Parallel and Separable Recursive Levenberg-Marquardt Training Algorithm

V.S. Asirvadam, S.F. McLoone and G. W. Irwin
Intelligent Systems and Control Research Group,
School of Electrical and Electronic Engineering,
The Queen's University of Belfast.
E-mail: s.mcloone@ee.qub.ac.uk

Abstract

A novel decomposed recursive Levenberg Marquardt (RLM) algorithm is derived for the training of feedforward neural networks. By neglecting inter-neuron weight correlations the recently proposed RLM training algorithm can be decomposed at neuron level enabling weights to be updated in an efficient parallel manner. A separable least squares implementation of decomposed RLM is also introduced. Experiment results for two nonlinear time series problems demonstrate the superiority of the new training algorithms.

1. INTRODUCTiON

Feedforward neural network training is normally defined as a squared-error cost function minimization problem with respect to the network parameters. This minimization can be performed in batch mode or online/recursive mode. In batch mode the cost function is explicitly defined over a batch of training data and minimized using gradient descent optimization procedures. Historically batch backpropagation (BBP), a first-order steepest descent method, was employed for training, but because of its poor convergence properties it has been superseded by more powerful second order techniques such as Conjugate Gradient, BFGS and Levenberg-Marquardt for most applications [1].

In recursive training mode network weights are updated at each iteration on the basis of an instantaneous cost function estimate using the current training data point. Early work on recursive training algorithms focused on stochastic backpropagation (SBP), the on-line version of the BBP routine. However, like its batch mode counterpart SBP suffers from slow training error convergence [2]. To address this deficiency several researches have proposed methods which exploit second order cost function information in a recursive framework. These include the recursive prediction error (RPE) algorithm by Chen and Billings [3] and the recursive Levenberg Marquardt algorithm (RLM) by Ngia *et al.* [4]. The former is the nonlinear equivalent of recursive least squares (RLS), while the later can be regarded as a nonlinear implementation of regularized RLS [5]. Chen *et al.* [6] describe how an efficient parallel RPE (PRPE) implementation can be achieved through a neuron-level decomposition of the covariance matrix used to recursively approximate the inverse of the cost function Hessian matrix. In experimental studies RLM has been found to have better convergence properties than RPE [7]-[8], however the availability of a parallel implementation of RPE makes it attractive for real-time applications.

Hybrid batch mode training algorithms yield significant performance gains for neural networks with linear output neurons [9],[10]. These algorithms, also referred to as separable least squares methods, operate by partitioning the weights space into linear and nonlinear subsets, corresponding to the output layer and hidden layer parameters respectively, so that they can be simultaneously optimised using the most appropriate techniques. Recently the authors have shown that a similar strategy can be used in recursive training algorithms. In particular separable versions of RPE and RLM have been developed and shown to be superior to their non-separable counterparts [8].

In this paper a parallel version of the RLM algorithm is presented. Then a novel separable parallel RLM implementation is derived. While both algorithms are more widely applicable, in the interest of conciseness they will be discussed here in the context of a multi input, single output, single hidden later Multilayer Perceptron with a linear output layer neuron. Such a network can be described by the equation

$$y = g(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j=1}^{N_h} w_j^o f\left(\sum_{i=1}^{N_i} w_{ij}^h x_i + b_j^h \right) + b^o \tag{1}$$

where $f(.)$ is a sigmoidal nonlinear function, N_i and N_h are the number of inputs and hidden layers neurons respectively and x_i is the ith element of the input vector $\boldsymbol{x}$. The various weights which make up the overall weight vector $\boldsymbol{w}$ can be partitioned into linear and nonlinear subsets $\boldsymbol{w}^L$ and $\boldsymbol{w}^{NL}$, that is $\boldsymbol{w} = [\boldsymbol{w}^{NL};\boldsymbol{w}^L]$ where

$$\boldsymbol{w}^{NL} = \left[w_{ij}^h\right], \ \boldsymbol{w}^L = \left[w_j^o\right] \quad \begin{array}{l} i = 0, \ldots N_i \\ j = 0, \ldots, N_h \end{array} \tag{2}$$

with $w_{0j}^h = b_j^h$ and $w_0^o = b^o$. Here w_j^o is the linear weight between the jth hidden layer and the output neuron, w_{ij}^h is the nonlinear weight between the ith input and the jth hidden layer neuron, b_j^h is the bias on the jth hidden layer neuron and b^o is the bias on the linear output neuron.

The remainder of the paper is organized as follows. Section 2 outlines the principle recursive training techniques. The parallel RLM algorithm is developed in section 3. Section 4 then outlines the new separable decomposed RLM algorithm. Simulation results for two benchmark problems are presented in section 5 and finally conclusions are provided in section 6.

2. RECURSIVE TRAINING

Recursive training schemes for MLPs attempt to minimize a cost function estimated instantaneously as

$$E(\boldsymbol{w}_t) \approx \varepsilon^2(\boldsymbol{w}_t) = (g(\boldsymbol{w}_t, \boldsymbol{x}_t) - d_t)^2 \tag{3}$$

where $\boldsymbol{w}_t$ is the network weights vector and $(\boldsymbol{x}_t, d_t)$ is the training data point at the t^{th} sample instant. In stochastic backpropagation (SBP) or gradient decent algorithm, the weights are updated according to the rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \lambda_t \nabla \varepsilon^2(\boldsymbol{w}_t) \tag{4}$$

where λ_t is the learning rate which can be fixed or time varying.

2.1. Recursive Prediction Error

The second order recursive least squares algorithm, which have been extensively studied for linear in-the-parameter models [11], can be extended to nonlinear models by linearizing about the current weights estimate $\boldsymbol{w}_t$. The resulting algorithm can be viewed as an instantaneous equivalent of Gauss-Newton optimization with the Hessian matrix approximated recursively at each iteration.

$$R_t = \alpha_t R_{t-1} + (1-\alpha_t)(\nabla y(\boldsymbol{w}_t) \nabla y^T(\boldsymbol{w}_t)) \tag{5}$$

$$0.97 \le \alpha_t \le 1 \qquad \nabla y(\boldsymbol{w}_t) = \left.\frac{\partial}{\partial \boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{x}_t)\right|_{\boldsymbol{w} = \boldsymbol{w}_t} \tag{6}$$

The recursive second order weight update is then given by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + R_t^{-1} \nabla y(\boldsymbol{w}_t) \varepsilon(\boldsymbol{w}_t) \tag{7}$$

The inverse of R at each iteration is computationally expensive to determine $((O(N_w^3))$ where $N = dim(\boldsymbol{w}))$ and normally the matrix inversion lemma (equation 8) is used to obtain a recursive formula for computing the inverse of R directly.

$$(A + BC)^{-1} = A^{-1} - A^{-1}B(I + CA^{-1}B)^{-1}CA^{-1} \tag{8}$$

The resulting weight update rules, usually referred as the recursive prediction error (RPE) algorithm, are given by

$$P_t = \frac{1}{\alpha_t}\left[P_{t-1} - \frac{P_{t-1}\nabla y(w_t)\nabla y^T(w_t)P_{t-1}}{\alpha_t + \nabla y^T(w_t)P_{t-1}\nabla y(w_t)}\right] \tag{9}$$

$$w_{t+1} = w_t + P_k \nabla y(w_t)\varepsilon(w_t) \tag{10}$$

Here the matrix $P_t = R_t^{-1}$ can be interpreted as the covariance matrix of weights estimate w_t.

2.2. Recursive Levenberg Marquardt

The RLM algorithm is derived by incorporating a regularization term in equation (5) of the RPE algorithm which then becomes

$$R_t = \alpha_t R_{t-1} + (1-\alpha_t)(\nabla y(w_t)\nabla y^T(w_t) + \rho I_{N_w}) \tag{11}$$

Unfortunately, the matrix lemma is now no longer practical and one partial but effective solution is to add ρ to one of the diagonal elements of $\nabla y(w_t)\nabla y^T(w_t)$ at a time as proposed by Ngia and Sjöberg [4][7]. Equation (11) can then be express as

$$R_t = \alpha_t R_{t-1} + (1-\alpha_t)(\nabla y(w_t)\nabla y^T(w_t) + \rho Z_{N_w}) \tag{12}$$

where Z_{N_w} is an $N_w \times N_w$ diagonal matrix with one non-zero diagonal element which changes from iteration to iteration as follows.

$$z_{ii} = 1 \quad \text{when } i = t \bmod(N_w)+1 \text{ and } t > N_w \tag{13}$$

$$z_{ii} = 0 \quad \text{otherwise} \tag{14}$$

With this modification the expression (12) can re-written in a concise form as

$$R_t = \alpha_t R_{t-1} + (1-\alpha_t)(\Omega(w_t)\Lambda_t^{-1}\Omega^T(w_t)) \tag{15}$$

where $\Omega(w_t)$ is a N_wx2 matrix with the first column corresponding to $\nabla y(w_t)$ and the second column consisting of a N_wx1 vector with one element set to 1 in accordance with (equation (13) and (14)) above.

$$\Omega^T(w_t) = \begin{bmatrix} & \nabla y^T(w_t) & \\ 0 \ \dots & 1 & \dots \ 0 \end{bmatrix} \quad \text{and} \quad \Lambda_t^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{16}$$

The matrix inversion lemma can now be applied to (15) leading to following recursive Levenberg Marquardt (RLM) formulation.

$$S(w_t) = \alpha_t \Lambda_t + \Omega^T(w_t)P_{t-1}\Omega(w_t) \tag{17}$$

$$P_t = \frac{1}{\alpha_t}[P_{t-1} - P_{t-1}\Omega(w_t)S^{-1}(w_t)\Omega^T(w_t)P_{t-1}] \tag{18}$$

$$w_{t+1} = w_t + P_t\nabla y(w_t)\varepsilon(w_t) \tag{19}$$

This weight update requires the inversion of the 2x2 matrix $S(w_t)$. This is clearly much more cost effective than having to invert the $N_w x N_w$ matrix which arises if the matrix inversion lemma is applied to (11).

3. PARALLEL RECURSIVE LEVENBERG MARQUARDT

By discarding the inter-neuron weight correlations the covariance matrix, P, in (18) *can be decomposed on layer by layer and neuron by neuron basis, that is:*

$$P_{N_w \times N_w} \cong \begin{bmatrix} P^{NL}_{N_{NL}\times N_{NL}} & 0 \\ 0 & P^{L}_{N_L\times N_L} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} P_1^{NL} & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & P_{N_h}^{NL} \end{bmatrix}_{N_{NL}\times N_{NL}} & 0 \\ 0 & P^{L}_{N_L\times N_L} \end{bmatrix} \tag{20}$$

$$\begin{matrix} N_{NL} = (\bar{N}_i)N_h \\ N_L = \bar{N}_h \end{matrix}, \quad \begin{matrix} \bar{N}_i = N_i + 1 \\ \bar{N}_h = N_h + 1 \end{matrix} \tag{21}$$

where P_i^{NL} is the $\bar{N}_i \times \bar{N}_i$ covariance matrix for ith neuron in the hidden layer and P^L is the $\bar{N}_h \times \bar{N}_h$ covariance matrix for the output layer neuron. The gradient vector can be equivalently decomposed as

$$\nabla y(w) = \begin{bmatrix} [\nabla y(w^{NL})]_{N_{NL}x1} \\ [\nabla y(w^{L})]_{N_L x1} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \nabla y(w_1^{NL}) \\ \dots \\ \nabla y(w_{N_h}^{NL}) \end{bmatrix}_{(\bar{N}_i N_h)x1} \\ [\nabla y(w^{L})]_{\bar{N}_h x1} \end{bmatrix} \tag{22}$$

The recursive algorithm (19) can therefore be decomposed into $\bar{N}_h$ sub-algorithms, that is

$$w^{NL}_{t+1(i)} = w^{NL}_{t(i)} + P^{NL}_{t(i)}\nabla y_{(i)}(w_t^{NL})\varepsilon(w_t) \text{ and } i = 1, \dots, N_h \tag{23}$$

$$w^{L}_{t+1} = w^{L}_t + P^{L}_t\nabla y(w^{L}_t)\varepsilon(w_t) \tag{24}$$

The routine for updating each $P_{t(i)}^{NL}$ is similar in form to (17) - (18), that is

$$S_{(i)}(w_t^{NL}) = \alpha_t \Lambda_t^{NL} + \Omega_{(i)}^T(w_t^{NL}) P_{t-1(i)}^{NL} \Omega_{(i)}(w_t^{NL}) \qquad (25)$$

$$P_{t(i)}^{NL} = \frac{1}{\alpha_t}(P_{t-1(i)}^{NL} - P_{t-1(i)}^{NL}\Omega_{(i)}(w_t^{NL})S_{(i)}^{-1}(w_t^{NL})\Omega_{(i)}^T(w_t^{NL})P_{t-1(i)}^{NL}) \qquad (26)$$

$$\Omega_{(i)}^T(w_t^{NL}) = \begin{bmatrix} & \nabla y_{(i)}^T(w_t^{NL}) & \\ 0 \ldots & 1 & \ldots 0 \end{bmatrix} \quad (\Lambda_t^{-1})^{NL} = \begin{bmatrix} 1 & 0 \\ 0 & \rho^{NL} \end{bmatrix} \quad i = 1, \ldots, N_h \qquad (27)$$

where $\Omega_{(i)}(w_t^{NL})$ is a $\overline{N}_i$x2 matrix whose second column has only one non zero element located at t mod $\overline{N}_i + 1$. Similarly the P_t^L matrix is updated as follows.

$$S(w_t^L) = \alpha_t \Lambda_t^L + \Omega^T(w_t^L) P_{t-1}^L \Omega(w_t^L) \qquad (28)$$

$$P_t^L = \frac{1}{\alpha_t}(P_{t-1}^L - P_{t-1}^L \Omega(w_t^L) S^{-1}(w_t^L) \Omega^T(w_t^L) P_{t-1}^L) \qquad (29)$$

$$\Omega^T(w_t^L) = \begin{bmatrix} & \nabla y^T(w_t^L) & \\ 0 \ldots & 1 & \ldots 0 \end{bmatrix} \qquad (\Lambda_t^{-1})^L = \begin{bmatrix} 1 & 0 \\ 0 & \rho^L \end{bmatrix} \qquad (30)$$

where $\Omega(w_t^L)$ is a $\overline{N}_h$x2 matrix whose second column has only one non zero element located at t mod $\overline{N}_h + 1$.

The advantage of this decomposition is that weight updates can be computed on a neuron by neuron basis facilitating parallel implementation [12]. In addition it allows the inclusion of $\overline{N}_h$ regularization terms at each sample (training) instant compared to just one when using the conventional RLM approach. The resulting algorithm, which will be referred to as parallel recursive Levenberg-Marquardt (PRLM), is also computationally more efficient than RLM.

4. SEPARABLE RECURSIVE LEVENBERG MARQUARDT

A separable parallel hybrid RLM algorithm (PHRLM) is obtained by optimizing the nonlinear network parameters using PRLM, as outlined in the previous section, while employing regularized RLS ([5],[11]) to optimise the linear weights. Generally for feedforward neural networks the nonlinear weights will correspond to hidden layer parameters and the linear weights to output layer parameters. Thus this approach can be thought of as a layer-level partitioning of the weights space.

Since the output layer neuron is linear-in-parameter the gradient in (22) can be re-written as

$$\nabla y(\boldsymbol{w}) = \begin{bmatrix} \nabla y(\boldsymbol{w}^{NL}) \\ \nabla y(\boldsymbol{w}^{L}) \end{bmatrix} = \begin{bmatrix} \nabla y(\boldsymbol{w}^{NL}) \\ \boldsymbol{r} \end{bmatrix} \text{ and } \boldsymbol{r} = \begin{bmatrix} f(\gamma_1) \ \dots \ f(\gamma_{N_h}) \ 1 \end{bmatrix}^T \tag{31}$$

where $\boldsymbol{r}$ is a vector consisting of the hidden layer neuron outputs and a dummy output of unity to accommodate the output layer bias. Thus optimization of the linear weights $\boldsymbol{w}^L$ is achieved using the RLM algorithm with $\nabla y(\boldsymbol{w}_t^L)$ replaced by $\boldsymbol{r}_t$, that is:

$$P_t^L = \frac{1}{\alpha_t}\left(P_{t-1}^L - \frac{P_{t-1}^L \Psi_t \Psi_t^T P_{t-1}^L}{\alpha_t \Lambda_t^L + \Psi_t^T P_{t-1}^L \Psi_t}\right) \tag{32}$$

$$\Psi^T = \begin{bmatrix} & \boldsymbol{r}^T & \\ 0 & \dots \ 1 \ \dots & 0 \end{bmatrix} \qquad (\Lambda_t^{-1})^L = \begin{bmatrix} 1 & 0 \\ 0 & \rho^L \end{bmatrix} \tag{33}$$

$$\boldsymbol{w}_{t+1}^L = \boldsymbol{w}_t^L + P_t^L \boldsymbol{r}_t \varepsilon(\boldsymbol{w}_t) \tag{34}$$

Again Ψ is a $\bar{N}_h$x2 matrix whose second column has only one non zero element located at t mod $\bar{N}_h$+1. Hybrid training then proceeds at each iteration by computing the nonlinear weights update, $\boldsymbol{w}_{t+1}^{NL}$, with the linear weights held constant, followed by a linear weights update, $\boldsymbol{w}_{t+1}^L$ with the nonlinear weights held constant.

5. SIMULATION RESULTS

Two benchmark neural network modelling problems are used to compare the performance of the proposed training algorithms with RLM and RPE.

Problem A: Identification of the chaotic Mackey-Glass time series defined by the differential delay equation

$$\frac{dy_t}{dt} = -0.1y_t + 0.2\frac{y_{t-\tau}}{1 + y_{t-17}^{10}} \tag{35}$$

and sampled at a rate of one second. A (4,10,1) MLP network with input vector

$$\boldsymbol{x}_t = [y_{t-1}, y_{t-6}, y_{t-12}, y_{t-18}] \tag{36}$$

was used to form a NAR model of the system.

Problem B: A non-linear dynamical system governed by the equation

$$y_k = 0.3y_{k-1} + 0.6y_{k-2} + 0.6\sin(\pi u_k) + 0.3\sin(3\pi u_k) + 0.1\sin(5\pi u_k) \tag{37}$$

and driven by the input

$$u_k = \sin\left(2\pi\frac{k}{250}\right) + \sin\left(2\pi\frac{k}{200}\right) \tag{38}$$

Here a (3,10,1) network was used and the input vector x_k chosen as

$$x_k = [y_{k-1}, y_{k-2}, u_k] \tag{39}$$

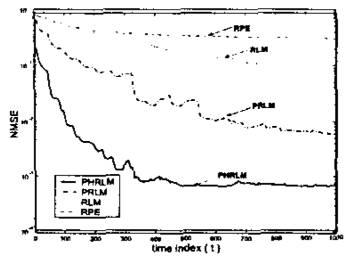
Evolution of the normalized mean squared error (NMSE) over a representative data set was used as a performance measure.

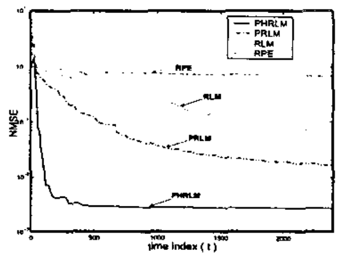
$$J(w_t) = \sum_{k=1}^{N_v} (y_k(w_t) - d_k)^2 / \sum_{k=1}^{N_v} d_k^2 \tag{40}$$

Network parameters were randomly set and the covariance matrices in each algorithm were initialized as identity matrices. The regularization parameters ρ in RLM, (ρ^{NL}, ρ^{L}) in PRLM and (ρ^{NL}, ρ^{L}) in PHRLM were set to 10, (1,0.01) and (0.01,0.001) respectively, while the forgetting factor, α_t, in each case was set to 0.999. Training was performed over the first 1000 iterations and the NMSE performance measure computed over the next 1000 iterations. The average performance obtained for each algorithm (computed on the basis of 10 simulation runs) is recorded in table 1. Figure 1 shows the evolution of the performance measure for a typical training run for both problems. For completeness the quality of the models obtained with PHRLM is illustrated in figure 2.

TABLE I:
NMSE RESULTS FOR PROBLEMS A AND B

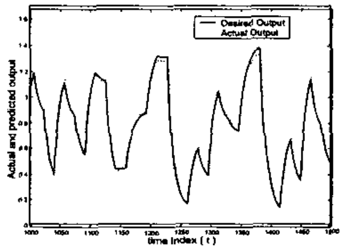
Method	Problem	Mean (10 runs)	Standard Deviation
RPE	A	0.43952	0.01831
	B	0.77432	0.00301
RLM	A	0.27332	0.00709
	B	0.32100	0.00647
PRLM	A	0.06928	0.00461
	B	0.12436	0.12771
PHRLM	A	0.00826	0.00018
	B	0.02280	0.00002

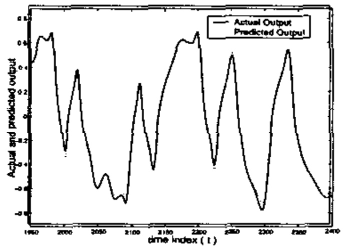




A (Mackey-Glass) B (dynamical system)

Figure 1: Evolution of the NMSE training error for Problem A and B





A (Mackey-Glass) B (dynamical system)

Figure 2: Performance of models obtained using PHRLM

From these results it can be seen that PRLM and PHRLM perform significantly better than the conventional methods (RPE and RLM). Of the two algorithms proposed the PHRLM showed superior results.

6. CONCLUSION AND DISCUSSION

Parallel recursive and parallel hybrid recursive Levenberg-Marquardt training algorithms has been developed on the basis of a neuron-level and layer-level decomposition of RLM. Preliminary results for the proposed methods indicate superior performance compared to existing RLM and RPE approaches. In addition the algorithms are computationally less complex than RLM and facilitate parallel implementation.

ACKNOWLEDGEMENTS

The authors wishes to acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) under Grant GR/N00333 and the Queen's University of Belfast.

REFERENCE

[1] P. Van Der Smagt, "Minimisation methods for training feedforward neural networks", **Neural Networks**, Vol. 7, No.1, pp. 1-11, 1994.

[2] S.A. Billings, H.B. Jamaluddin and S.Chen, "A Comparison of the Backpropagation and Recursive Prediction Error Algorithms for Training Neural Networks", **Mechanical Systems and Signal Processing**, vol. 5, no. 3, pp. 233-255, 1991.

[3] S. Chen and S.A. Billings, "Recursive prediction error parameter estimator for non-linear models", **Int. Journal of Control**, vol. 49 no. 2, pp. 569-594, 1989.

[4] S.H. Ngia, J. Sjöberg and M. Viberg, "Adaptive neural nets filter using a recursive Levenberg-Marquardt search direction", 3rd Asilomar Conf. on Signals, System and Computer Proceeding, pp. 697-701,1998.

[5] S.Gunnarsson, "Combining Tracking and Regularization in Recursive Least Square Identification", **Proc. of the 35th IEEE Conference on Decision and Control,** pp. 2551-2552, 1996.

[6] S. Chen, C.F.N Cowan S.A. Billings and P.M Grant, "Parallel Recursive Prediction Error Algorithm for Training Layered Neural Networks.", **Int J. Control**, vol. 51, no. 6, pp. 1215-1228, 1990.

[7] S.H. Ngia and J. Sjöberg, "Efficient training of neural nets for nonlinear adaptive filtering using a recursive Levenberg-Marquardt algorithm", **IEEE Transactions on Signal Processing**, vol. 48, no. 7, pp. 1915-1927, July 2000.

[8] V.S. Asirvadam, S.F. McLoone and G.W.Irwin, "Separable Recursive Training Algorithms for Feedforward Neural Networks", IEEE World Congress on Computational Intelligence, Honolulu, Hawaii, pp. 1212-17,2002

[9] J. Sjöberg and M. Viberg, "Separable non-linear least-squares minimization possible improvements for neural net fitting", **Proc. IEEE Workshop in Neural Networks for Signal Processing**, Amelia Island Plantation, Florida, Sep. 24-26, 1997.

[10] S. McLoone, M. Brown, G. Irwin and G. Lightbody, "A Hybrid linear/ nonlinear training algorithm for feedforward neural network", **IEEE Transaction on Neural Networks,** vol. 9 no. 4, pp. 669-684, July 1998.

[11] L. Ljung and T. Söderström, "**Theory and Practice of Recursive Identification**", MIT, Cambridge, MA, 1983.

[12] S.F. McLoone and G.W. Irwin, "Fast parallel off-line training of multilayer perceptrons", **IEEE Trans. Neural Networks**, vol. 8 no. 3,pp. 646-653, July 1997.