Proof: See [14].

Proof of Proposition 3: First, from Lemma 2 we obtain by setting $u^* := Ax^*$, $u := Ax - y$, and $\Omega := \Omega_2$

$$[Ax - y - P_{\Omega 2}(Ax - y)]^T \{P_{\Omega 2}(Ax - y) - Ax^*\} \geq 0 \quad \forall x \in R^n \text{ and } y \in R^m. \quad (14)$$

Similarly, by setting $u^* := A^T y^*$, $u := A^T y + x$, and $\Omega := \Omega_1$ we get

$$[A^T y + x - P_{\Omega 1}(A^T y + x)]^T [P_{\Omega 1}(A^T y + x) - A^T y^*] \geq 0 \quad \forall x \in R^n \text{ and } y \in R^m. \quad (15)$$

On the other hand, from Proposition 1 we have

$$[P_{\Omega 2}(Ax - y) - Ax^*]^T y^* \geq 0, \qquad \forall x \in R^n \text{ and } y \in R^m \quad (16)$$

and

$$[P_{\Omega 1}(A^T y + x) - A^T y^*]^T (-x^*) \geq 0, \qquad \forall x \in R^n \text{ and } y \in R^m. \quad (17)$$

Adding (14) and (16) we obtain

$$[Ax - y - y^* - P_{\Omega 2}(Ax - y)]^T [P_{\Omega 2}(Ax - y) - Ax^*] \geq 0. \quad (18)$$

Adding (15) and (17) we get

$$[A^T y + x - x^* - P_{\Omega 1}(A^T y + x)]^T [P_{\Omega 1}(A^T y + x) - A^T y^*] \geq 0. \quad (19)$$

From (18) and (19) we have

$$\begin{cases} [E_1 - (y - y^*)]^T [A(x - x^*) - E_1] \geq 0 \\ [E_2 + (x - x^*)]^T [A^T (y - y^*) - E_2] \geq 0 \end{cases} \quad \forall x \in R^n \text{ and } y \in R^m$$

where E_1 and E_2 are defined in Theorem 1, respectively. It follows that for any $x \in R^n$ and $y \in R^m$:

$$\begin{cases} (x - x^*)^T A^T E_1 + (y - y^*)^T E_1 \geq (y - y^*)^T A(x - x^*) + E_1^2 \\ (y - y^*)^T A E_2 - (x - x^*)^T E_2 \geq -(y - y^*)^T A(x - x^*) + E_2^2 \end{cases}.$$

$$\begin{pmatrix} x - x^* \\ y - y^* \end{pmatrix}^T \begin{bmatrix} A^T E_1 - E_2 \\ A E_2 + E_2 \end{bmatrix} \geq E_1^2 + E_1^2.$$

This complete the proof of Proposition 3.

ACKNOWLEDGMENT

The author thanks the reviewers for valuable comments.

REFERENCES

[1] G. B. Dantzig, *Linear Programming and Extensions*. Princeton: Princeton Univ. Press, 1963.
[2] D. Kinderlehrer and G. Stampcchia, *An Introduction to Variational Inequalities and Their Applications.* New York: Academic, 1980.
[3] Y. Xia, "A new neural network for solving linear programming problems and its application," *IEEE Trans. Neural Networks*, vol. 7, pp. 525–529, Mar. 1996.
[4] J. J. Hopfield, and D. W. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, 1985.
[5] ——, "Simple 'neural' optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circuits Syst.*, vol. CAS-33, May 1986.
[6] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Trans. Circuits Syst.*, vol. CAS-35, May 1988.
[7] A. Rodríguez-Vázquez, R. Dominguer-Castro, A. Rueda, J. L. Huertas, and E. Sanchez-Sinencio, "Nonlinear switched-capacitor 'neural' networks for optimization problems," *IEEE Trans. Circuits Syst.*, vol. 37, Mar. 1990.
[8] X. Wu, Y. Xia, and W.-K. Chen, "A high-performance neural network for solving linear and quadratic programming problems," *IEEE Trans. Neural Networks*, vol. 7, pp. 643–651, May 1996.
[9] C. Y. Maa and M. Shanblatt, "Linear and quadratic programming neural network analysis," *IEEE Trans. Neural Networks*, vol. 3, July 1992.
[10] S. H. Zak, V. Upatising, and S. Hui, "Solving linear programming problems with neural network: A comparative study, *IEEE Trans. Neural Networks*, vol. 6, pp. 94–104, Jan. 1995.
[11] A. Bouzerdoum and T. R. Pattison, "Neural Network for quadratic optimization with bound constraints," *IEEE Trans. Neural Networks*, vol. 4, Mar. 1993.
[12] R. K. Miller and A. N. Michel, *Ordinary Differential Equations.* New York: Academic, 1982.
[13] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming.* New York: Addison-Wesley, 1973.
[14] Y. Xia and J. Wang, "Neural network for solving linear programming problems with bounded variables," *IEEE Trans. Neural Networks*, vol. 6, pp. 515–519, Mar. 1995.
[15] J. P. LaSalle, *The Stability of Dynamical Systems.* Philadelphia, PA: SIAM, 1976.

Extended Least Squares Based Algorithm for Training Feedforward Networks

Jim Y. F. Yam and Tommy W. S. Chow

***Abstract*—An extended least squares-based algorithm for feedforward networks is proposed. The weights connecting the last hidden and output layers are first evaluated by least squares algorithm. The weights between input and hidden layers are then evaluated using the modified gradient descent algorithms. This arrangement eliminates the stalling problem experienced by the pure least squares type algorithms; however, still maintains the characteristic of fast convergence. In the investigated problems, the total number of flops required for the networks to converge using the proposed training algorithm are only 0.221%–16.0% of that using the Levenberg–Marquardt algorithm. The number of floating point operations per iteration of the proposed algorithm are only 1.517–3.521 times of that of the standard backpropagation algorithm.**

I. INTRODUCTION

Conventional techniques used to train feedforward neural networks are mostly based on the method of gradient descent, i.e., the weights update is performed in the opposite direction to the gradient of an error cost function [1]. However, the convergence rate of the pure gradient-descent algorithm is often too slow for practical applications. A lot of methods were proposed including using different cost functions and applications of different optimization techniques [2]–[8]. Applications of different optimization techniques to train the feedforward networks are the most popular methods to increase the rate of convergence. The most well known algorithms of this type is the conjugate gradient training algorithm and the Levenberg–Marquardt (LM) training algorithm [6], [7]. These algorithms have the property of quadratical convergent. There are different

Manuscript received May 21, 1996; revised January 21, 1997.

The authors are with the City University of Hong Kong, Kowloon, Hong Kong.

Publisher Item Identifier S 1045-9227(97)03407-3.

1045–9227/97$10.00 © 1997 IEEE

versions of the conjugate gradient algorithm, each has a different search direction and line search method. The computational complexity of the conjugate gradient algorithm is heavily dependent on the search method. The LM algorithm is an extension of Gauss–Newton optimization technique [7]. It is more powerful than the gradient descent and seldom stuck in the local minima, but the memory required and computational complexity increases quadratically with the number of weights. This makes this method impossible to be used in large-scale problems. Recently, new training algorithms based on linear least squares have been proposed like optimization layer by layer (OLL) and least squares-based (LSB) learning algorithms [9]–[11]. In the pure linear LSB training algorithm, each layer of a neural network is decomposed into a linear part and a nonlinear part. The linear part is determined by least squares method. The remaining error is propagated back into the preceding layer of the neural network through the inverse of the activation function and a transformation matrix. These methods have an advantage that the network error can be minimized to a very small value after few iterations [10], [11]. In general, the error level of networks is small enough for practical applications. Moreover, the computational complexity of this kind of algorithm is only two to three times of that of the conventional training algorithm. However, the training process usually stalls after ten iterations; the network error cannot be further reduced with additional training. The problem arises from the use of a transformation matrix to transform the optimal output of the hidden layer into the range of the activation function. Apparently, there are numerous ways to construct a transformation matrix in accordance with different distribution of training data. The transformation matrix suggested in the paper may not be the best under all conditions [10]–[11]. For the problems that require the networks with very high accuracy, conventional training algorithms such as backpropagation and conjugate gradient algorithms are required to further train the networks. The OLL learning algorithm is also based on an optimization of the multilayer perceptron layer by layer using the least squares method and constrained optimization [9]. However, the activation function is approximated by a first-order Taylor series. In order to limit the error caused by linearization, a penalty term is introduced into the cost function. Also, the computational complexity and memory requirement of this algorithm increase quadratically with the number of weights. Also, Ergezinger reported that the flops per iteration of the OLL algorithm is about eight to nine times of that of the backpropagation algorithm. The OLL algorithm has an advantage over the LSB algorithm when a very small training error is required.

In this paper, an extended LSB training algorithm is developed to solve the stalling problem experienced by the pure linear LSB training algorithm. In the proposed algorithm, the weights between the last hidden and output layers are evaluated by linear least squares method. The weights between input and hidden layers are then evaluated by a modified gradient based training algorithm. This method eliminates the difficulty in determining the appropriate transformation matrices in pure linear LSB algorithm and thus the stalling problem is eliminated. The storage requirement and computational complexity of the proposed algorithms is slightly smaller than that of the pure linear LSB algorithms and is much smaller than those of the OLL and LM algorithms.

II. Training Algorithm

A multilayer neural network with L fully interconnected layers is considered. Layer l consists of $n_l + 1$ neurons ($l = 1, \cdots, L-1$), the last neuron being a bias node with a constant output of 1.0. If there are P patterns for network training, all given inputs can be represented by a matrix A^1 with P rows and $n_1 + 1$ columns. All elements of the last column of the matrix A^1 are constant 1.0. As the output layer does not have any bias node, the network output can be represented by a matrix A^L with P rows and n_L columns. Similarly, the target can be represented by a matrix T^L with P rows and n_L columns. The weights between neurons in the layers l and $l+1$ form a matrix W^l with entries $w^l_{i,j}$ ($i = 1, \cdots, n_l + 1, j = 1, \cdots, n_{l+1}$). Entry $w^l_{i,\,j}$ connects neuron i of layer l with neuron j of layer $l+1$.

The output of all hidden layers and the output layer are obtained by propagating the training patterns through the network. Let us define the matrix

$$O^l = A^l W^l. \tag{1}$$

The entries of A^{l+1} for all layers (i.e., $l = 1, \cdots, L-1$) are evaluated as follows:

$$a^{l+1}_{p,j} = f(o^l_{p,j}) \quad p = 1, \cdots, P \text{ and } j = 1, \cdots, n_{l+1} \tag{2}$$

where $f(x)$ is the activation function. The activation function for hidden neurons is the conventional sigmoidal function with the range between zero and one. The linear function is used for the output neurons.

Learning is achieved by adjusting the weights such that A^L is as close as possible or equal to T^L so that the mean squared error E is minimized, where E is defined as

$$E = \frac{1}{2P} \sum_{p=1,\cdots,P} \sum_{j=1,\cdots,n_L} \left(a^L_{p,j} - t^L_{p,j}\right)^2. \tag{3}$$

In this learning algorithm, the weights between the last hidden layer and the output layer are evaluated by a pure least squares algorithm; the weights between the input and the first hidden layer, and the weights between the hidden layers are evaluated by modified gradient descent algorithm. The problem of determining the W^{L-1} optimally can be formulated as follows:

$$\min \|A^{L-1}W^{L-1} - T^L\|_2 \text{ with respect to } W^{L-1}. \tag{4}$$

This linear least squares problem can be solved by using QR factorization together with Householder transforms or singular value decomposition (SVD) [11]. QR factorization using Householder transforms was implemented because it has less computational complexity than SVD [12]. After the optimal weights W^{L-1} are found, the new network output A^L are evaluated. To determine the appropriate weights change in the preceding layer, the remaining error is backpropagated to the preceding layer of the neural network. After the gradient information is obtained, the appropriate learning rate and momentum coefficient for each layer are determined in accordance with the correlation between the negative error gradient and the previous weight update of that layer [4]. The correlation coefficient between the negative gradient and last weight update for layer l is given by (5), shown at the bottom of the next page, where t indexes the presentation number, $-\nabla E^l_{i,j}(t)$ is the negative error gradient with respect to $w^l_{i,j}$ in the layer l and $\Delta w^l_{i,j}(t-1)$ is the previous weight change of weight $w^l_{i,j}$. $\overline{-\nabla E^l_{i,j}}(t)$ and $\overline{\Delta w^l_{i,j}}(t)$ are the mean values of the negative error gradients and the weight changes in layer l respectively. From this correlation coefficient, three different conditions can be identified.

1) When the correlation coefficient is near to one, there is almost no change in the direction of local error minimization and the change of weights is likely moving on the plateau. The learning rate can be increased to improve the convergence rate.
2) When the correlation coefficient is near to minus one, it implies an abrupt change in the direction of local error minimization which is likely moving along the wall of ravine. The learning rate should then be reduced to prevent oscillation across both sides of ravine.

3) When there is no correlation between the negative gradient and previous weight change, the learning rate should be kept constant.

According to these three conditions, the following heuristic algorithm is proposed to adjust the learning rate:

$$\eta^l(t) = \eta^l(t-1)\left(1 + \frac{1}{2}r^l(t)\right). \tag{6}$$

We notice that the learning rate can increase or decrease rapidly when the successive values of correlation coefficient remain the same sign. This feature enables the appropriate learning rate to be found in few iterations, and thus reduces the output error rapidly. Furthermore, the algorithm does not greatly increase the storage requirement as the second-order method. It does not need to calculate the second-order derivatives either.

The convergence rate is not optimized with a fixed momentum coefficient. The momentum term has an accelerating effect only when the $-\nabla E^l_{i,j}$ and $\Delta w^l_{i,j}$ have the same direction. For the fixed momentum coefficient, the momentum term may override the negative gradient term when the $-\nabla E^l_{i,j}$ and $\Delta w^l_{i,j}$ are in the opposite direction. The momentum coefficient $\alpha^l(t)$ at the tth iteration is determined as follows:

$$\alpha^l(t) = \lambda^l(t)\eta^l(t)\frac{\|-\nabla E^l(t)\|_2}{\|\Delta w^l(t-1)\|_2} \tag{7}$$

where

$$\|-\nabla E^l(t)\|_2 = \left(\sum_i \sum_j \left(-\nabla E^l_{i,j}(t)\right)^2\right)^{\frac{1}{2}}$$

and

$$\|\Delta w^l(t-1)\|_2 = \left(\sum_i \sum_j (\Delta w^l(t-1))^2\right)^{\frac{1}{2}} \tag{8}$$

$$\lambda^l(t) = \begin{cases} \eta^l(t) & \eta^l(t) < 1 \\ 1 & \eta^l(t) \geq 1 \end{cases} \tag{9}$$

As $\lambda^l(t)$ is always less than one, the momentum term will not override the negative gradient term.

After evaluating the $\eta^l(t)$ and $\alpha^l(t)$ for layers $l = L-2, \cdots, 1$, the new weights are determined. After that, the new network output and error are evaluated. One epoch is said to be completed. As the fast increase in learning rate may drive the neurons to their saturation region in some neural-network problems, the following strategy is used to improve the stability of the algorithm. If the present root mean squares error (RMSE) is greater than the previous one by 0.1, the last weight change is canceled and $\eta^l(t)$ is reduced by half. This strategy gives a small preference to learning rate reduction and enhances the robustness of the training process. The process is repeated until the network error reaches the specified error level.

The proposed algorithm can be summarized as follows.

1) Generate initial weights for each layer using pseudorandom generator.
2) Propagate all given patterns through the network so that the matrices $A^l, l = 2, \cdots, L$, can be successively evaluated. If the error norm between A^L and T^L is smaller than the specified value, the training is completed.
3) Evaluate a new set of weights W^{L-1} by solving the least squares problem in (4).
4) Compute a new A^L from the A^{L-1} and the new weights W^{L-1}. If the error norm between A^L and T^L is smaller than the specified value, the training completes.
5) Compute the gradient for layers $l = L-2, \cdots, 1$ using the new A^L and the new weights W^{L-1}
6) For $l = L-2, \cdots, 1$:
   A) Evaluate the learning rate η^l and momentum coefficient by applying (5)–(9).
   B) Evaluate the weight change $\Delta w^l_{i,j}$.
   C) Evaluate the new weights W^l.
7) Compute a new A^L from $A^1, W^1, \cdots, W^l, W^{l+1} \cdots$, and W^{L-1}. If error norm between A^L and T^L is smaller than the specified value, the training completes.
8) If the RMS error is larger than the previous RMS error by 0.1, the weight changes for all layers are canceled, the learning rate is reduced by half, and go to 6B) again.
9) Continue with 2).

III. Results and Discussions

In this section, we compare the convergence performance of this algorithm with other fast training algorithms; i.e., LSB algorithm and LM algorithm. We applied these algorithms to the following problems:

1) nonlinear function approximation;
2) learning the MG time series;
3) sunspot series prediction.

In order to demonstrate the advantages of using the least squares method to evaluate the output weights in the proposed algorithm, our results will also be compared to an algorithm that uses adaptive learning rate and momentum [(5)–(9)] to evaluate the weights of all layers, which we call the adaptive backpropagation (ABP) algorithm.

All the networks are started with random weights between -1 and one. In each problem, one hundred and eighty simulations were performed using different initial weights, learning rate, and momentum. All algorithms were written in Matlab scripts and function files. The BP and LM algorithms are provided in Matlab Neural Network Toolbox.

A nonlinear function approximation of eight input quantities x_i into three output quantities y_i, defined by the following equations, is used [10]:

$$\begin{aligned} y_1 &= (x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8)/4 \\ y_2 &= (x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8)/8 \\ y_3 &= (1 - y_1)^{0.5}. \end{aligned} \tag{10}$$

Fifty sets of input signals $x_i \in (0, 1)$ were generated by a random number generator, the corresponding y_i were computed using (10). The network architecture used to learn this function is a 8-12-3 one, i.e., the network has eight input neurons, one hidden layer with 12 neurons, and three output neurons. The termination criterion is chosen to be RMSE $= 0.01$.

The results for the function approximation problem using different algorithms are shown in Table I. The results show that the proposed

$$r^l(t) = \frac{\sum_i \sum_j \left(-\nabla E^l_{i,j}(t) - \overline{-\nabla E^l_{i,j}}(t)\right)\left(\Delta w^l_{i,j}(t-1) - \overline{\Delta w^l_{i,j}}(t-1)\right)}{\left[\sum_i \sum_j \left(-\nabla E^l_{i,j}(t) - \overline{-\nabla E^l_{i,j}}(t)\right)^2\right]^{\frac{1}{2}}\left[\sum_i \sum_j \left(\Delta w^l_{i,j}(t-1) - \overline{\Delta w^l_{i,j}}(t-1)\right)^2\right]^{\frac{1}{2}}} \tag{5}$$

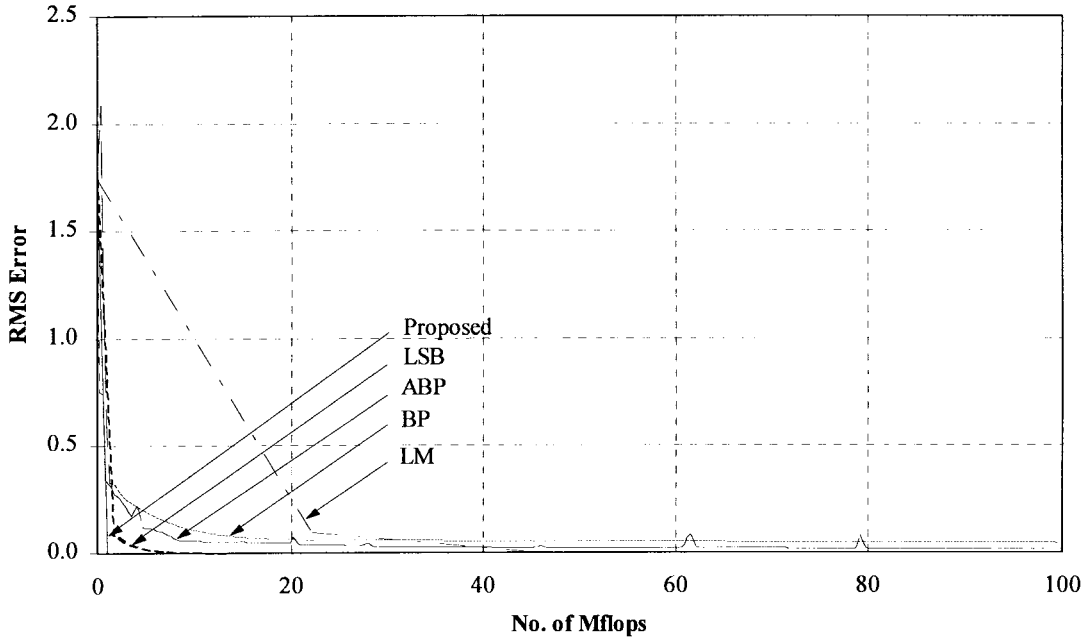

Fig. 1. The learning curves of different algorithms on the MG problem.

TABLE I
THE PERFORMANCES OF DIFFERENT ALGORITHMS ON THE NONLINEAR FUNCTION APPROXIMATION PROBLEM

Method	Mflops to achieve RMSE=0.01
Proposed	2.895
LSB	Fail
LM	58.97
ABP	118.7
BP	672.4

TABLE II
THE PERFORMANCES OF DIFFERENT ALGORITHMS ON THE MACKEY–GLASS PROBLEM

Method	Mflops to achieve RMSE=0.001	Training Error	Test Error
Proposed	1.102	0.000534	0.000601
LSB	7.751	0.000784	0.000769
LM	499.7	0.000980	0.00102
ABP	>5,000	Not Available	Not Available
BP	>5,000	Not Available	Not Available

algorithm provides the best performance. The proposed algorithm needs 2.895 Mflops on average to train the networks to the required convergence criterion, while the LM algorithm requires 58.97 Mflops. The number of flops required by the proposed algorithm is only 4.91% of that required by LM algorithm. The proposed algorithm also performs much better than the ABP algorithm in terms of the number of flops. All the networks trained by the LSB algorithm fail to achieve the specified error level and got stuck at the local minima. The mean RMSE achieved by the networks using the LSB algorithm is 0.0286. It is noticed that the computational complexity of the proposed algorithm is smaller than that of both LSB and LM algorithm. The computational complexity (in terms of number of flops per iteration) of the employed algorithms in this problem scales as Proposed: LSB: ABP: LM: BP $= 1.836{:}1.0406{:}2.677{:}201.0{:}1$.

The second problem is to learn the Mackey–Glass (MG) series [13]. A discrete-time representation of the time-series is shown as follows:

$$x(k+1) - x(k) = \frac{0.2x(k-\tau)}{1+x^{10}(k-\tau)} - 0.1x(k); \quad \tau = 17. \qquad (11)$$

A network was used to predict the value of the MG at a future time $x(k+1)$ from the most recent four consecutive data points of MG series, i.e., used the sequence $[x(k-3) \quad x(k-2) \quad x(k-1) \quad x(k)]$ to predict $x(k+1)$. Three hundred patterns were generated for training and 200 patterns were generated for testing the performance of the trained network. The number of hidden neurons is chosen to be 30, so that the network architectures are 4-30-1. The termination criterion is chosen to be RMSE $= 0.001$.

The results for MG time series are shown in Table II and Fig. 1. The proposed algorithm requires 1.102 Mflops to achieve RMSE $= 0.001$, whereas 7.751 Mflops and 499.7 Mflops are required by the LSB and LM algorithms, respectively. The total flops used by the proposed algorithm to train the network is only 0.221% of that used by the LM algorithm; the improvement is spectacular. The computational complexity of the employed algorithms in this problem scales as Proposed: ABP: LSB: LM: BP $= 3.521{:}1.0047{:}5.454{:}71.89{:}1$. The network trained by the new algorithm has the smallest test error due to the smallest training error.

The last problem is to predict the sunspots activities [14]. The network architecture is chosen to be 12-8-1 so that the input data cover the whole period. The data is divided into three groups. The first group contains the data from 1700 to 1919 and has 208 patterns. The second and third groups contains data from 1920 to 1955 and 1956 to 1979, respectively. The first is used to train the neural network. The other two groups are used as the test sets. The termination criterion is RMSE $= 0.06$.

The results for the sunspot series prediction are shown in Table III and Fig. 2. Only 5.987 Mflops are required for the proposed algorithm to train the network to the termination criterion. The flops used by the proposed algorithm is only 16.0% of that used by the LM

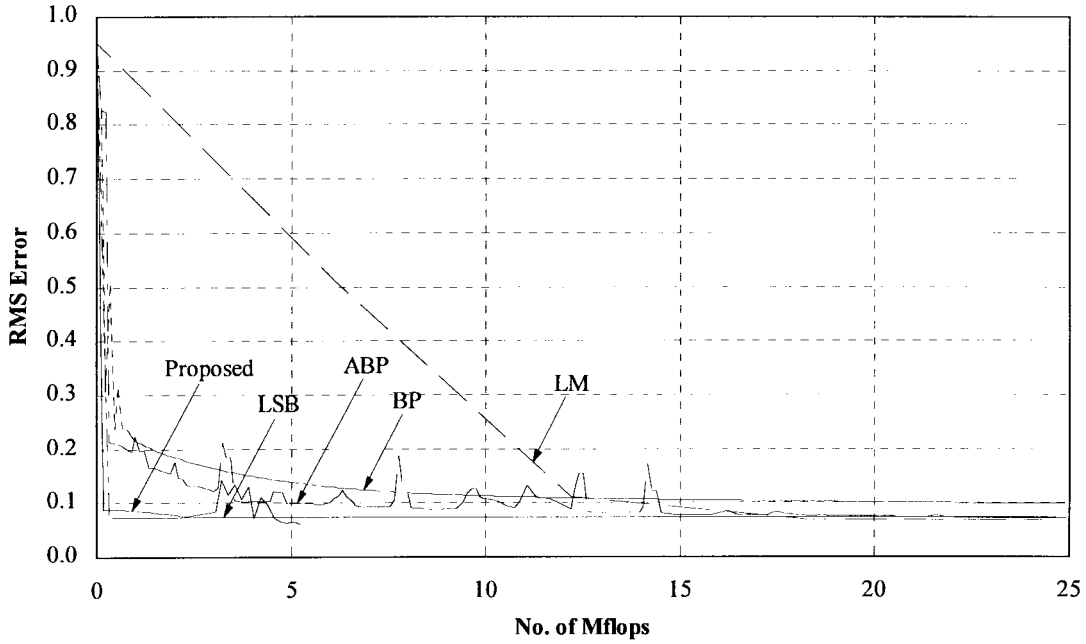

Fig. 2. The learning curves of different algorithms on the prediction of sunspots.

TABLE III
THE PERFORMANCES OF DIFFERENT ALGORITHMS ON THE PREDICTION OF SUNSPOTS

Method	Mflops to achieve RMSE=0.06	Training Error	Test 1 Error	Test 2 Error
Proposed	5.987	0.0590	0.0680	0.151
LSB	Fail	Not Available	Not Available	Not Available
LM	37.37	0.0574	0.0720	0.153
ABP	91.71	0.0600	0.0660	0.138
BP	1680	0.06	0.0665	0.139

algorithm. In the sunspot problem, the network trained by the LSB algorithm cannot achieve the termination criterion, and got stuck at the average RMSE $= 0.0842$. The computational complexity of the algorithms in this problem scales as Proposed: ABP: LSB: LM: BP $= 1.517{:}1.0082{:}2.939{:}54.77{:}1$.

IV. CONCLUSION

A training algorithm for feedforward neural networks based on the linear least squares and modified gradient descent method is developed successfully. The proposed algorithm eliminates the stalling problem of the pure linear LSB algorithms with even smaller computational effort [10]–[11]. The performance of the proposed algorithm is compared with those of other well-known fast training algorithms including LM and LSB algorithms. Experimental results show that the proposed algorithm can greatly reduce the number of flops to achieve the required accuracy. In the learning of MG time series using feedforward neural networks, the proposed training algorithm took only 1.102 Mflops, which is 0.221% of the flops taken by the LM algorithm, to reach the termination error. The total number of flops required for the network to converge using the proposed algorithm is only 14.2% of that using LSB algorithm. In the nonlinear function approximation and sunspots time series problem, the LSB algorithm cannot train the network to achieve the required error criterion, but the proposed algorithm can achieve that with the smallest Mflops. The algorithm can also be applied to networks having two or more hidden layers. Furthermore, there is no fundamental difference in generalization ability between networks trained with the proposed algorithm and the other fast training algorithms.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error backpropagation," in *Parallel Distributed Processings*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.
[2] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.
[3] T. Tollenaere, "Super SAB: Fast adaptive back propagation with good scaling properties," *Neural Networks*, vol. 3, pp. 561–573, 1990.
[4] Y. F. Yam and T. W. S. Chow, "Extended backpropagation algorithm," *Electron. Lett.*, vol. 29, no. 19, pp. 1701–1702, 1993.
[5] C. T. Leung, T. W. S. Chow, and Y. F. Yam, "A novel least third-order cumulants objective function," *Neural Processing Lett.*, vol. 3, no. 2, pp. 91–96, June 1996.
[6] E. Barnard, "Optimization for training neural nets," *IEEE Trans. Neural Networks*, vol. 3, pp. 232–240, 1992.
[7] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, vol. 5, pp. 989–993, Nov. 1994.
[8] D. A. Karras and S. J. Perantonis, "An efficient constrained training algorithm for feedforward networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 1420–1434, Nov. 1995.
[9] S. Ergezinger and E. Thomsen, "An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer," *IEEE Trans. Neural Networks*, vol. 6, pp. 31–42, Jan. 1995.
[10] F. Biegler-König and F. Bärmann, "A learning algorithm for multilayered neural networks based on linear least squares problems," *Neural Networks*, vol. 6, pp. 127–131, 1993.
[11] Y. F. Yam and T. W. S. Chow, "Accelerated training algorithm for feedforward neural networks based on least squares method," *Neural Processing Lett.*, vol. 2, no. 4, pp. 20–25, July 1995.
[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1989.
[13] M. C. Mackey and L. Glass, "Oscillations and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.
[14] T. Fröhlinghaus, A. Weichert, and P. Ruján, "Hierarchical neural networks for time-series analysis and control," *Network* 5, pp. 101–116, 1994.