dot{\Psi}_j(n) \frac{\partial v_j(n)}{\partial w_{1j}^{20}(n)} \tag{13}$$

The last partial derivative is non-zero because of feedback. From equation (4), we know,

$$v_j(n) = \sum_{i=0}^{M+N} w_{0i}^{1j}(n)\, u_i(n)$$

$$\frac{\partial v_j(n)}{\partial w_{1j}^{20}(n)} = \sum_{i=0}^{M+N} \left[u_i(n) \frac{\partial w_{0i}^{1j}(n)}{\partial w_{1j}^{20}(n)} + w_{0i}^{j}(n) \frac{\partial u_i(n)}{\partial w_{1j}^{20}(n)} \right] = \sum_{i=1}^{N} \left[w_{0(i+M)}^{1j}(n) \frac{\partial h_i(n-1)}{\partial w_{1j}^{20}(n)} \right] \tag{14}$$

The simplification of equation (14) makes use of the following :- (i) the assumption that weights at different layers are not functions of each other and hence $\partial w^{1j}{}_{0k}(n)/\partial w^{20}{}_{1j}(n) = 0$ and (ii) as seen from equation (5), (1+M) elements of $u_i(n)$ are d(n-1) and $x_j(n)$ (desired response and external inputs respectively), which are not functions of weights. Therefore, only elements left in equation (14) are those corresponding to the delayed feedback inputs from the hidden layer, hi(n-1), which are functions of weights due to feedback and small step size.

We will substitute equation (14) in (13) and employ the following notation for compactness.

$$\Pi_j(n) = \frac{\partial h_j(n)}{\partial w_{1j}^{20}(n)}$$

$$\Pi_j(n) = \phi'_j(n) \sum_{i=1}^{N} \Pi_i(n-1)\, w_{0(i+M)}^{ij}(n) \tag{15}$$

Note that for j = 0, $\phi'_j(n) = 1$ and that $\Pi_j(0)$ = random since the network initial state (hidden layer node outputs) dependence on weights is unknown. Using equation (15) in (12), we get,

$$\frac{\partial y_0(n)}{\partial w_{1j}^{2c}(n)} = \delta_{j0}\, h_0(n) + w_{10}^{2c}(n)\, \Pi_0(n) + \sum_{k=1}^{N} \left[\delta_{jk} h_k(n) + w_{1k}^{20}(n) \Pi_j(n-1)\right] \tag{16}$$

Using equation (16) in equations (7) and (8),

$$\Delta w_{1j}^{20}(n) = \eta\, e_0(n) \left[\delta_{j0}\, h_0(n) + w_{10}^{20}(n)\, \Pi_0(n)\right] + \eta\, e_0(n) \sum_{k=1}^{N} \left[\delta_{jk} h_k(n) + w_{1k}^{20}(n)\, \Pi_j(n-1)\right]$$

Hidden Node Weight Update

Similar to equation (9), we develop the expression for the partial derivative of $y_0(n)$ with respect to, in this case, $w^{1j}{}_{0k}(n)$. From equation (1),

$$\frac{\partial y_0(n)}{\partial w_{0k}^{1j}(n)} = \frac{\partial}{\partial w_{0k}^{1j}(n)} \left[w_{10}^{20}(n)\, h_0(n) + \sum_{i=1}^{N} w_{1i}^{2c}(n)\, h_i(n-1)\right] \tag{17}$$

Using the derivative of product rule and using the assumption that weights in one layer are not functionally dependent on weights in another, we can simplify equation (17) as follows.

$$\frac{\partial y_0(n)}{\partial w_{0k}^{1j}(n)} = w_{10}^{20}(n) \frac{\partial h_0(n)}{\partial w_{0k}^{1j}(n)} + \sum_{i=1}^{N} \left[w_{1i}^{20}(n) \frac{\partial h_i(n-1)}{\partial w_{0k}^{1j}(n)} \right] \tag{18}$$

Similar to equations (13), we have

$$\frac{\partial h_j(n)}{\partial w_{jk}^{1k}(n)} = \frac{\partial h_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}^{1j}(n)} = \phi'_j(n) \frac{\partial v_j(n)}{\partial w_{0k}^{1j}(n)} \tag{19}$$

But from equation (4), we know,

$$v_j(n) = \sum_{l=0}^{M+N} w_{0l}^{1j}(n)\, u_l(n)$$

$$\begin{aligned} \frac{\partial v_j(n)}{\partial w_{0k}^{1j}(n)} &= \sum_{l=0}^{M+N} \left[u_l(n) \frac{\partial w_{0l}^{1j}(n)}{\partial w_{0k}^{1j}(n)} + w_{0l}^{1j}(n) \frac{\partial u_l(n)}{\partial w_{0k}^{1j}(n)} \right] \\ &= \sum_{l=0}^{M+N} \delta_{kl} u_l(n) + \sum_{m=1}^{N} \left[w_{0(m+M)}^{1j}(n) \frac{\partial h_m(n-1)}{\partial w_{0k}^{1j}(n)} \right] \end{aligned} \tag{20}$$

Here, we have made use of the fact that the desired response and external inputs are not functionally dependent on weights. Using equation (20) in (19) and the following notation,

$$\Omega_k^j(n) = \frac{\partial h_j(n)}{\partial w_{0k}^{1j}(n)}$$

we can write,

$$\therefore \Omega_k^j(n) = \phi'_j(n) \left[\sum_{l=0}^{M+N} \delta_{kl} u_l(n) + \sum_{m-1}^{N} \Omega_m^j(n-1)\, w_{0(m+M)}^{1j}(n) \right] \tag{21}$$

Note that for j = 0, $\phi'_j(n) = 1$ and that $\Omega_j(0)$ = random since the network initial state (hidden layer node outputs) dependence on weights is unknown. Then, we can rewrite equation (18) as follows.

$$\frac{\partial y_0(n)}{\partial w_{0k}^{1j}(n)} = w_{10}^{20}(n)\, \Omega_k^0(n) + \sum_{i=1}^{N} \left[w_{1i}^{20}(n) \Omega_k^i(n-1) \right] \tag{22}$$

Using equation (22) in equations (7) and (8),

$$\Delta w_{0k}^{1j}(n) = \eta\, e_0(n) \left[w_{10}^{20}(n)\, \Omega_k^0(n) + \sum_{i=1}^{N} w_{1i}^{20}(n)\, \Omega_k^i(n-1) \right]$$

IV. ALGORITHM

There are M external inputs and N non-linear nodes in the hidden layer.

Forward calculations:-

$$u_j(n) = \begin{cases} d(n-1) & \text{for } k=0;\ u_0(n)=y_0(n-1) \text{ during testing} \\ x_j(n) & \text{for } k \in [\text{external inputs}, j=1,\ldots,M] \\ h_j(n-1) & \text{for } k \in [\text{hiddenlayerfeedback}, j=1,\ldots,N] \end{cases} \quad \text{(A)}$$

$$v_j(n) = \sum_{k=0}^{P-1} w_{jk}^{ij}(n)\, u_k(n) \qquad \text{where } P=1+M+N \quad \text{(B)}$$

$h_j(n) = \phi_j[v_j(n)]$ where ϕ is the sigmoidal operator and
$h_j(n) = v_j(n)$ for $j = 0$ (C)

$$y_0(n) = w_{10}^{20}(n)\, h_0(n) + \sum_{j=1}^{N} w_{1j}^{20}(n)\, h_j(n-1) \quad \text{(D)}$$

Learning procedure:-

1. Initialization: For $n < 0$, $d(n)=x_j(n)=h_j(n)=0$ for all j;
$\Omega_j(n)= \Pi_j(n)=w^{ij}_{kl}(0)$ = uniformly distributed random numbers for all i,j,k,l.

2. For each time step, n –
$e_0(n) = d(n) - y_0(n)$ (E)

For j = 0 to N,

$$\Pi_j(n) = \phi'_j(n) \sum_{i=1}^{N} \Pi_i(n-1)\, w_{0(i+M)}^{ij}(n) \quad \text{(F)}$$

for $j = 0$, $\phi'_j(n) = 1$; for $j \neq 0$, $\phi'_j(n) = h_j(n)[1 - h_j(n)]$.

For j = 0 to N,

$$w_{1j}^{20}(n+1) = w_{1j}^{20}(n) + \eta\, e_0(n) \left[\delta_{j0}\, h_0(n) + w_{10}^{20}(n)\, \Pi_0(n) \right] + \eta\, e_0(n) \sum_{k=1}^{N} \left[\delta_{jk} h_k(n-1) + w_{1k}^{20}(n)\, \Pi_j(n-1) \right] \quad \text{(G)}$$

and $\delta_{pq} = 1$ for p = q and zero otherwise.

For j = 0 to N AND k = 0 to M+N,

$$\Omega_k^j(n) = \phi'_j(n)\left[\sum_{l=0}^{M+N} \delta_{kl} u_l(n) + \sum_{m=1}^{N} \Omega_m^j(n-1)\, w_{0(m+M)}^{1j}(n)\right] \quad \text{(H)}$$

for j = 0, $\phi'_j(n) = 1$; for j $\neq$ 0, $\phi'_j(n) = h_j(n)[1 - h_j(n)]$.

For j = 0 to N AND k = 0 to M+N,

$$w_{0k}^{1j}(n+1) = w_{0k}^{1j}(n) + \eta\, e_0(n)\left[w_{10}^{20}(n)\, \Omega_k^0(n) + \sum_{i=1}^{N} w_{1i}^{20}(n)\, \Omega_k^i(n-1)\right] \quad \text{(I)}$$

V. TIME SERIES PREDICTION

To utilize the recurrent ANN of figure 1 for the prediction of time series, u(n), during training, we set d(n) = u(n) and $x_1(n) = u(n-1)$. Then, $y_0(n)$ is the 1-step-ahead prediction of u(n). The network in figure 1 with this training regime will be called the "Prediction Recurrent Artificial Neural Network" (PRANN).

The transversal filter structure traditionally employed for time series prediction (Makhoul, 1975) is shown in figure 2. The one-step forward prediction problem is to predict the value of u(n) given M previous samples of u(n), where M is the order of the predictor. The measure of performance of the predictor is reflected in the variance of the prediction error, $f_M(n)$, after convergence. When the prediction is done using non-recurrent ANNs, the tapped-delay line outputs are connected to the input nodes of the ANN. Such neural network predictors are called Time Delay Neural Networks (TDNNs).

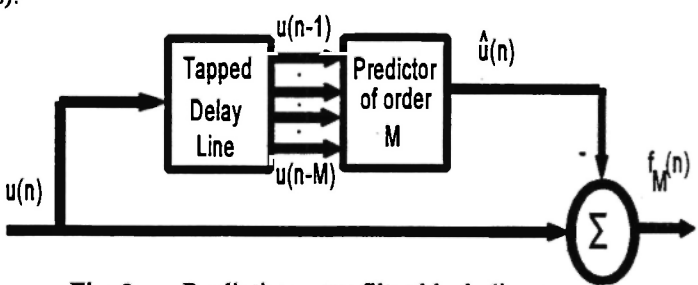

Fig. 2: Prediction error filter block diagram.

The performance of the PRANN and TDNN predictor structures will be examined for two different time series models: a linear auto-regressive model and a nonlinear Volterra model. It should be noted that the specific models chosen are for illustration purposes only and that similar results could be obtained with different models. In each case, the optimal predictor for the time series model can be found from solving the conditional expectation,

$$\hat{u}(n) = E[u(n|n-1, n-2, ...)]. \tag{23}$$

The linear time series model investigated was a second-order auto-regressive random process, given as,

$$u(n) = 0.804u(n-1) - 0.18u(n-2) + w(n), \tag{24}$$

where w(n) is a zero mean, unit variance white Gaussian noise process. The optimal predictor of this time series can be found from (23),

$$\begin{aligned} \hat{u}(n) &= E[0.804u(n-1) - 0.18u(n-2) + w(n)] \\ &= 0.804u(n-1) - 0.18u(n-2) \end{aligned} \tag{25}$$

The prediction error variance obtained when the optimal predictor is employed can be shown to be the variance of the underlying white noise process, w(n). Ideally, the two neural network predictors should be compared upon the amount of training required to reach this value. In practice, due to misadjustment, the amount of training required to reach the point where the convergence plot stabilizes will be the basis for comparison.

The PRANN used had 1 external input only, i.e., $x_1(n) = u(n-1)$ and 2 hidden layer non-linear nodes. The TDNN had 3 input nodes and 1 hidden layer with 7 nodes. The learning rates in both cases were 0.0001. The learning curves, showing the mean squared prediction error versus the number of iterations, are shown in figure 3, with the solid line representing the PRANN and the dotted line representing the TDNN. The PRANN converges to a lower prediction error of 0.6 compared to 0.82 for the TDNN predictor. The minimum prediction error is reached in approximately 3000 iterations for the PRANN whereas the TDNN takes up to 6000 iterations.

The Volterra nonlinear time series model was investigated next. The Volterra time series [6] provides a general expression for the relationship

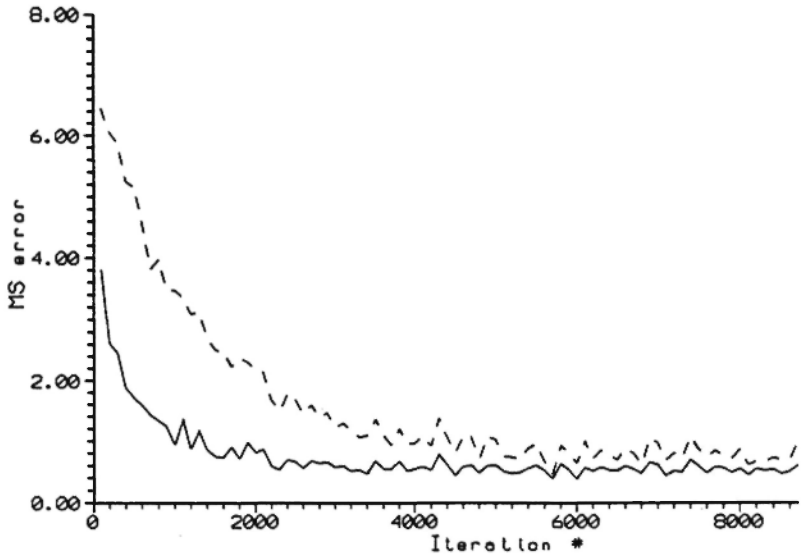

Fig. 3: Learning curve for linear time series; solid – PRANN; dotted – TDNN.

between the input and output of a nonlinear system, given below as,

$$u(n) = \sum_{i=0}^{\infty} a(i)\,w(n-i) + \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} a(i,j)\,w(n-i)\,w(n-j) + \ldots, \tag{26}$$

where w(n) is a zero mean, unit variance white Gaussian noise process and a(i), a(i,j), etc., represent the kernels of the nonlinear system. In this paper, the following model is used,

$$u(n) = w(n) + \Omega\; w(n-1)\,w(n-2), \tag{27}$$

where α is a constant chosen to be –0.75. This model can exhibit higher order behavior yet remain mathematically tractable. To illustrate, consider the optimal predictor of u(n), found by solving the conditional expectation in (23).

$$\begin{aligned} \hat{u}(n) &= E[w(n) + \Omega\; w(n-1)\,w(n-2)] \\ &= \Omega\; w(n-1)\,w(n-2) \end{aligned} \tag{28}$$

As can be seen, the optimal predictor is a cross-term between the delayed values of w(n). To express this result in terms of delayed samples of u(n) would require inversion of (27) and is a very difficult problem.

Given that the input process, w(n), was zero mean, the mean of the output process, u(n), is identically zero and the variance of the process can be shown [7] to be,

$$\sigma_u^2 = \sigma_w^2 + \Omega^2 \sigma_w^4 , \qquad (29)$$

where σ_w^2 is the variance of the white noise process. The prediction error variance obtained when this optimal predictor is employed will again be the variance of the white noise process. The neural network predictors will be compared below.

The PRANN used had 1 external input only, i.e., $x_1(n) = u(n-1)$ and 7 hidden layer non-linear nodes. The TDNN had 3 input nodes and 1 hidden layer with 7 nodes. The learning rates in both cases were 0.0001. The learning curves are shown in figure 4, with the solid line representing the PRANN and the dotted line representing the TDNN. The PRANN and the TDNN predictors converge to a prediction error of 1.15. The minimum prediction error is reached in approximately 3000 iterations for the PRANN where as the TDNN takes up to 5000 iterations.

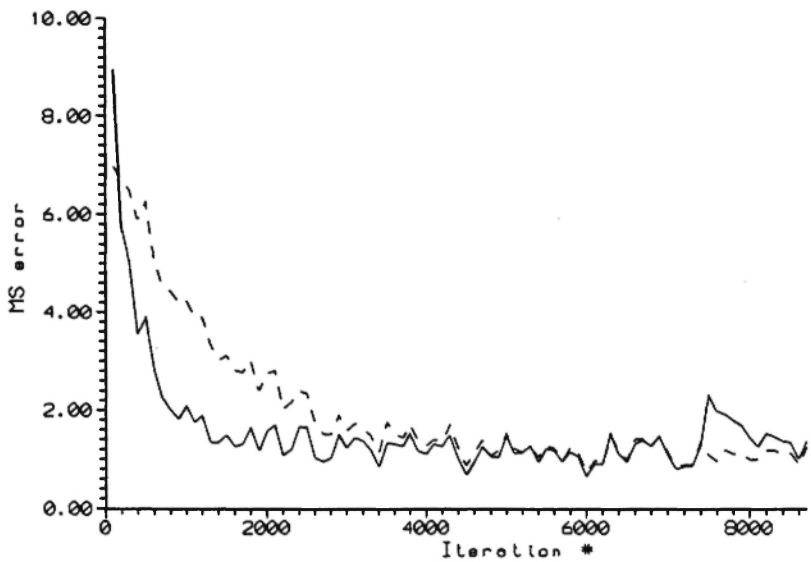

Fig. 4: Learning curve for nonlinear time series; solid – PRANN; dotted – TDNN.

VI. CONCLUSIONS

The PRANN has a specific network topology characterized by linear output node and one linear hidden layer node in addition to a structure similar to the recurrent neural network of William and Zipser (1989). We derived a back-propagation training algorithm for this network. Recasting the network proposed by William and Zipser (1989) into a topology similar to ours, it can be shown that they do not update weights connecting the nonlinear hidden layer nodes to the output node. Therefore, the PRANN can be seen as recurrent ANN specifically designed for time series processing with complete weight update.

The performance of the PRANN in comparison to the traditional TDNN shows that both in the linear and nonlinear time series case, the performance of the PRANN is superior. In the case of autoregressive models such as in equation (29), the output feedback allows delayed time series values to be represented internally to form the optimal predictor. The major limitation of the TDNN architecture is that it does not allow for the explicit representation of the past values. The uses of PRANN for a wide variety of time series models such as the bilinear and the soft threshold autoregressive models are currently being developed.

REFERENCES

1. Connor, J. & Atlas, L. 1991. Recurrent neural networks and time series prediction, *Proceedings of the Second International Joint Conference on Neural Networks*, Seattle, 1301-1306.
2. Haykin, S. 1991. *Neural Networks*, Macmillan, New York.
3. Madhavan, P.G., Stephens, B.E. & Low, W.C. 1996. Tri-state neural network and analysis of its performance, *International Journal of Intelligent Automation & Soft Computing*, 2, 1-8.
4. Makhoul, J. 1975. Linear prediction: A tutorial review, *Proceedings of IEEE*, **63**, 561-580.
5. Rumelhart, D.E., Hinton, G.E. & Williams, R.J. 1986. Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D.E. Rumelhart & J.L. McClelland (eds.), MIT Press, MA.

6. Schetzen, M. 1980. *The Volterra and Wiener Theories of Nonlinear Systems*, John Wiley and Sons, New York, NY.
7. Ukrainec, A., Haykin, S. & McGregor, J. 1989. A neural network nonlinear predictor, *Proceedings of the Second International Joint Conference on Neural Networks*, Washington, D.C., **2**, 622-625.
8. Weigend, A.S., Huberman, B.A. & Rumelhart, D.E. 1992. Predicting sunspots and exchange rates with connectionist networks, in *Nonlinear Modeling and Forecasting*, M. Casdagli and S. Eubank, (eds.), Addison Wesley, CA.
9. Werbos, P. 1990. Backpropagation through time: What it does and how to do it, *Proceedings of IEEE*, **78**, 1550-1560.
10. Williams, R.J. & Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, 270-280.