01 Jun 1994

A Recursive Least Squares Training Algorithm for Multilayer Recurrent Neural Networks

Q. Xu

K. Krishnamurthy
Missouri University of Science and Technology, kkrishna@mst.edu

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

Wen Feng Lu

Recommended Citation

Q. Xu et al., "A Recursive Least Squares Training Algorithm for Multilayer Recurrent Neural Networks," *Proceedings of the American Control Conference (1994, Baltimore, MD)*, vol. 2, pp. 1712-1716, Institute of Electrical and Electronics Engineers (IEEE), Jun 1994.
The definitive version is available at https://doi.org/10.1109/ACC.1994.752364

Proceedings of the
American Control Conference
Baltimore, Maryland • June 1994

A RECURSIVE LEAST SQUARES TRAINING ALGORITHM FOR MULTILAYER RECURRENT NEURAL NETWORKS

Q. Xu,[†] K. Krishnamurthy,[†] B. McMillin[‡] and W. Lu[†]
† Department of Mechanical and Aerospace Engineering and Engineering Mechanics
‡ Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401-0249

Abstract

Recurrent neural networks have the potential to perform significantly better than the commonly used feedforward neural networks due to their dynamical nature. However, they have received less attention because training algorithms/architectures have not been well developed. In this study, a recursive least squares algorithm to train recurrent neural networks with an arbitrary number of hidden layers is developed. The training algorithm is developed as an extension of the standard recursive estimation problem. Simulated results obtained for identification of the dynamics of a nonlinear dynamical system show promising results.

1. Introduction

Considerable attention is being focused on using neural networks for identification and control of dynamical systems [1]. Neural networks are attractive because they can be trained off-line with very high accuracy over a large input space without *a priori* knowledge of the system equations, and they can continue to learn (training and learning are used interchangeably here) during on-line application.

Past studies on neural networks have concentrated on feedforward neural networks because of the existence of well developed training algorithms. On the other hand, recurrent neural networks have received less attention because training algorithms/architectures have not been well developed. Considerable motivation exists in developing this because recurrent neural networks have the potential for better approximation ability, shorter training period, and wider range of dynamic behavior due to their dynamical nature.

In this study, a recursive least squares (RLS) algorithm is developed to train recurrent neural networks with an arbitrary number of hidden layers. The RLS training algorithm uses second derivative information of the error function and presumably will result in faster learning. Simulated results are presented for the identification problem of a nonlinear dynamical system to show the effectiveness of the training method.

2. Previous Research

Although feedforward neural networks have been used successfully to solve a wide variety of problems, they are not without problems. One of the major problems is in the slow convergence resulting in excessive training time. This has been confirmed by Hecht-Nielsen [2] who has shown that the error surface being minimized is very complicated with local minima and flat regions.

A number of higher order training algorithms have been presented to accelerate convergence. Parker [3], Watrous [4] and Becker and le Cun [5] have used some form of Newton's method to include second-order terms for learning. Kollias and Anastassiou [6] have developed an adaptive training algorithm based on an efficient implementation of the Marquardt-Levenberg least squares optimization technique. Singhal and Wu [7], Shah and Palmieri [8] and Jin, *et al.* [9] have used the extended Kalman filter algorithm to train feedforward neural networks.

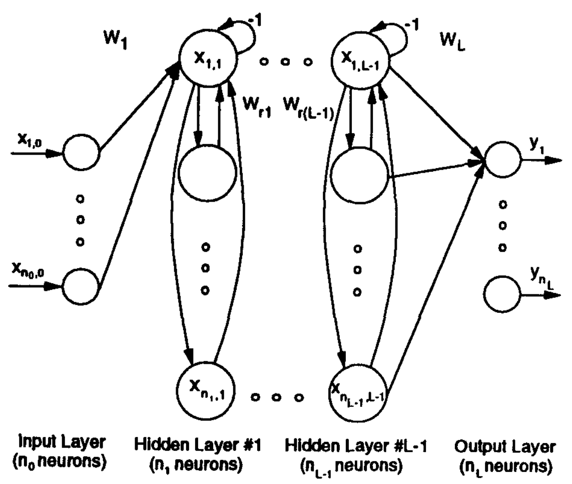
One common feature of the higher order training algorithms is that they are computationally intensive. The better the convergence property, the more intense are the computations. Thus the advantage gained in the need to present the neural network with the training set fewer times may be negated by the higher computational requirements. However, if the computations can be carried out in a parallel environment, there is potential for significantly decreasing the training time.

Various approaches to train recurrent neural networks have been presented. Rumelhart, *et al.* [10] have presented the general framework for this problem. The recurrent neural network is unfolded into a multilayer neural network that grows by one layer with each time step. Thus the storage and computational requirements for a long training sequence can be prohibitive. Pineda [11] has generalized the backpropagation technique to recurrent neural networks. This method requires a second dynamical system of the same size as the original system to implement the backward propagation equation in the weight update process. Pearlmutter [12] has extended Pineda's work to include time-dependent trajectories. Sudharsanan and Sundareshan [13] have recently presented an elegant approach which does not require the solution of a second dynamical system and results in simplified training rules. The three-layer architecture (one input layer, one hidden layer and one output layer) resembles that of feedforward neural networks. Puskorius and Feldkamp [14] and Ku and Lee [15] have used similar feedforward type architectures. In these two studies, a discrete time formulation was used compared to the networks in Refs. [11-13] which evolve continuously in time according to a set of coupled differential equations.

In this study, a RLS training algorithm is developed to train recurrent neural networks with an arbitrary number of hidden layers. The training algorithm is quite general, and is developed as an extension of the standard linear recursive estimation problem and is similar to the one obtained by Kollias and Anastassiou [6] using the Marquardt-Levenberg least squares optimization method. The RLS training algorithm uses second derivative information of the error function and presumably will result in faster learning. This study generalizes and extends the results presented in Refs. [11-15].

3. Recursive Least Squares Training Algorithm

As shown in Fig. 1, the input layer is layer 0 with n_0 neurons, layers 1 - $L-1$ are hidden layers with n_1 - n_{L-1} neurons, respectively, and the output layer is layer L with n_L neurons. The hidden layers form



W_1
$x_{1,1}$
-1
W_{r1}
$W_{r(L-1)}$
$x_{1,L-1}$
-1
W_L
$x_{1,0}$
$x_{n_0,0}$
y_1
y_{n_L}
$x_{n_1,1}$
$x_{n_{L-1},L-1}$
Input Layer (n_0 neurons)
Hidden Layer #1 (n_1 neurons)
Hidden Layer #L-1 (n_{L-1} neurons)
Output Layer (n_L neurons)

Figure 1: Schematic of a Multilayer Recurrent Neural Network

a dynamical neural network with sigmoidal processing elements. The dynamics of the network can be described by

$$\mathbf{T}_k\,\dot{\mathbf{x}}_k = -\mathbf{x}_k + \mathbf{W}_{rk}\,\mathbf{g}_k(\mathbf{x}_k) + \mathbf{W}_k\,\mathbf{x}_{k-1}, \quad k = 1, \ldots, L-1, \tag{1}$$

$$\mathbf{y} = \mathbf{W}_L\,\bar{\mathbf{x}}_{L-1}, \tag{2}$$

where $\mathbf{x}_0 \in \Re^{n_0}$ is the input vector, $\mathbf{x}_k = [x_{1,k}, x_{2,k}, \ldots, x_{n_k,k}]^T \in \Re^{n_k}$ is a vector describing the state of the neurons in the kth hidden layer, $\mathbf{g}_k(.) : \Re^{n_k} \rightarrow \Re^{n_k}$ is a vector-valued function with sigmoidal elements for the kth hidden layer, $\mathbf{W}_{rk} = [\mathbf{w}_{1,rk}, \mathbf{w}_{2,rk}, \ldots, \mathbf{w}_{n_k,rk}]^T$, $\mathbf{w}_{i,rk} = [w_{i1,rk}, w_{i2,rk}, \ldots, w_{in_k,rk}]^T \in \Re^{n_k}$ denote the intra-layer connection weights from neurons in the kth hidden layer to the ith neuron within the kth hidden layer, $\mathbf{W}_k = [\mathbf{w}_{1,k}, \mathbf{w}_{2,k}, \ldots, \mathbf{w}_{n_k,k}]^T$, $\mathbf{w}_{i,k} = [w_{i1,k}, w_{i2,k}, \ldots, w_{in_{k-1},k}]^T \in \Re^{n_{k-1}}$ denote the connection weights from neurons in the $(k-1)$th layer to the ith neuron in the kth layer, $\mathbf{T}_k = \mathrm{diag}[T_{1,k}, T_{2,k}, \ldots, T_{n_k,k}] \in \Re^{n_k \times n_k}$ is a diagonal matrix of time constants for the kth hidden layer, $\bar{\mathbf{x}}_{L-1}$ denotes the stable equilibrium state of the neurons in the $(L-1)$th layer for the input $\mathbf{x}_0$, $\mathbf{y} = [y_1, y_2, \ldots, y_{n_L}]^T \in \Re^{n_L}$ is the output vector, and the over dot denotes time derivative. Note that under steady-state conditions, Eq. (1) can be written as

$$\bar{\mathbf{x}}_k = \mathbf{W}_{rk}\,\mathbf{g}_k(\bar{\mathbf{x}}_k) + \mathbf{W}_k\,\bar{\mathbf{x}}_{k-1}, \; k = 1, \ldots, L-1, \tag{3}$$

where the input is now denoted by $\bar{\mathbf{x}}_0$, for convenience. It is assumed that for an input at the $(q-1)$th time instant, the neural network reaches steady state before the qth time instant. The finite amount of time allowed for the system to reach steady state is to facilitate, for example, calculation of the steady-state solution in real-time.

The problem is to find the connection weights such that the following error function is minimised.

$$ {}^{D}E = \sum_{d=1}^{D} \beta^{D-d} \sum_{n=1}^{n_L} [{}^{d}\hat{y}_n - {}^{d}y_n]^2, \qquad (4) $$

where β is a weight factor or forgetting factor allowing a higher weight for the last training pair, $\hat{y}_n$ and y_n are the desired and actual outputs of the nth neuron in the output layer, respectively, and the leading superscript denotes the training pair number. The error function in Eq. (4) can be minimised by setting its derivative with respect to $\mathbf{w}_{i,rk}$ ($i = 1,\ldots,n_k$, $k = 1,\ldots,L-1$) and $\mathbf{w}_{i,k}$ ($i = 1,\ldots,n_k$, $k = 1,\ldots,L$) to zero. A recursive method for solving these equations for the connection weights can be formulated in the following manner. Note that with the exception of the partial derivative of ${}^{D}E$ with respect to W_L, the other equations are nonlinear. Therefore, the standard recursive estimation can be employed directly to obtain the weight update rule for the neurons in the Lth layer. But for the other partial derivatives, the equations are first linearized about ${}^{D-1}\mathbf{w}_{i,rk}$ ($i = 1,\ldots,n_k$, $k = 1,\ldots,L-1$) and ${}^{D-1}\mathbf{w}_{i,k}$ ($i = 1,\ldots,n_k$, $k = 1,\ldots,L-1$). These weights minimize ${}^{D-1}E$ and are assumed to be known. Then, by following the procedure employed in standard recursive estimation, one can obtain the weight update rule for the remaining neurons of the network (see Xu, *et al.* [16] for details). The resulting recursive weight update rules for the neurons have the general form

$$ {}^{D}\mathbf{w} = {}^{D-1}\mathbf{w} + \eta\, {}^{D}\mathbf{k}\, {}^{D}\delta, \qquad (5) $$

where η is a small learning rate, as in gradient descent, the recursive update rule for the Kalman gain ${}^{D}\mathbf{k}$ and the approximate error covariance matrix ${}^{D}\mathbf{P}$ is given by

$$ {}^{D}\mathbf{k} = \frac{{}^{D-1}\mathbf{P}\, {}^{D}\mathbf{a}}{\beta + {}^{D}\gamma^2\, {}^{D}\mathbf{a}^T\, {}^{D-1}\mathbf{P}\, {}^{D}\mathbf{a}}, \qquad (6) $$

$$ {}^{D}\mathbf{P} = \frac{1}{\beta}[\mathbf{I} - {}^{D}\gamma^2\, {}^{D}\mathbf{k}\, {}^{D}\mathbf{a}^T]\, {}^{D-1}\mathbf{P}, \qquad (7) $$

and expressions for ${}^{D}\delta$, ${}^{D}\mathbf{a}$ and ${}^{D}\gamma^2$ for the various layers are as follows:

$\underline{{}^{D}\mathbf{w} = {}^{D}\mathbf{w}_{i,L} \; (i = 1,\ldots,n_L)}$

$$ \begin{aligned} {}^{D}\delta &= {}^{D}\hat{y}_n - {}^{D}y_n \\ {}^{D}\mathbf{a} &= {}^{D}\tilde{\mathbf{x}}_{L-1} \\ {}^{D}\gamma^2 &= 1 \end{aligned} $$

$\underline{{}^{D}\mathbf{w} = {}^{D}\mathbf{w}_{i,rk} \; (i = 1,\ldots,n_k,\ k = 1,\ldots,L-1)}$

$$ \begin{aligned} {}^{D}\delta &= \sum_{n=1}^{n_L} ({}^{D}\hat{y}_n - {}^{D}y_n)\, {}^{D}\tilde{h}_{ni,rk} \\ {}^{D}\mathbf{a} &= \mathbf{g}_k({}^{D}\tilde{\mathbf{x}}_k) \\ {}^{D}\gamma^2 &= \sum_{n=1}^{n_L} {}^{D}\tilde{h}^2_{ni,rk} \end{aligned} $$

$\underline{{}^{D}\mathbf{w} = {}^{D}\mathbf{w}_{i,k} \; (i = 1,\ldots,n_k,\ k = 1,\ldots,L-1)}$

$$ \begin{aligned} {}^{D}\delta &= \sum_{n=1}^{n_L} ({}^{D}\hat{y}_n - {}^{D}y_n)\, {}^{D}\tilde{h}_{ni,rk} \\ {}^{D}\mathbf{a} &= {}^{D}\tilde{\mathbf{x}}_{k-1} \\ {}^{D}\gamma^2 &= \sum_{n=1}^{n_L} {}^{D}\tilde{h}^2_{ni,rk} \end{aligned} $$

Here ${}^{d}\tilde{h}_{ni,rk}$ ($k = 1,\ldots,L-1$) are the steady-state solution of

$$ {}^{d}\dot{h}_{ni,rk} = -\,{}^{d}h_{ni,rk} + g'_{i,k} \sum_{l=1}^{n_k} w_{li,rk}\, {}^{d}h_{nl,rk} + \epsilon_k, \qquad (8) $$

where $\epsilon_k = \sum_{l=1}^{n_{k+1}} w_{li,k+1}\ {}^{d}h_{nl,r(k+1)}$, for $k = 1,\ldots,L-2$, $\epsilon_k = w_{ni,L}$, for $k = L-1$, and $g'_{i,k} = (\partial g_{i,k}({}^{d}x_{i,k})/\partial\, {}^{d}x_{i,k})|_{{}^{d}x_{i,k} = {}^{d}\tilde{x}_{i,k}}$. Note that it is possible to tailor the sigmoidal function such that $g'_{i,k}$ in Eq. (8) is small. Under this condition, Eq. (8) simplifies to ${}^{d}\tilde{h}_{ni,rk} = \epsilon_k$, and thus precludes the need for integrating Eq. (8), for example, to obtain the steady-state solution. This idea was exploited by Sudharsanan and Sundareshan [13] in deriving simplified learning rules for their recurrent neural network.

Training using the RLS algorithm is begun by initializing the $\mathbf{P}$-matrices to be equal to the identity matrix multiplied by a large constant. Then, for each training pair, Eq. (1) is integrated to obtain the stable equilibrium state of the network. Following this, Eq. (8) is integrated to obtain the steady-state ${}^{d}\tilde{h}_{ni,rk}$ ($k = 1,\ldots,L-1$) values. Finally, the $\mathbf{k}$-vector, $\mathbf{P}$-matrix and weights for each neuron are updated. After one pass through the training set, another pass is begun. This is repeated until the error at the output is within desirable bounds.

Due to the higher order nature, the RLS training algorithm is much more computationally intensive. For a single update of the network weights, the RLS algorithm requires exactly the same calculations of response and backpropagated errors as gradient descent. In addition, the RLS algorithm requires the

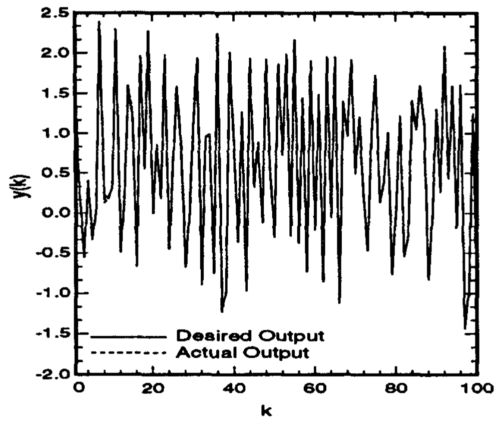


Figure 2: Response of the System

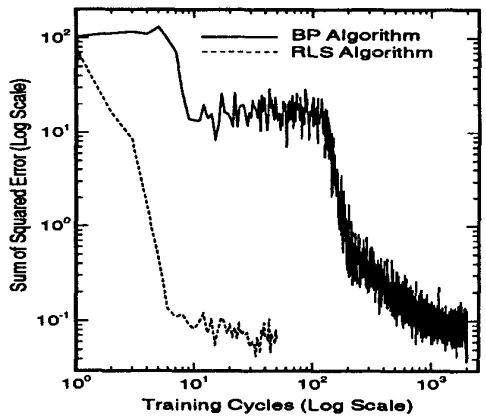


Figure 3: Learning Curves

calculation and storage of a **P**-matrix for each neuron in the network. Although these calculations are computationally intensive, an important point to note is that they can be done independently. This can be exploited in a parallel environment. In fact, in Ref. [17] it has been shown that the computation time of the RLS algorithm approaches that of standard backpropagation, the latter not being parallelizable, as more processors are applied to the matrix calculations in a multiple processor machine, such as the Intel iPSC/2 multicomputer.

Pineda [18] has noted that training a recurrent neural network requires $O(mN)$ calculations, where m is the number of time steps required to integrate the network differential equations, and N is the number of connection weights. The standard backpropagation training of a feedforward neural network, on the other hand, requires $O(N)$ calculations. Although training the recurrent neural network is computationally more intensive than a feedforward neural network, one can argue that the recurrent neural network will generally need to be presented with the training set fewer times. Thus the overall computing time will be less, resulting in faster learning. Of course, this can be further improved upon by solving the network differential equations and implementing the RLS training algorithm on a parallel computer.

4. Simulated Results

The effectiveness of the RLS training algorithm will be shown by solving the identification problem for a nonlinear dynamical system. The dynamical system considered is given by the difference equation [13]

$$y(k+1) = f[x(k), y(k)], \qquad (9)$$

where $x(.)$ is the input, $y(.)$ is the system output, and k is the discrete time instant. A recurrent neural network is trained to identify the unknown function $f[x(k), y(k)] = 1.1\sin(\cos(y(k))) + 1.5\,x(k)$. Random numbers between ± 1 were chosen for the input $x(k)$. Training sets of 100 points each were created. The recurrent neural network was trained using these training sets starting with random values between ± 0.1 for the connection weights, η=0.25, β=0.96, and ${}^0\mathbf{P}=10^3$ **I**. The forward and backward propagation equations were numerically integrated using a 4th-order Runge-Kutta method with zero initial values. The input layer included one bias neuron with its value set equal to 1 and two hidden layers with 3 neurons each were chosen. The diagonal elements of $\mathbf{T}_1$ and $\mathbf{T}_2$ were chosen to be 1/400 and the sigmoidal function chosen was $g(x) = -1 + 2/(1 + e^{-x})$. Figure 2 shows the desired and actual outputs for the tenth training cycle (set). As can be seen, the two curves are almost identical showing that the recurrent neural network has been trained to identify the nonlinear system dynamics. The sum of the squared error in this case was calculated to be 8×10^{-2}. Figure 3 shows that the sum of the squared error decreases rapidly during the first few training cycles.

To evaluate the performance of the recurrent neural network, the identification problem was solved by training a 4-layer feedforward neural network (3-6-5-1 neurons) using the standard backpropagation (BP) algorithm. The same parameters as in the recurrent neural network case were chosen. Figure 3 shows the training results. It is clear that a very large number of training cycles (in excess of 2000) are required to reduce the sum of the squared error to the level achieved by the recurrent neural network in a small number of training cycles.

5. Concluding Remarks

A recursive least squares algorithm to train recurrent neural networks with an arbitrary number of hidden layers is developed in this study. Simulated results obtained for identification of the dynamics of a nonlinear dynamical system show that the proposed training scheme could potentially reduce the training time considerably. Parallel implementation of the RLS training algorithm on an Intel iPSC/2 multicomputer is currently being investigated.

Acknowledgement

This work was supported by the National Science Foundation under Grant Number MSS-9216479 and the Missouri Department of Economic Development through the Manufacturing Research and Training Center-UMR.

References

[1] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, pp. 4-27, 1990.

[2] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. I-593 - I-607, 1989.

[3] D. B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *Proc. of the Int. Conf. on Neural Networks*, pp. II-593 - II-600, 1987

[4] R. L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods for Nonlinear Optimization," *Proc. of the Int. Conf. on Neural Networks*, pp. II-619 - II-628, 1987.

[5] S. Becker and Y. le Cun, "Improving the Convergence of Back-Propagation Learning with Second Order Methods," *Proc. of the Connectionist Models Summer School*, pp. 29-37, 1988.

[6] S. Kollias and D. Anastassiou, "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Trans. on Circuits and Systems*, Vol. 36, No. 8, pp. 1092-1101, 1989.

[7] S. Singhal and L. Wu, "Training Feed-Forward Networks with the Extended Kalman Filter," *Proc. of 1989 Int. Conf. on ASSP*, pp. 1187-1190, 1989.

[8] S. Shah and F. Palmieri, "MEKA - A Fast, Local Algorithm for Training Feedforward Neural Networks," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. III-41 - III-46, 1990.

[9] L. Jin, P. N. Nikiforuk and M. M. Gupta, "Decoupled Recursive Estimation Training and Trainable Degree of Feedforward Neural Networks," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. I-894 - I-900, 1992.

[10] D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Parallel Distributed Processing*, Vol. 1, MIT Press, Cambridge, Massachusetts, 1986.

[11] F. J. Pineda, "Dynamics and Architecture for Neural Computation," *Journal of Complexity*, Vol. 4, pp. 216-245, 1988.

[12] B. A. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks," *Neural Computation*, Vol. 1, pp. 263-269, 1989.

[13] S. L. Sudharsanan and M. K. Sundareshan, "Training of a Three-Layer Dynamical Recurrent Neural Network for Nonlinear Input-Output Mapping," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. II-111 - II-115, 1991.

[14] G. V. Puskorius and L. A. Feldkamp, "Model Reference Adaptive Control with Recurrent Networks Trained by the Dynamic DEKF Algorithm," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. II-106 - II-113, 1992.

[15] C.-C. Ku and K. Y. Lee, "Nonlinear System Identification Using Diagonal Recurrent Neural Networks," *Proc. of the Int. Joint Conf. on Neural Networks*, pp. III-839 - III-844, 1992.

[16] Q. Xu, K. Krishnamurthy, B. McMillin and W. Lu, "A Recursive Least Squares Training Algorithm for Multilayer Recurrent Neural Networks," Tech. Memo. MAE-TM-28, Dept. of Mech. & Aero. Eng. & Eng. Mech., University of Missouri-Rolla, 1993.

[17] J. E. Steck, B. McMillin, K. Krishnamurthy and G. Leininger, "Parallel Implementation of a Recursive Least Squares Neural Network Training Method on the Intel iPSC/2," *J. of Parallel and Distributed Computing*, Vol. 18, pp. 89-93, 1993.

[18] F. J. Pineda, "Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation," *Neural Computation*, Vol. 1, pp. 161-172, 1989.