International Journal of Systems Science
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tsys20

Quadratic optimization method for multilayer neural networks with local error-backpropagation

Chin-Sung Liu & Ching-Huan Tseng
Published online: 26 Nov 2010.

To cite this article: Chin-Sung Liu & Ching-Huan Tseng (1999) Quadratic optimization method for multilayer neural networks with local error-backpropagation, International Journal of Systems Science, 30:8, 889-898, DOI: 10.1080/002077299291985

To link to this article: http://dx.doi.org/10.1080/002077299291985

International Journal of Systems Science, 1999, volume 30, number 8, pages 889–898

Quadratic optimization method for multilayer neural networks with local error-backpropagation

CHIN-SUNG LIU† and CHING-HUAN TSENG†*

A fast new local error-backpropagation (LBP) algorithm is presented for the training of multilayer neural networks. This algorithm is based on the definition of a new local mean-squared error function. If the local desired outputs have been estimated, the multilayer neural networks can be decomposed into a set of adaptive linear elements (Adaline) that can be trained by quadratic optimization methods. Among a lot of quadratic optimization methods, the conjugate gradient (CG) method is one of the most famous methods that can find the global optimal solution of quadratic problems within finite steps. The iteration number and the computation time are significantly reduced because the stepsize is computed without line-search. Experimental results on the pattern recognition and memorizing of spatiotemporal patterns are provided.

1. Introduction

The batch error-backpropagation (BBP) algorithm (Rumelhart *et al.*, 1986a, b) is one of the most widely used algorithms for training multilayer neural networks. This algorithm plays an important role in the applications of pattern recognition, signal processing, control problems, etc. The training of multilayer neural networks involves adapting the weights between the computing units in the network to minimize the mean-squared errors between the desired outputs and the actual network outputs. The gradient of the mean-squared error function is computed by the errors that are backpropagated from the output layer to the hidden layers. Then the weights can be adapted by gradient descent methods.

Two major shortcomings of the BBP algorithm are the slow convergence rate and the convergence to local minima. Enormous amounts of research have been reported to improve this algorithm. Some of these research efforts were based on high order optimization methods, e.g. the conjugate gradient method (Charalambous 1992, Johansson *et al.* 1992, Møller 1993, Ergezinger and Thomsen 1995, Stäger and Agarwal 1997) and the quasi-Newton method (Battiti 1992, McLoone and Irwin 1997). Most of these optimization methods are heavily dependent on line-search. Therefore, the performance and accuracy of line-search play a critical role in the performance of these algorithms. However, accurate line-search is usually inefficient because it requires many function and gradient evaluations (Johansson *et al.* 1992). Some other research has been focused on the adaptive learning parameters without line-search, e.g. the Quickprop (Fahlman 1988), the RPROP (Riedmiller and Braun 1993) and the delta-bar-delta rule (Jacobs 1988).

The other approaches to improve the performance of the BBP algorithm are based on error estimation in all layers of multilayer neural networks. In these studies, the errors were estimated from the output layer to the hidden layers. Once the errors were estimated, the multilayer neural networks can be trained with nonlinear subproblems (Watanabe *et al.* 1991, Shah *et al.* 1992, Kimura *et al.* 1996), linear subproblems (Shah *et al.* 1992, Biegler-König and Bärmann 1993, Lou 1996, Yam and Chow 1997) or quadratic subproblems (Scalero and Tepedelenlioglu 1992, Trafalis and Couellan 1996).

Convergence to local minima is an inherent problem of gradient descent optimization methods. This problem can be overcome by global optimization methods, e.g. genetic methods (Whitley *et al.* 1990) and random methods (Brunelli 1994). However, global optimization methods have the problem of being computationally expensive. Recently, a lot of hybrid algorithms were

Received 18 September 1998; accepted 20 October 1998.

† Applied Optimum Design Laboratory, Department of Mechanical Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, R.O.C.

* Author to whom correpondence should be addressed.

employed to overcome the computationally expensive problem. For example, Adler (1993) emerged the genetic algorithms (GA) and the simulated annealing (SA) for the training of feedforward neural networks. Baba *et al.* (1994) combined the conjugate gradient method and the random optimization method to the training of neural networks. Liu and Tseng (1998) combined the perceptron learning rule with the SA technique for the training of multilayer hard-limiting perceptrons. McLoone *et al.* (1998) combined the singular value decomposition (SVD) computation to the output linear weights and the gradient-based optimization to the hidden nonlinear weights. Treadgold and Gedeon (1998) proposed a SARPROP algorithm that combined the RPROP algorithm with the SA algorithm. Combinations of different techniques can often lead to an improvement in global optimization methods, while maintaining the convergence to a global minimum.

This paper presents a new local error-backpropagation (LBP) algorithm, which combines the cheap gradient descent method with the global search technique of quadratic optimizations. The aim of this combination is to maintain the quick convergence and to reduce the likelihood of convergence to poor local minima. The main principles of this new algorithm are based on the defining of a new local mean-squared error function and the estimation of linear desired outputs for all local neurons by the gradient descent method. The multilayer neural networks can thus be decomposed into a set of adaptive linear elements (Adaline), which can be independently trained by quadratic optimization methods. The conjugate gradient (CG) method is one of the most famous quadratic optimization methods, which is only slightly more complicated than the steepest descent method. However, this convergent property can be significantly improved. Furthermore, the stepsize can be exactly computed without line-search and the second order matrix can be eliminated. Therefore, the storage requirements of the new LBP algorithm can be only slightly more than the BBP algorithm when applying the CG method as the optimization method.

This paper is organized as follows: in §2, the classical batch error backpropagation algorithm is briefly discussed. In §3, the quadratic optimization methods for Adaline are discussed. The new local error-backpropagation (LBP) algorithm is presented in §4. Finally, numerical experiments are given in §5.

2. Classical batch error-backpropagation (BBP) algorithm

The BBP algorithm is one of the most popular training algorithms for multilayer neural networks. It is a generalized LMS algorithm that minimizes the mean-squared error between the desired outputs d_{pj} and the actual outputs y_{pj} of the network. The mean-squared error function is generally defined as

$$E = \frac{1}{2P}\sum_{p=1}^{P} \|\mathbf{d}_p - \mathbf{y}_p\|^2 = \frac{1}{2P}\sum_{p=1}^{P}\sum_{j}^{n^{(L)}} [d_{pj} - y_{pj}]^2 \quad (1)$$

where P denotes the number of input–output patterns and $n^{(L)}$ denotes the node number in the output layer. The BBP algorithm is based on the gradient descent method, which adapts the weights and bias in the descent direction of the mean-squared error function (1). Specifically, the weights for a L layer neural networks are adapted by

$$\Delta w_{ji}^{(\ell)} = \alpha s_{ji}^{(\ell)}, \quad \ell = 1 \ldots L \quad (2)$$

where α denotes the learning rate and $s_{ji}^{(\ell)}$ can be computed based on the following δ-errors that are backpropagated from the output layer to the hidden layers (Rumelhart *et al.* 1986a, b). That is,

$$s_{ji}^{(\ell)} = -\frac{\partial E}{\partial w_{ji}^{(\ell)}} = \frac{1}{P}\sum_{p=1}^{P} \delta_{pj}^{(\ell)} y_{pi}^{(\ell-1)}, \quad \ell = 1 \ldots L \quad (3)$$

where

$$\delta_{pj}^{(L)} = \frac{\partial y_{pj}}{\partial u_{pj}^{(L)}}[d_{pj} - y_{pj}] \quad (4)$$

$$\delta_{pj}^{(\ell)} = \frac{\partial y_{pj}^{(\ell)}}{\partial u_{pj}^{(\ell)}} \sum_{i=1}^{n^{(\ell+1)}} w_{ij}^{(\ell+1)} \delta_i^{(\ell+1)}, \quad \ell = 1 \ldots L-1 \quad (5)$$

The nonlinear output $y_{pj}^{(\ell)}$ is related to the linear output $u_{pj}^{(\ell)}$ by $y_{pj}^{(\ell)} = f(u_{pj}^{(\ell)})$, where $f(\cdot)$ is a nonlinear activation function, e.g. the hyperbolic tangent function. The training stops when the mean-squared error is reduced to a prespecified tolerance ε for all training-vector pairs.

One simple way to improve the performance of the BBP algorithm is to smooth the weight change by adding a momentum term. For example,

$$s_{ji}^{(\ell)}(t) = -\frac{\partial E}{\partial w_{ji}^{(\ell)}}(t) + \beta s_{ji}^{(\ell)}(t-1) \quad (6)$$

where β denotes the momentum coefficient. The other approaches to improve the performance of the BBP algorithm are based on the adaptive learning rate α and momentum coefficient β (Stäger and Agarwal 1997).

3. Quadratic optimization method for adaptive linear elements

In this section, the optimization of adaptive linear elements (Adaline) is considered. As figure 1 shows, an Adaline is a single neuron with linear output u_p and

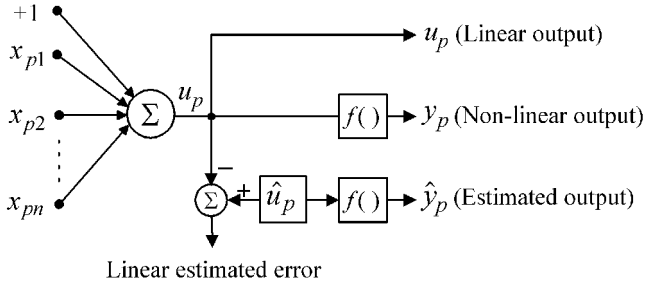

Figure 1. Adaptive linear element (Adaline) with linear estimated desired output.

nonlinear output y_p. The linear estimated error is generated by the difference between linear estimated desired output $\hat{u}_p$ and the linear output u_p. The nonlinear estimated output $\hat{y}_p$ is computed by $\hat{y}_p = f(\hat{u}_p)$, where $f(\cdot)$ is a nonlinear activation function.

It has been shown that the weight adapting of Adalines is a typical quadratic optimization problem (Widrow and Lehr 1990). Therefore, some efficient global optimization methods for quadratic problems are discussed in the following sections. Since only a single Adaline is considered in this section, the subscript and superscript are omitted in this section for convenience.

3.1. *Quadratic problem formulation*

The mean-squared error function of a single Adaline can be defined and expanded as follows.

$$\begin{aligned} E_{\mathrm{A}}(\mathbf{w}) &\equiv \frac{1}{2P}\sum_{p=1}^{P}\left[\hat{u}_p - \mathbf{x}_p^T\mathbf{w}\right]^2 \\ &= \frac{1}{2P}\sum_{p=1}^{P}\hat{u}_p^2 - \frac{1}{P}\sum_{p=1}^{P}(\hat{u}_p\mathbf{x}_p^T)\mathbf{w} \\ &\quad + \frac{1}{2P}\mathbf{w}^T\left[\sum_{p=1}^{P}\mathbf{x}_p\mathbf{x}_p^T\right]\mathbf{w} \end{aligned} \tag{7}$$

where $\mathbf{x}_p$ and $\hat{u}_p$ are pth input vector and pth estimated desired output, respectively. Defining

$$\mathbf{Q} \equiv \frac{1}{P}\sum_{p=1}^{P}\mathbf{x}_p\mathbf{x}_p^T, \quad \text{and} \quad \mathbf{b} \equiv \frac{1}{P}\sum_{p=1}^{P}[\hat{u}_p\mathbf{x}_p]$$

the mean-squared error function E_A can be expressed as the following typical quadratic function of weight $\mathbf{w}$.

$$E_{\mathrm{A}}(\mathbf{w}) = \tfrac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} - \mathbf{b}^T\mathbf{w} + c \tag{8}$$

where

$$c = \frac{1}{2P}\sum_{p=1}^{P}\hat{u}_p^2$$

can be treated as a constant during the adapting process of weights. Because the input correlation matrix $\mathbf{Q}$ is positive definite, or in the rare case, positive semi-definite (Widrow and Lehr 1990), the mean-squared error function E_{A} is a convex function (Arora 1989). Therefore, the global optimal solution $\mathbf{w}^*$ can be obtained by solving $\nabla E_{\mathrm{A}} = 0$. That is, $\mathbf{w}^* = \mathbf{Q}^{-1}\mathbf{b}$. However, the computation of the inverse matrix $\mathbf{Q}^{-1}$ will take a lot of storage and computing time. Therefore, other more efficient optimization methods, e.g. the conjugate direction methods are utilized to compute the global optimal solution $\mathbf{w}^*$.

3.2. *Conjugate direction methods*

A set of n vectors $\mathbf{s}_0, \ldots, \mathbf{s}_{n-1}$, is said to be $\mathbf{Q}$-conjugate with respect to a symmetric matrix $\mathbf{Q}$ if $\mathbf{s}_i^T\mathbf{Q}\mathbf{s}_j = 0$, $\forall i \neq j$. If the matrix $\mathbf{Q}$ is also positive definite, the set of non-zero $\mathbf{Q}$-conjugate vectors is linearly independent (Luenberger 1989) and forms a basis that spans the weight space of an Adaline. Therefore, the global optimal solution $\mathbf{w}^*$ can be expanded as

$$\mathbf{w}^* = \sum_{j=0}^{n-1}\alpha_j\mathbf{s}_j \tag{9}$$

In fact, multiplying equation (9) by $\mathbf{s}_k^T\mathbf{Q}$ and substituting $\mathbf{b}$ for $\mathbf{Q}\mathbf{w}^*$ gives

$$\alpha_k = \frac{\mathbf{s}_k^T\mathbf{Q}\mathbf{w}^*}{\mathbf{s}_k^T\mathbf{Q}\mathbf{s}_k} = \frac{\mathbf{s}_k^T\mathbf{b}}{\mathbf{s}_k^T\mathbf{Q}\mathbf{s}_k} \tag{10}$$

Therefore, if a set of $\mathbf{Q}$-conjugate search directions can be generated, all α_i can be computed simultaneously by equation (10) without line-search, and the global optimal solution $\mathbf{w}^*$ can be obtained from equation (9).

3.3. *Modified conjugate gradient method*

There are several methods that can generate a set of $\mathbf{Q}$-conjugate directions (McDowell 1983, Housos and Wing 1984). The conjugate gradient method is one of the most famous conjugate direction methods. In this method, the conjugate directions are generated as follows.

$$\mathbf{s}_0 = -\mathbf{g}_0 = \mathbf{b} - \mathbf{Q}\mathbf{w}_0 \tag{11}$$

$$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{s}_k \tag{12}$$

Then, the weights are adapted by

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k\mathbf{s}_k \tag{13}$$

where

$$\mathbf{g}_k = \nabla E_A(\mathbf{w}_k) = \mathbf{Q}\mathbf{w}_k - \mathbf{b} \tag{14}$$

$$\left.\begin{aligned}\alpha_k &= \frac{\|\mathbf{g}_k\|^2}{\mathbf{s}_k^T\mathbf{Q}\mathbf{s}_k} \quad \text{(Luenberger 1989)}\\ \beta_k &= \frac{\|\mathbf{g}_{k+1}\|^2}{\|\mathbf{g}_k\|^2} \text{(Fletcher and Reeves 1964)}\end{aligned}\right\} \tag{15}$$

The computation of the stepsize α_k can be further simplified to

$$\alpha_k = \frac{\|\mathbf{g}_k\|^2}{\frac{1}{P}\sum_{p=1}^{P}\left[\mathbf{s}_k^T\mathbf{x}_p\mathbf{x}_p^T\mathbf{s}_k\right]} = P\frac{\|\mathbf{g}_k\|^2}{\sum_{p=1}^{P}\chi_{pk}^2} \tag{16}$$

where $\chi_{pk} \equiv \mathbf{x}_p^T\mathbf{s}_k$. The computation of the gradient $\mathbf{g}_k$ can also be further simplified to

$$\mathbf{g}_k = \frac{1}{P}\sum_{p=1}^{P}\left[\left(\mathbf{x}_p^T\mathbf{w}_k - \hat{u}_p\right)\mathbf{x}_p\right] = -\frac{1}{P}\sum_{p=1}^{P}\left[\hat{e}_{pk}\mathbf{x}_p\right] \tag{17}$$

where $\hat{e}_{pk} = \left(\hat{u}_p - \mathbf{x}_p^T\mathbf{w}_k\right)$ is the local estimated error, as shown in figure 1. The modified conjugate gradient method for adapting the weight of a single Adaline without computing the correlation matrix $\mathbf{Q}$ is summarized as follows.

Step 1. Initialize the weight $\mathbf{w}_0$ with random small values. Set $k = 0$ and the initial search direction to $\mathbf{s}_0 = -\mathbf{g}_0$.

Step 2. Terminate the iterative process if any of the following stop criteria is satisfied.

(1) $k > n$;
(2) $\|\mathbf{g}_k\| \le \varepsilon_g$;
(3) $|E_A| \le \varepsilon_{Adaline}$;

where

$$\mathbf{g}_k = -\frac{1}{P}\sum_{p=1}^{P}\left[\hat{e}_{pk}\mathbf{x}_p\right] \quad \text{and} \quad E_A = \frac{1}{2P}\sum_{p=1}^{P}\hat{e}_{pk}^2$$

Step 3. Adapt the weight by

$$\mathbf{w}_{k+1} = \mathbf{w}_k + P\frac{\|\mathbf{g}_k\|^2}{\sum_{p=1}^{P}\chi_{pk}^2}\mathbf{s}_k \tag{18}$$

where $\chi_{pk} \equiv \mathbf{x}_p^T\mathbf{s}_k$.

Step 4. Adapt the search directions by

$$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \frac{\|\mathbf{g}_{k+1}\|^2}{\|\mathbf{g}_k\|^2}\mathbf{s}_k \tag{19}$$

Step 5. Set k $k+1$ and go to Step 2.

Equation (13) shows that

$$\Delta\mathbf{w}_k = \mathbf{w}_{k+1} - \mathbf{w}_k = \alpha_k\mathbf{s}_k \tag{20}$$

Therefore, it can be concluded from equations (2), (6) and (12) that the conjugate gradient method is a special case of the batch error-backpropagation method with adaptive learning rate and momentum coefficient (van der Smagt 1994), where the adaptive learning rate is computed by equation (16) and the adaptive momentum coefficient is computed by equation (15).

There are four primary advantages of the conjugate gradient method presented in this paper. First, the computation of the matrix $\mathbf{Q}$ can be eliminated in this modified method. Therefore, the computing time and storage can be significantly reduced.

Second, this method can find the global optimal solution from any initial weight within n steps if the matrix $\mathbf{Q}$ is positive definite. If the solution can be reached before n steps, the gradient $\mathbf{g}_k$ vanishes and the iterative process terminates.

Third, the stepsize can be exactly computed by equation (16) with no line-search and restart procedure. Therefore, significantly improving the performance of the weight adapting significantly improved due to the line-search will take a lot of function and gradient evaluations.

The fourth advantage of the conjugate gradient method is the especially simple formula to determine the new search direction $\mathbf{s}_k$. It is only slightly more complicated than the steepest descent method, but the convergent property can be significantly improved. Furthermore, the learning rate α_k and the momentum coefficient β_k are automatically adapted by equations (16) and (15), respectively. Therefore, the method can perform well even for an inexperienced user.

4. Local error-backpropagation (LBP) algorithm

4.1. *Local mean-squared error function*

A new mean-squared error function for multilayer neural networks is defined by

$$\widetilde{E}\left(\mathbf{W}, \hat{\mathbf{u}}\right) \equiv \sum_{\ell=1}^{L}\sum_{j=1}^{n^{(\ell)}} E_{\text{local}}\left(\mathbf{w}_j^{(\ell)}, \hat{\mathbf{u}}_j^{(\ell)}\right) \tag{21}$$

where $n^{(\ell)}$ denotes the node number of the ℓth layer network and $E_{\text{local}}\left(\mathbf{w}_j^{(\ell)}, \hat{\mathbf{u}}_j^{(\ell)}\right)$ denotes the local mean-squared error function of the jth local Adaline in the ℓth layer, which is defined by

$$E_{\text{local}}\left(\mathbf{w}_j^{(\ell)}, \hat{\mathbf{u}}_j^{(\ell)}\right) = \frac{1}{2P}\sum_{p=1}^{P}\left[\hat{u}_{pj}^{(\ell)} - u_{pj}^{(\ell)}\right]^2 \tag{22}$$

where $\hat{u}_{pj}^{(\ell)}$ is the local estimated desired output. The linear estimated output $u_{pj}^{(\ell)}$ is computed by

$$u_{pj}^{(\ell)} = \langle \mathbf{w}_j^{(\ell)}, \hat{\mathbf{y}}_p^{(\ell-1)} \rangle = \sum_{i=1}^{n^{(\ell-1)}} w_{ji}^{(\ell)} \hat{y}_{pi}^{(\ell-1)}, \ \ell = 1 \ldots L \quad (23)$$

where $\hat{y}_{pj}^{(0)}$ is the pth input pattern on the input layer network and $\hat{y}_{pi}^{(\ell-1)}$ is the nonlinear estimated input for the other layer network that is computed by

$$\hat{y}_{pi}^{(\ell-1)} = f\left(\hat{u}_{pi}^{(\ell-1)}\right), \ \ell = 2 \ldots L \quad (24)$$

The gradient of the new mean-squared error function (21) can be derived as

$$\begin{aligned} \frac{\partial \tilde{E}(\mathbf{W}, \hat{\mathbf{u}})}{\partial w_{ji}^{(\ell)}} &= \frac{\partial E_{\text{local}}\left(\mathbf{w}_j^{(\ell)}, \hat{\mathbf{u}}_j^{(\ell)}\right)}{\partial w_{ji}^{(\ell)}} \\ &= \frac{1}{P} \sum_{p=1}^{P} \left[\hat{u}_{pj}^{(\ell)} - u_{pj}^{(\ell)}\right] \frac{-\partial u_{pj}^{(\ell)}}{\partial w_{ji}^{(\ell)}} \\ &= -\frac{1}{P} \sum_{p=1}^{P} \left[\hat{u}_{pj}^{(\ell)} - u_{pj}^{(\ell)}\right] \hat{y}_{pi}^{(\ell-1)} \end{aligned} \quad (25)$$

and

$$\begin{aligned} \frac{\partial \tilde{E}(\mathbf{W}, \hat{\mathbf{u}})}{\partial \hat{u}_{pi}^{(\ell)}} &= \frac{1}{P}\left[\hat{u}_{pi}^{(\ell)} - u_{pi}^{(\ell)}\right] \\ &\quad + \frac{1}{P} \sum_{j=1}^{n^{(\ell+1)}} \left[\left(\hat{u}_{pj}^{(\ell+1)} - u_{pj}^{(\ell+1)}\right) \frac{-\partial u_{pj}^{(\ell+1)}}{\partial \hat{u}_{pi}^{(\ell)}}\right] \\ &= \frac{1}{P}\left[\hat{u}_{pi}^{(\ell)} - u_{pi}^{(\ell)}\right] \\ &\quad - \frac{1}{P} \frac{\partial \hat{y}_{pi}^{(\ell)}}{\partial \hat{u}_{pi}^{(\ell)}} \sum_{j=1}^{n^{(\ell+1)}} \left[w_{ji}^{(\ell+1)} \left(\hat{u}_{pj}^{(\ell+1)} - u_{pj}^{(\ell+1)}\right)\right] \end{aligned}$$

Figures 2 and 3 show the local estimated errors in the hidden layer and output layer, respectively. Figure 3 also shows that the linear desired output $\hat{u}_{pj}^{(L)}$ in the output layer can be computed by the inverse activation function. For instance, $\hat{u}_{pj}^{(L)}$ can be computed by using the inverse function of the hyper tangent function $y_{pj}^{(L)} = \tanh\left(\gamma u_{pj}^{(L)}\right)$. That is,

$$\hat{u}_{pj}^{(L)} = \frac{1}{2\gamma} \ln \frac{1 + d_{pj}}{1 - d_{pj}} \quad (26)$$

where d_{pj} is the actual desired output of the multilayer neural networks.

4.2. *Local error-backpropagation (LBP) algorithm*

Equation (25) shows that the gradient of the mean-squared error function (21) with respect to the weights of local Adalines are independent of each other if the estimates $\hat{\mathbf{u}}$ have been estimated. That is, the optimizations of the local mean-squared error function (22) for all local Adalines are independent of each other. Therefore, the multilayer neural networks can be decomposed into a set of Adalines that can be trained independently if $\hat{\mathbf{u}}$ have been estimated. This is particularly suited for parallel training of multilayer neural networks. The training process can stop if the mean-squared error (1) is decreased to a sufficiently small value (i.e. $E \leq \varepsilon$); otherwise, the estimates $\hat{\mathbf{u}}$ are estimated again and the training process continues. The iterative process is summarized as the local error-back-propagation (LBP) algorithm

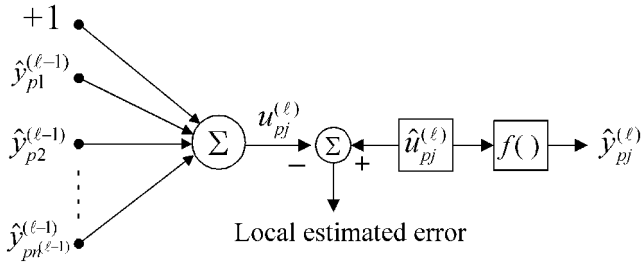

Figure 2. Estimated local error for the hidden layer neuron.

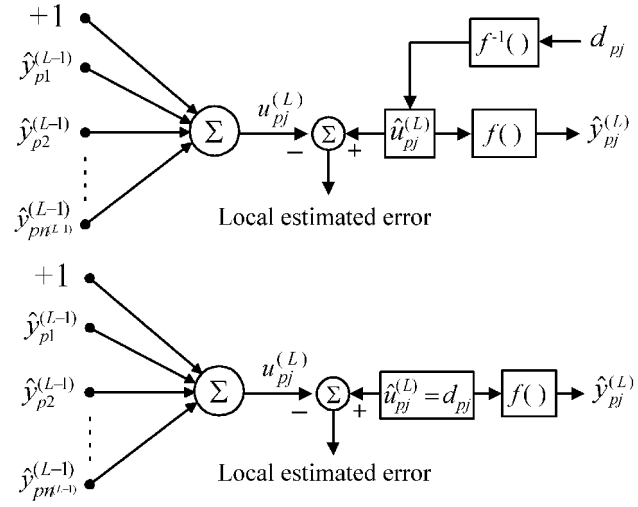

Figure 3. Estimated local error for the output layer neuron.

Step 1. Initialize all weights $\mathbf{w}_j^{(\ell)}$ and estimates $\hat{\mathbf{u}}_p^{(\ell)}$ with small random values. Set $k = 0$.

Step 2. For $p = 1 \ldots P$, adapt the estimates by $\hat{\mathbf{u}}_{pk+1}^{(\ell)} = \hat{\mathbf{u}}_{pk}^{(\ell)} + \alpha_k \mathbf{s}_k$, where α_k satisfies (Armijo 1966, Johansson *et al.* 1992)

$$\tilde{E}\left(\hat{\mathbf{u}}_{pk+1}^{(\ell)}\right) \leq \tilde{E}\left(\hat{\mathbf{u}}_{pk}^{(\ell)}\right) - \mu\alpha \|\mathbf{s}_k\|^2, \ 0 < \mu \leq 0.5 \quad (27)$$

where

$$s_{ik}^{(\ell)} = -\frac{\partial \tilde{E}(\mathbf{W}_k, \hat{\mathbf{u}}_k)}{\partial \hat{u}_{pik}^{(\ell)}} \quad (28)$$

Step 3. Adapt the weights $\mathbf{w}_{jk}^{(\ell)}$ of all Adalines independently by the quadratic optimization methods

that are discussed in §3, where $\ell = 1 \ldots L$ and $j = 1 \ldots n^{(\ell)}$.

Step 4. Stop if $E \leq \varepsilon$, or $\widetilde{E} \leq \tilde{\varepsilon}_{\text{local}}$; otherwise, set $k \quad k+1$ and go to Step 2.

The network trained by the LBP algorithm is always in a state where the new mean-squared error function (21) is at a global minimum with respect to the weights of all local Adalines because the weights are obtained by global optimization methods. Since the estimates $\hat{\mathbf{u}}$ are the auxiliary parameters that are estimated during the iterative process of the LBP algorithm, they are not the actual output of the multilayer neural networks. The LBP algorithm stops when the actual mean-squared error E is decreased to a sufficiently small value ($E \leq \varepsilon$). Therefore, the simple descent condition (27) can be efficiently applied to estimate the auxiliary parameters $\hat{\mathbf{u}}$.

There are two primary advantages of the LBP algorithm. First, only the mean-square errors of local Adalines instead of the mean-squared errors of whole networks are computed when adapting the weights. Therefore, the computing efforts can be reduced for every iteration of the LBP algorithm. Second, all $\hat{\mathbf{u}}_p$, $p = 1 \ldots P$, can be estimated independently in Step 2, then all weight $\mathbf{w}_j^{(\ell)}$ can be adapted independently in Step 3. Therefore, the LBP algorithm can be further modified for parallel training.

4.3. *The convergence of the LBP algorithm*

For a positive small value ε, we can find a sufficient small value $\varepsilon_{\text{local}}$ that satisfies the convergent condition of the LBP algorithm if all local mean-squared errors are not greater than $\varepsilon_{\text{local}}$. That is, $E \leq \varepsilon$ if $E_{\text{local}}\left(\mathbf{w}_j^{(\ell)}, \hat{\mathbf{u}}_j^{(\ell)}\right) \leq \varepsilon_{\text{local}}$ for all local Adalines, where $\ell = 1 \ldots L$ and $j = 1 \ldots n^{(\ell)}$.

Proof: Due to the continuity of the activation function in the output layer, we can find a positive value $\varepsilon^{(L)}$ that satisfies

$$E(\mathbf{W}) = \frac{1}{2P}\sum_{p=1}^{P}\sum_{j}^{n^{(L)}}\left[d_{pj} - y_{pj}\right]^2 \leq \varepsilon \tag{29}$$

if

$$\frac{1}{2P}\sum_{p=1}^{P}\left[\hat{u}_{pj}^{(L)} - u_{pj}^{(L)}\right]^2 = e_{\text{local}}^{(L)} + \sum_{p=1}^{P}\Delta_{pj}^{(L)} \leq \varepsilon^{(L)}, \quad \text{for} \quad j = 1 \ldots n^{(L)} \tag{30}$$

where $u_{pj}^{(L)}$ and y_{pj} are the actual linear and nonlinear outputs on the output layer network, respectively. The error $e_{\text{local}}^{(L)}$ is the local error of the output layer, and the $\Delta_{pj}^{(L)}$ is the error that is generated by the local errors forward propagated from the hidden layers to the output layer. Due to the continuity of the activation function in hidden layers, we can find a positive value $\varepsilon^{(\ell)}$ that satisfies

$$\sum_{p=1}^{P}\Delta_{pj}^{(L)} \leq \varepsilon^{(L)} - e_{\text{local}}^{(L)} \tag{31}$$

if

$$\frac{1}{2P}\sum_{p=1}^{P}\left[\hat{u}_{pj}^{(\ell)} - u_{pj}^{(\ell)}\right]^2 \leq \varepsilon^{(\ell)} \tag{32}$$

for all Adalines in the hidden layers. Therefore, equation (30) can be satisfied if $\varepsilon_{\text{local}}$ is chosen as

$$\varepsilon_{\text{local}} = \min\left(\varepsilon^{(\ell)},\ \varepsilon^{(L)}\right) \tag{33}$$

Since the descent optimization algorithms are applied to Steps 2 and 3 of the LBP algorithm, the sequence $\{\widetilde{E}^{(k)}\}$ generated by the LBP algorithm is a non-increasing sequence. In addition, $\widetilde{E}$ is a continuous differentiable and lower bounded function ($\widetilde{E} \geq 0$). Therefore,

$$\lim_{k \to \infty}\left\{\widetilde{E}^{(k+1)} - \widetilde{E}^{(k)}\right\} = 0 \tag{34}$$

That is, an optimal or stationary solution can be approached as $k \to \infty$. Furthermore, if $E_{\text{local}} \leq \varepsilon_{\text{local}}$ for all Adalines, the final convergent condition (i.e. $E \leq \varepsilon$) can be satisfied.

5. Numerical experiments

To demonstrate the performance of the new LBP algorithm, the following methods are compared:

- LBP-CG: local error-backpropagation algorithm with modified conjugate gradient optimization method;
- BBP: batch error-backpropagation algorithm with fixed learning rate;
- SD: batch error-backpropagation algorithm with adaptive learning rate (Stäger and Agarwal 1997);
- CG: conjugate gradient algorithm with line-search and restart procedure (Stäger and Agarwal 1997).

These algorithms are compared for three problems. The first two problems are the pattern classification and recognition problems. The third problem is the memorizing of spatiotemporal patterns with various grey levels. All experiments are programmed in Visual C++ and run on a Pentium MMX 200 MHz PC machine with 64 Mbytes of RAM memory. The LBP-CG algorithm is tested with over 1000 various initial weights. All experiments can converge within a small number of iterations.

Table 1. Experimental results for the printed character classification with $P = 52$, $\xi_{local} = 10^{-2}$ and $\varepsilon = 10^{-5}$.

	LBP–CG without attenuation				SD			CG		
$n^{(1)}$	k	$\tilde{E}(\times 10^{-2})$	$E(\times 10^{-5})$	T(s)	k	$E(\times 10^{-5})$	T(s)	k	$E(\times 10^{-5})$	T(s)
10	9	0.9231	0.1473	1.53	5197	0.9996	46.63	1818	0.9997	44.43
20	3	0.8324	0.2133	1.43	4276	0.9999	80.08	854	0.9987	33.18
30	4	0.8340	0.0213	2.36	3985	0.9998	111.34	612	0.9974	41.85
52	2	0.8113	0.0052	3.57	3769	0.9994	180.48	415	0.9991	50.04

A drawback of the LBP algorithm is the strongly fluctuating values for the weights at the output layer. To avoid this difficulty, it is advisable to allow only small changes in the weights $\mathbf{w}^{(L)}$ (Biegler-König and Bärmann 1993). The huge values for the weights can be avoided by using this 'attenuation' for the change in the weights. Therefore, the generalization performance can be significantly improved.

5.1. *Printed character classification*

The printed character classification is a typical application of multilayer neural networks. To compare the performance of the LBP–CG algorithm with other algorithms, a set of printed characters is used as the input patterns for a two-layer neural network. The training set consists of 52 letters that are eight column × eight row pixel matrix characters. The outputs are the ASCII code of these characters. There are 64 input nodes for the input of character pixel and seven output nodes for the output of the ASCII code. The outputs are set to 0.9999 or − 0.9999. Although the experiment is tested on the patterns of two levels, similar results can be obtained from the patterns with various grey levels. Table 1 shows the experimental results for various hidden node numbers $n^{(1)}$. It can also be observed from table 1 that the LBP–CG algorithm can converge within a much smaller number of iterations than the other algorithms. Therefore, the performance of the LBP–CG algorithm can be significantly improved over the other three compared algorithms.

5.2. *Grey pattern recognition*

In this experiment, two-layer neural networks are trained to recognize 32 × 32 input patterns of 16-bit grey-level pixel. The outputs are the identity numbers of these patterns. There are 1024 input nodes for the input of patterns and 7 output nodes for the outputs of identity number. Table 2 shows the experimental results for various hidden node numbers $n^{(1)}$. It can also be observed from table 2 that the LBP–CG algorithm can converge within a small number of iterations in this experiment. Furthermore, the testing mean-squared error E_t for the patterns with 5% noise pixel can be significantly reduced for the LBP–CG algorithm with attenuation, although the convergence is still nearly as fast as the LBP–CG algorithm without attenuation. Figure 4 shows the typical learning curves for the LBP–CG algorithm and other algorithms.

The experimental results for the large-scale pattern recognition problems are shown in table 3. The LBP–CG algorithm can converge within a small iteration number. However, the other three compared algorithms are very difficult to converge in this experiment. It can also be observed in this table that the testing mean-squared error E_t for the patterns with 5% noise pixel can be significantly reduced for the LBP–CG algorithm with attenuation.

5.3. *Memorizing of spatiotemporal patterns*

Spatiotemporal patterns are an ordered set of sequential patterns. Full recurrent multilayer neural networks can be used to memorize the spatiotemporal patterns (Lin and Lee 1996). The spatiotemporal patterns can be binary pixel patterns or grey-level patterns. In this experiment, a set of 16-bit grey-level and random generated patterns is stored in the two-layer full recurrent neural networks. The experimental results for storing various numbers of patterns are shown in tables 4 and 5, with patterns of 8 × 8 and 64 × 64 grey-level pixel,

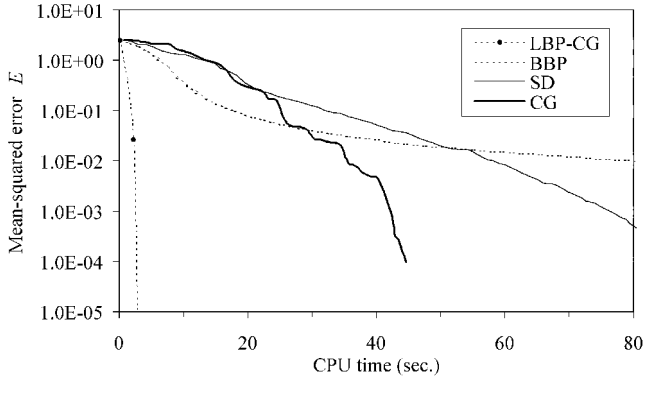

Figure 4. Learning curves of the pattern recognition problem with 30 hidden nodes and twenty 32×32 grey-level patterns.

Table 2. Experimental results for the 32×32 pixel pattern recognition with $\varepsilon_{local} = 10^{-1}$ and $\varepsilon = 10^{-4}$.

(a)

		LBP–CG without attenuation					LBP–CG with attenuation				
P	$n^{(1)}$	k	$\tilde{E}$	$E(\times 10^{-4})$	E_t	T(s)	k	$\tilde{E}$	$E(\times 10^{-4})$	E_t	T(s)
20	10	3	0.0603	0.6471	1.2528	1.15	3	0.4561	0.0005	0.00540	1.34
	30	1	0.0226	0.0310	0.6405	1.38	2	0.7184	0.0007	0.00101	3.19
	50	1	0.0002	0.0004	0.0345	1.88	2	0.7799	0.0300	0.00007	5.47
100	30	1	0.1362	0.0001	0.3214	12.94	1	1.3459	0.0477	0.10431	13.21
	50	1	2.7591	0.8441	0.3307	12.81	1	0.1907	0.0002	0.04392	22.48

(b)

		SD				SD			
P	$n^{(1)}$	k	$E(\times 10^{-4})$	E_t	T(s)	k	$E(\times 10^{-4})$	E_t	T(s)
20	10	5819	0.9995	0.00134	779.18	1231	0.9514	0.03969	205.69
	30	199	0.9935	0.00066	88.54	95	0.9455	0.00059	44.76
	50	177	0.9637	0.00071	153.58	40	0.9333	0.00070	33.06
100	30	854	0.9978	0.04129	2463.21	125	0.9614	0.03414	398.01
	50	695	0.9965	0.02769	4333.30	141	0.9856	0.02542	819.26

Table 3. Experimental results for the large-scale pattern recognition with $P = 20$ and $\varepsilon = 10^{-4}$ (128×128 and 256×256 pixel)

		LBP–CG without attentuation					LBP–CG with attenuation				
$n^{(0)}$	$n^{(1)}$	k	$\tilde{E}$	$E(\times 10^{-4})$	E_t	T(s)	k	$\tilde{E}$	$E(\tilde{1}0^{-4})$	E_t	T(s)
128 × 128	10	3	1.87472	0.08755	1.6625	25.74	4	1.6682	0.8225	0.3034	32.39
128 × 128	20	1	0.11355	0.00006	3.3332	12.55	3	2.1486	0.1881	0.1751	52.29
128 × 128	30	1	0.00001	0.00001	3.2091	19.48	3	2.2410	0.5654	0.0068	79.81
256 × 256	20	1	0.00002	0.67760	4.4690	242.68	1	2.2477	0.1443	0.8666	267.30
256 × 256	30	1	0.00002	0.0000004	3.4089	73.09	2	1.1567	0.0216	0.7025	234.51

respectively. Typical learning curves for the memorizing of spatiotemporal patterns are shown in Figure 5.

It can be observed from Tables 4 and 5 that the LBP–CG algorithm is more efficient compared to the other algorithms. It can also be observed that the LBP–CG algorithm can converge in a few iterations. As shown in §4.3, the mean-squared error E can decrease to a sufficiently small value (i.e. $E \leq \varepsilon$) if $E_{local} \leq \varepsilon_{local}$ for all Adalines. That is, the LBP-CG algorithm can converge if all local mean-squared errors are sufficiently small; otherwise, the iterative process continues. In the case where the hidden layer exists with more nodes than the number of patterns to be learned, the LBP algorithm can converge in 1 iteration (without attenuation) or within 10 iterations (with attenuation).

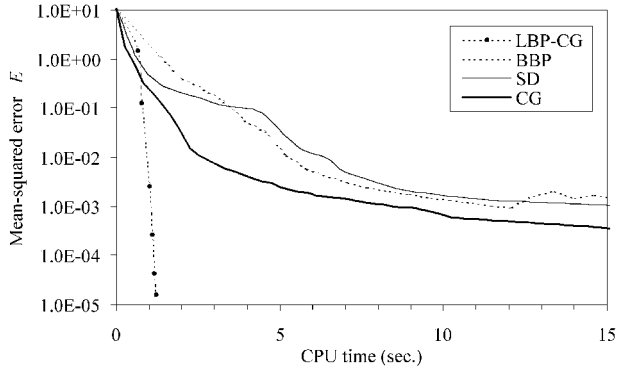

Figure 5. Learning curves of memorizing spatiotemporal patterns with 12 hidden nodes and ten 8×8 grey-level patterns.

6. Conclusions

In this paper, a new local error-backpropagation (LBP) algorithm is presented for the training of multilayer neural networks. From the estimation of the local errors, the multilayer neural networks can be decomposed into a set of adaptive linear elements, which can then be trained by quadratic optimization methods. The modified conjugate gradient method is selected as the quadratic optimization method due to the method that

Table 4. Experimental results for the memorizing of 8×8 pixel spatiotemporal patterns ($\tilde{\varepsilon}_{local} = 10^{-5}$ and $\varepsilon = 10^{-4}$)

		LBP–CG with attenuation				SD			CG		
P	$n^{(1)}$	k	$\tilde{E}(\times 10^{-5})$	$E(\times 10^{-4})$	T(s)	k	$E(\times 10^{-4})$	T(s)	k	$E(\times 10^{-4})$	T(s)
10	12	6	0.9738	0.1602	1.22	14168	0.9991	88.82	631	0.9979	17.52
20	24	8	0.7340	0.1517	2.20	14373	0.9933	276.49	1380	0.9987	152.97
30	36	9	0.6187	0.1560	4.27	14685	0.9988	622.25	2280	0.9997	487.30

Table 5. Experimental results for the memorizing of 64×64 pixel spatiotemporal patterns ($\tilde{\varepsilon}_{local} = 10^{-4}$ and $\varepsilon = 10^{-4}$)

		LBP–CG with attenuation				SD		CG	
P	$n^{(1)}$	k	$\tilde{E}(\times 10^{-4})$	$E(\times 10^{-4})$	T(s)	k	T(s)	k	T(s)
10	12	8	0.12414	0.06667	28.44	20000*	12550.16*	1000*	10249.87*
20	24	7	0.09480	0.05931	91.27	20000*	42269.12*	1000*	27158.50*
30	36	7	0.06738	0.04797	171.38	20000*	83382.01*	1000*	53418.35*

*Fail to converge.

can find the global optimal solution within finite steps. In addition, the second order matrix **Q** can be eliminated in this method and the stepsize can be exactly computed without the line-search and restart procedure. Three experiments are tested on this new LBP algorithm, and it is shown that the new algorithm can perform better than the other three compared algorithms.

Acknowledgment

The research reported in this paper was supported under a project sponsored by the National Science Council Grant, Taiwan, R.O.C., NSC-86-2212-E-009-030.

References

Adler, D., 1993, Genetic algorithms and simulated annealing: a marriage proposal. *Proceedings of the IEEE International Conference on Neural Networks*, Vol. II, San Francisco, CA, pp. 1104–1109.

Armijo, L., 1966, Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal on Mathematics*, **16**, 1–3.

Arora, J. S., 1989, Introduction to Optimum Design (New York: McGraw-Hill), pp. 139–142.

Baba, N., Mogami, Y., Kohzaki, M., Shiraishi, Y., and Yoshida, Y., 1994, A hybrid algorithm for finding the global minimum of error function of neural networks and its applications. *Neural Networks*, **7**, 1253–1265.

Battiti, R., 1992, First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, **4**, 141–166.

Biegler-König, F., and Bärmann, F., 1993, A learning algorithm for multilayered neural networks based on linear least squares problems. *Neural Networks*, **6**, 127–131.

Brunelli, R., 1994, Training neural nets through stochastic minimization. *Neural Networks*, **7**, 1405–1412.

Charalambous, C., 1992, Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings Part G*, **139**, 301–310.

Ergezinger, S., and Thomsen, E., 1995, An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer. *IEEE Transactions on Neural Networks*, **6**, 31–42.

Fahlman, S. E., 1988, Faster-learning variations on backpropagation: an empirical study. *Proceedings of the 1988 Connectionist Models Summer School*, pp. 38–51.

Fletcher, R., and Reeves, C. M., 1964, Function minimization by conjugate gradients, *Computer Journal*, **7**, 149–154.

Housos, E. C., and Wing, O., 1984, Pseudo-conjugate directions for the solution of the nonlinear unconstrained optimization problem on a parallel computer. *Journal of Optimization Theory and Applications*, **42**, 169–180.

Jacobs, R. A., 1988, Increased rates of convergence through learning rate adaptation. *Neural Networks*, **1**, 295–307.

Johansson, E. M., Dowla, F. U., and Goodman, D. M., 1992, Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, **2**, 291–301.

Kimura, A., Arizono, I., and Ohta, H., 1996, An improvement of a back propagation algorithm by the extended Kalman filter and demand forecasting by layered neural networks. *International Journal of Systems Science*, **27**, 473–482.

Lin, C. T., and Lee, C. S. G., 1996. *Neural fuzzy systems* (Prentice-Hall), pp. 349–350.

Liu, C. S., and Tseng, C. H., 1998, Simulated annealing perceptron training algorithm for multilayer discrete-perceptrons. Tech. Report AODL 98-1, Department of Mechanical Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Lou, K. N., 1996, A new system identification technique using Kalman filtering and multilayer neural networks. *Artificial Intelligence in Engineering*, **10**, 1–8.

Luenberger, D. G., 1989, Linear and nonlinear programming. 2nd edn (Menlo Park: Addison-Wesley), pp. 238–246.

McDowell, D. G., 1983, Generalized conjugate directions for unconstrained function minimization. *Journal of Optimization Theory and Applications*, **41**, 523–531.

McLoone, S., Brown, M. D., Irwin, G., and Lightbody, G., 1998, A hybrid linear/nonlinear training algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, **9**, 669–684.

McLoone, S., and Irwin, G. W., 1997, Fast parallel off-line training of multilayer perceptrons. *IEEE Transactions on Neural Networks*, **8**, 646–653.

Møller, M. F., 1993, A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, **6**, 525–533.

Riedmiller, M., and Braun, H., 1993, A direct adaptive method for faster backpropagation learning: the RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks (ICNN 93)*, Vol. I, San Francisco, CA, pp. 586–591.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986a, Learning internal representations by error propagation, In D. E. Rumelhart and J. L. McClelland (eds) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. I* (Cambridge, MA: MIT Press), pp. 318–362.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986b, Learning representations by back-propagation errors. *Nature*, **323**, 533–536.

Scalero, R. S., and Tepedelenlioglu, N., 1992, A fast new algorithm for training feedforward neural networks. *IEEE Transactions on Signal Processing*, **40**, 202–210.

Shah, S., Palmieri, F., and Datum, M., 1992, Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, **5**, 779–787.

Stäger, F., and Agarwal, M., 1997, Three methods to speed up the training of feedforward and feedback perceptrons. *Neural Networks*, **10**, 1435–1443.

Trafalis, T. B., and Couellan, N. P., 1996, Neural network training via an affine scaling quadratic optimization algorithm. *Neural Networks*, **9**, 475–481.

Treadgold, N. K., and Gedeon, T. D., 1998, Simulated annealing and weight decay in adaptive learning: the SARPROP algorithm. *IEEE Transactions on Neural Networks*, **9**, 662–668.

van der Smagt, P. P., 1994, Minimisation methods for training feedforward neural networks. *Neural Networks*, **7**, 1–11.

Watanabe, K., Fukuda, T., and Tzafestas, S. G., 1991, Learning algorithms of layered neural networks via extended Kalman filters. *International Journal of Systems Science*, **22**, 753–768.

Whitley, D., Starkweather, T., and Bogart, C., 1990, Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, **14**, 347–361.

Widrow, B., and Lehr, M. A., 1990, 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation. *Proceedings of the IEEE*, **78**, 1415–1442.

Yam, Y. F., and Chow, T. W. S., 1997, Extended least squares based algorithm for training feedforward networks. *IEEE Transactions on Neural Networks*, **8**, 806–810.