DCDIS A Supplement, Advances in Neural Networks, Vol. 14(S1) 242--247

A Least-Squares Learning for Block-based Neural Networks

Wei Jiang and Seong G. Kong

Department of Electrical and Computer Engineering
The University of Tennessee
Knoxville, TN 37996-2100, U.S.A.
E-mail: {wjiang, skong}@utk.edu

***Abstract* – This paper presents a block-wise least squares (BLS) learning algorithm for finding the optimal internal weights of block-based neural networks (BbNNs). A BbNN consists of a 2-D array of modular basic blocks that can be easily implemented using reconfigurable digital hardware. In the proposed algorithm, a set of linear equations is constructed for each block that is solved using a linear least-squares method to optimize the weights. The least squares-based optimization is performed in a block-wise fashion starting from blocks in higher stages to those in lower stages. The effectiveness of BLS algorithm is validated by a time series prediction and a realistic nonlinear heat exchanger system identification problem. The BLS algorithm demonstrates faster convergence speed with orders of magnitude compared with a gradient descent search.**

I. INTRODUCTION

Block-based neural network (BbNN) model [1] incorporates modular structures and evolutionary algorithms. BbNNs can be implemented by use of reconfigurable digital electronic hardware such as FPGAs that allow on-line partial reorganization of internal structures [2]. The modular structure of BbNNs enables easy expansion in size by adding more blocks. The structure and internal weights of BbNNs are optimized with evolutionary algorithms. BbNNs have been applied to the problems such as mobile robot navigation [1], pattern recognition [3], time series prediction [4] and ECG signal analysis [5][6].

A block-based neural network can be used only after it is properly trained for the target problem. The evolutionary algorithms provide an effective optimization technique for general problems including neural network training, but it suffers from the uncertainty in finding an exact solution due to its stochastic nature. The slow convergence speed is another major drawback of evolutionary algorithm-based learning for neural networks. Gradient descent algorithm [7] and its variations [8] have been widely used in training feedforward neural networks. However, gradient-based learning algorithms suffer from several major drawbacks that include the slow rate at which the algorithm converges to a satisfactory solution. Moreover, a set of parameters like the learning rate need to be tuned for optimized performance. As an alternative to gradient-based approaches, least squares-based learning algorithms for multilayer perceptrons have demonstrated superior learning speed [9][10][11].

This paper explores the use of deterministic algorithms for the optimization of internal weights of BbNN in an effort to circumvent the difficulties faced with stochastic evolutionary algorithms. Observing the slow learning speed associated with the gradient descent search (GDS), we propose a block-wise least squares-based (BLS) learning algorithm to increase the convergence speed. The proposed method takes a block-wise optimization procedure based on least squares method. Each block in the network corresponds to a simple feedforward neural networks and its optimization is treated separately. For blocks with known desired outputs, the internal weights are optimized by minimizing the least squares criterion. For other blocks with unknown target outputs, weight optimization is completed using estimated desired outputs. Optimization of the weights of blocks in higher stages is performed earlier than the blocks in lower stages. A training epoch consists of weight learning for all blocks from last stage to first stage. The training process is finished after a stop criterion is satisfied. The proposed learning algorithm is validated using function approximation and system identification problems. We describe the BbNN model in section II. Section III briefs the gradient descent search for BbNN. The least squares-based learning algorithm is presented in section IV. Experimental results for a times series prediction and nonlinear system identification are given in section V. Section VI concludes this paper.

II. BLOCK-BASED NEURAL NETWORKS

A. Network Structure

A BbNN can be represented by a two-dimensional (2-D) array of blocks. Each block is a basic processing element that corresponds to a feedforward neural network with four variable input/output nodes. A block is

connected to its four neighboring blocks with signal flow represented by an arrow between the blocks. A feedforward implementation of block-based neural network is considered in which all horizontal connections are fixed leftward. Signal flow uniquely specifies the internal configurations of a block as well as the overall network structure. Figure 1 illustrates the network structure of an $m \times n$ BbNN with m rows and n columns of blocks labeled as B_{ij}. The first column of blocks B_{11}, B_{21}, ..., B_{m1} is an input layer and the blocks B_{1n}, B_{2n}, ..., B_{mn} form an output layer. BbNNs with m rows can have up to m inputs and m outputs.

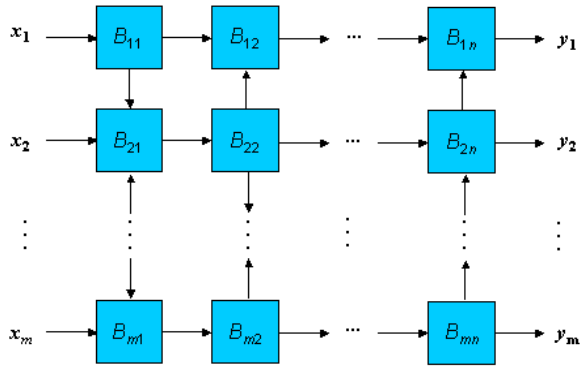

Figure 1: Structure of block-based neural networks.

Internal configuration is characterized by the input-output connections of the nodes. A node can be either an input or an output according to the internal configuration determined by the signal flow. An incoming arrow to a block specifies the node as an input, and output nodes are associated with outgoing arrows. Generalization capability emerges through various internal configurations of a block. A block can be represented by one of the three different types of internal configurations. Figure 2 shows a block with three inputs and one output (3/1).

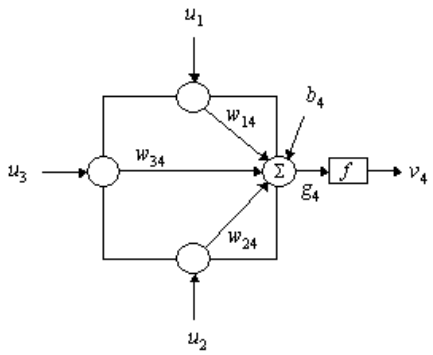

Figure 2: Example of an internal configuration of a block with three inputs and one output (3/1).

The four nodes inside a block are connected with each other with the weights. The signal u_i denotes the input and v_j indicates the output of the block, in which the subscripts indicate the node positions. The top, bottom, left and right node have indices 1, 2, 3, and 4, respectively. A weight w_{ij} thus denotes a connection from node i to node j. A block can have up to six connection weights including the bias. For the case of two inputs and two outputs (2/2), there are four weights and two biases. The 1/3 case has three weights and three biases, and the 3/1 configuration has three weights and one bias.

B. Signal Propagation

The overall signal flows determine the input-output computation path, along which an input signal $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ propagates through the blocks from left to right and generates a network output $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$. Outputs from the blocks in stage s become the inputs of the blocks in stage s+1. A block B_{ij} has four horizontal and vertical neighbors. Let us denote the indices of this set of neighbors of B_{ij} by $N(B_{ij})$ given by:

$$N(B_{ij}) = \{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\} \quad (1)$$

The block B_{ij} is connected with its four neighbors by either incoming or outgoing arrow depending on the signal flow. We further use $I(B_{ij})$ to denote the subset of $N(B_{ij})$ that are connected to block B_{ij} with outgoing arrows. The computation stage of block B_{ij} is computed according to the following equation:

$$s = \max_k \left(s^k\right) + 1, \quad k \in I\left(B_{ij}\right) \quad (2)$$

where $I(B_{ij})$ may include 0, 1, 2, or 3 neighbors depending on block configuration. When a block B_{ij} is in the input layer and $I(B_{ij})$ is a null set, its computation stage equals 1.

For a block B_{ij} in the network, its output node produces an output v_q with an activation function $f(\cdot)$ according to:

$$v_q = f\left(g_q\right), q \in J \quad (3)$$

where the net activation is computed as:

$$g_q = \sum_{p \in I} w_{pq} u_p + b_q, q \in J \quad (4)$$

where I and J are the respective index sets of input nodes and output nodes in the block. For the type 3/1 basic block shown in Figure 2, $I = \{1, 2, 3\}$ and $J = \{4\}$. For blocks of the type 1/3, $I = \{3\}$ and $J = \{1, 2, 4\}$. The type 2/2 blocks have $I = \{1, 3\}$ and $J = \{2, 4\}$. The term b_q is the bias term to the qth node. A standard sigmoid activation function of the form:

$$f\left(g_q\right) = \frac{1}{1+e^{-g_q}}, q \in J \tag{5}$$

is used.

III. GRADIENT DESCENT SEARCH (GDS)

The gradient descent search updates the block weights in a BbNN along gradient descent direction. There are two steps within an epoch: forward pass of the inputs to compute network outputs and backward pass of error signals to update internal weights. In a backward pass, the error signals propagate from the blocks of higher stages to lower stages. The error criterion function is defined as:

$$\varepsilon = \frac{1}{2}\sum_{l=1}^{M}\left\|\boldsymbol{d}^l - \boldsymbol{y}^l\right\|^2 \tag{6}$$

where M is the total number of training patterns. The two vectors $\boldsymbol{d}^l$ and $\boldsymbol{y}^l$ are the target and actual outputs for the l-th training pattern respectively. Gradient steepest descent rule updates the weights according to the following equation:

$$\Delta w_{ij} = -\eta\frac{\partial\varepsilon}{\partial w_{ij}}, \quad i \in I, j \in J \tag{7}$$

where η is the learning rate. I and J are the index sets of input and output nodes. The increment Δw_{ij} of the internal weight of a block can be deduced according to the generalized delta rule [8].

$$\frac{\partial\varepsilon}{\partial w_{ij}} = -\rho_j u_i \tag{8}$$

where ρ_j is the sensitivity of the output node j of the block to the error, and u_i is the input to the input node i of the block. For a network output node, the sensitivity becomes:

$$\rho_j = f'\left(g_j\right)\left(d - y\right) \tag{9}$$

where d and y are the desired and actual outputs at the node. f' is the first derivative of the tangent activation function and g_j is a net input to the node. The sensitivity of an output node of a block in stage s is computed as:

$$\rho_j = f'\left(g_j\right)\sum_{j\in J}\rho_j^{'}w_{ij} \tag{10}$$

where $\rho_j^{'}$ is the sensitivity of the output node j of the block in stage s+1 that is connected to the current output node through weight w_{ij}. Thus the weight update can be summarized in the following equation using above Eqs. (8)-(10):

$$\Delta w_{ij} = \eta\rho_j u_i, \quad i \in I, j \in J \tag{11}$$

IV. BLOCK-WISE LEAST SQUARES LEARNING (BLS)

BLS algorithm takes a block-wise optimization procedure in that the internal weights of each block are optimized by solving a set of linear equations and the blocks in the network are optimized from higher stages to lower stages.

A. Training a Single Block

Each block makes a simple feedforward neural network. For a given input to the block, the output is computed according to Eq. (3). For a set of inputs, the output equation can be written in matrix format as:

$$\mathbf{G} = \mathbf{UW} \tag{12}$$

where U is input to the block with the first column being the output (a constant of 0.5) from a bias node and each row representing a data sample vector. W is the internal weights of the block.

Determining the optimal weights W can be formulated as a linear least squares problem:

$$\text{minimize} \quad \left\|\mathbf{UW} - \mathbf{D}\right\|_2 \tag{13}$$

where D is the desired output. A linear least squares problem can be solved using QR decomposition together with Householder transformation or through singular value decomposition (SVD). This paper utilizes QR decomposition with Householder transformation due to its lower computation complexity compared to SVD.

After the optimal weights are determined, a set of desired input R are found to further reduce the least squares error according to:

$$\text{minimize} \quad \left\|\mathbf{W}^T\mathbf{R}^T - \mathbf{D}^T\right\|_2 \tag{14}$$

where matrix W is the optimal weights determined from Eq. (13). Matrix R is solved by using QR factorization with Householder transformation and the thus acquired desired output becomes the target output for the blocks connected to current block. However, due to the use of nonlinear sigmoidal function, the output from output node is bounded. The acquired input R has to be transformed to bring its range into that of the activation function. To that purpose, a transformation matrix C is used [9] with the elements of C given in the following:

$$c_{0,0}=1,\quad c_{0,k}=2\left(\alpha_k-\max\left(0.5,\beta_k-\alpha_k,\alpha_k-\gamma_k\right)\right)$$
$$c_{k,k}=2\max\left(0.5,\beta_k-\alpha_k,\alpha_k-\gamma_k\right) \tag{15}$$

where α_k, β_k, and γ_k are the mean, maximum and minimum of the kth column of matrix R.

In order to keep the minimum achieved in Eq. (14) unchanged, the following transformation is made to R:

$$\mathbf{R}_t=\mathbf{R}\mathbf{C}^{-1} \tag{16}$$

and W

$$\mathbf{W}_t=\mathbf{C}\mathbf{W} \tag{17}$$

After the optimal weights W and desired input R are determined, the learning process for the block is completed.

B. Training a Block-based Neural Network

The computation stage s associated with a block denotes the priority according to which each block is trained. The blocks in higher stages are trained earlier than those in lower stages. The blocks within the same stages are trained with the same priority.

The BLS algorithm for BbNN can be summarized in the following:

1) Generate randomly initial internal weights for each block in the network.

2) Propagate all patterns through the network from blocks in lower computation stages to blocks in higher stages producing outputs according to Eq. (12).

3) Update the weights for the block in stage s using Eq. (13).

4) Update the input for the block in stage s (the desired output for the connected block in stage s-1) according to Eq. (14).

5) Transform the required input R and weight W using Eqs. (16) and (17).

6) Repeat steps 3) - 5) for each block in stage s-1.

7) If end condition is met, stop learning; otherwise, go to step 2).

C. Computation Complexity

The number of multiplications required to solve a linear least squares problem using QR decomposition and Householder transformation equals $M\times n\times(m+n)$, in which M and n are the dimensions of input matrix and n and m are dimensions of weight matrix. The computation complexity of optimizing the block shown in Figure 2 will be $O(20M)$. As a comparison, the GDS optimization of the same block type will have a complexity of $O(4M)$. Thus, the operation complexity of both algorithms is only linearly correlated to the number of examples. The BLS algorithm takes more operations than the GDS algorithm per epoch. However, the experiment results presented in the following show that BLS is much faster than GDS since BLS takes only a few epochs compared to hundreds of epochs of GDS.

V. EXPERIMENTAL RESULTS

This section presents experimental results of the GDS and BLS algorithms for two function approximation problems: one is the well-known Mackey-Glass time series prediction and the other is a realistic nonlinear heater exchanger system identification problem. Both GDS and BLS algorithms are implemented using Matlab 6.5 and run on a PC platform with Pentium 4 2.80 GHz CPU. The performance of the two algorithms is compared.

A. Time Series Prediction

The time series prediction is to find future values based on observations up to current time, i.e.:

$$\hat{x}\left(t+I\right)=f\left(x\left(t\right),x\left(t-1\right),...,x\left(t-D\right)\right) \tag{18}$$

where t denotes current time index.

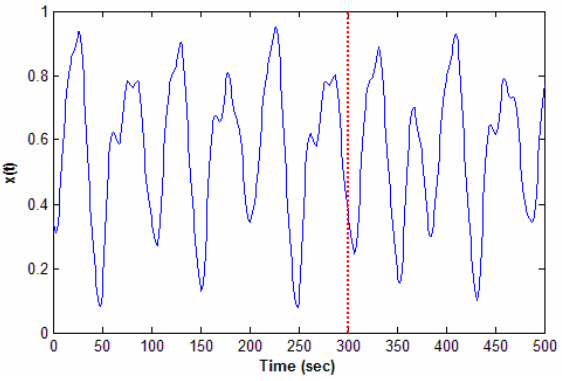

Figure 3: A Mackey-Glass time series.

The Mackey-Glass (MG) time series is a chaotic time series simulating blood flow [12] and it is one of the widely investigated benchmark examples in time series prediction. The MG time series can be represented using the following differential equation:

$$\dot{x}\left(t\right)=-ax\left(t\right)+\frac{bx\left(t-\tau\right)}{1+x^{10}\left(t-\tau\right)} \tag{19}$$

The MG time series value at integer points was obtained by applying the fourth-order Runge-Kutta method to find solution to Eq. (19). The parameters used are a = 0.1, b = 0.2 and τ = 17. For this particular value of τ, the system exhibits a chaotic behavior. Figure 3 shows a MG times series generated for experiments where the dotted vertical

line marks the beginning of test phase. Among the total 500 data points, 300 of them are used to train a selected 3×2 BbNN with fixed structure and the remaining 200 points serve as test data.

The time series data in lagged space $x(t)$, $x(t\text{-}1)$, and $x(t\text{-}2)$ are inputs to the BbNN and $x(t+1)$ is the output from the network, i.e., $I = 1$ and $D = 2$. Starting from an randomly generated initial set of weights, both GDS and BLS algorithms are applied to optimize the internal weights of selected BbNN. The learning rate selected for GDS is 0.05 that allows faster convergence based on some initial trials. There is no parameter that needs to set for the BLS algorithm to work. Figure 4 show the typical 1-step prediction results from the trained BbNN for the test data.

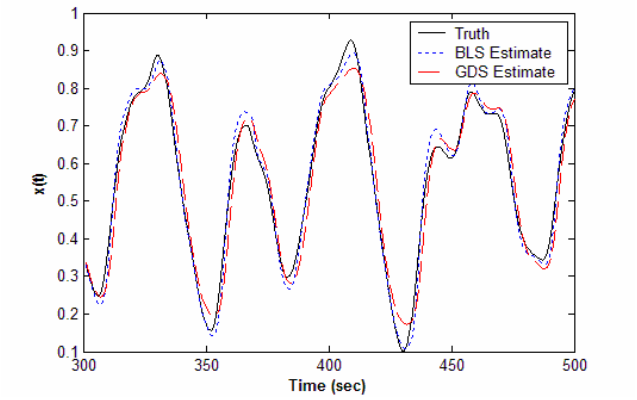

Figure 4: Test data of MG time series BbNN prediction.

The performance of the BLS algorithm regarding convergence speed is compared to that of GDS. Table 1 lists the numerical comparison between the two methods. The BLS takes only a few epochs to reach the level of error that the GDS algorithm does not achieve after 2,000 epochs. The actual CPU running time of BLS is also significantly less than that of GDS algorithm.

Table 1: Performance comparison of GDS and BLS for MG time series prediction

Method	Epoch	Time (s)	MSE(Train/Test)
GDS	2,000	2.6	$4.56/4.40\times10^{-3}$
BLS	4	0.031	$6.50/6.37\times10^{-4}$

The Mean Squared Error (MSE) as the error criterion is also compared between the two methods for both training and test data. The BbNN trained with BLS algorithm achieves the error level that is nearly 10 times less than that achieved with GDS after 2000 epochs. The BbNN trained with BLS is found to generalize well to the test data that is not seen before. The errors for training and test data are comparable.

B. Nonlinear System Identification

Conventional techniques for nonlinear system identification utilizing mathematical models require the structure of the model must be known in advance. Block-based neural networks provide a general model-free approach for identifying nonlinear systems. The system in interest is a practical liquid-saturated steam heat exchanger [13], where water is heated by pressurized saturated steam through a copper tube. The input variables are the liquid flow rate, the steam temperature, and the inlet liquid temperature. The system output is the outlet liquid temperature. In this experiment the steam temperature and the inlet liquid temperature are kept constant to their nominal values. The system model can be described as in Eq. (20), in which u and y denote the system input and output, respectively.

The set of data employed in training the neural network has a large impact on the quality of the identified system model, which means the set of training data needs to include as much information as possible about the dynamics of the system. It is therefore important to construct a balanced set of training data that covers the whole system operation range. To this end uniformly distributed input over the process range are generated and serve as system input. Figure 6 shows the corresponding fluid outlet temperature in which dotted vertical line separates the training and test data.

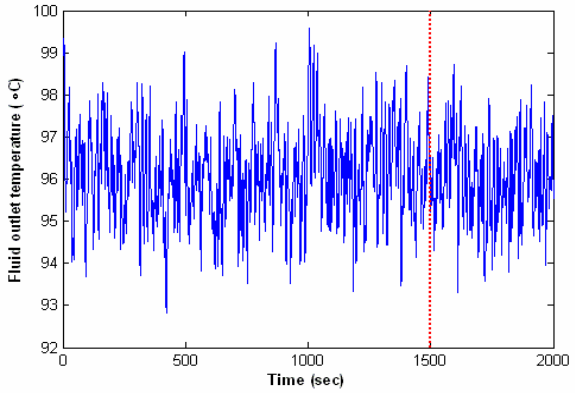

Figure 5: Fluid outlet temperature of a practical heat exchanger.

The system outputs data $y(t\text{-}1)$, and lagged input $x(t\text{-}1)$, and $x(t)$ are inputs to the BbNN and $y(t)$ is the output from the network, i.e., $D_1 = D_2 = 1$. Starting from an randomly generated initial set of weights, both GDS and BLS algorithms are applied to optimize the weights of a selected 3×2 BbNN with fixed structure. The first 1500 input-output pairs in the data set are training data and the rest serve as independent test data. Figure 6 show the typical predicted system output from the trained BbNN for the first 300 test samples.

$$\hat{y}(t) = f\left(y(t-1), y(t-2), ..., y(t-D_1); u(t), u(t-1), ..., u(t-D_2)\right) \quad (20)$$

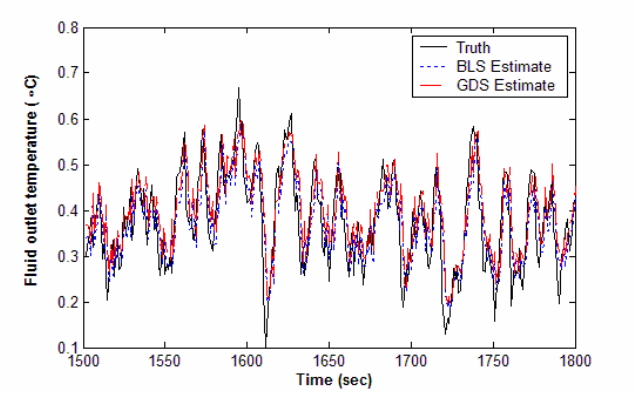

Figure 6: Outlet temperature of the simulated process and BbNN prediction.

The performance of the BLS algorithm regarding convergence speed is compared to that of GDS. The learning rate selected for the GDS is 0.02. Table 2 lists the comparison of numerical results between the two methods. The BLS algorithm takes only a few epochs to reach the level of error that the GDS algorithm takes 2,000 epochs to achieve. The actual CPU running time of BLS is also significantly less than that of GDS algorithm.

Table 2: Performance comparison of GDS and BLS for nonlinear heat exchanger identification

Method	Epoch	Time (s)	MSE(Train/Test)
GDS	2,000	6.86	$4.28/4.50\times10^{-3}$
BLS	4	0.20	$3.96/4.10\times10^{-3}$

Similar to the case of MG time series prediction, the BbNN trained with BLS is found to generalize well to the test data that is not seen before. The errors for training and test data are comparable.

VI. Conclusion

This paper presented a block-wise least squares learning algorithm for adjusting the weights of block-based neural networks. The proposed algorithm takes a learning procedure based on least squares method to optimize the internal weights of BbNN in a block-wise fashion. The effectiveness of the proposed learning method was validated by time series prediction and realistic nonlinear system identification problems. The simulation results demonstrate that the proposed learning algorithm successfully optimizes the weights of block-based neural networks for both problems in a speed orders of magnitude faster than a gradient descent approach.

REFERENCES

[1] S. W. Moon and S. G. Kong, "Block-based Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 12, No. 2, pp. 307-317, Mar. 2001.

[2] S. Merchant, G. D. Peterson, S. K. Park, and S. G. Kong, "FPGA Implementation of Evolvable Block-based Neural Networks," *Proc. Congress on Evolutionary Computation (CEC-2006)*, Vancouver, July 2006.

[3] S. W. Moon and S. G. Kong, "Pattern Recognition with Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2002)*, Vol. 1, pp. 992-996, May 2002.

[4] S. G. Kong, "Time Series Prediction with Evolvable Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2004)*, Vol. 2, pp. 1579-1583, July 2004.

[5] W. Jiang, S. G. Kong, and G. D. Peterson "ECG Signal Classification with Evolvable Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2005)*, Vol. 1, pp. 326-331, July 2005.

[6] W. Jiang, S. G. Kong, and G. D. Peterson "Continuous Heartbeat Monitoring Using Evolvable Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2006)*, July 2006.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation", In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, Vol. 1, Ch. 8, pp. 318-362, MIT Press, Cambridge, MA, 1986.

[8] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Upper Saddle River, N.J.: Prentice Hall, 1999.

[9] F. B. König and F. Bärmann, "A Learning Algorithm for Multilayered Neural Networks Based on Linear Least Squares Problems," *Neural Networks*, 6:127-131, 1993.

[10] J. Y. F. Yam and T. W. S. Chow, "Accelerated Training Algorithm for Feedforward Neural Networks Based on Least Squares Method," *Neural Processing Letters*, 2(4): 20-25, 1995.

[11] J. Y. F. Yam and T. W. S. Chow, "Extended Least Squares Based Algorithm for Training Feedforward Networks," *IEEE Trans. on Neural Networks*, 8(3): 806-810, 1997.

[12] M. C. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control Systems," *Science*, Vol. 197, pp. 287-289, 1977.

[13] S. Bittanti and L. Piroddi, "Nonlinear Identification and Control of a Heat Exchanger: A Neural Network Approach," Journal of Franklin Institute, Vol. 334B, No. 1, pp.135-153, 1997.