An Adaptive Conjugate Gradient Learning Algorithm for Efficient Training of Neural Networks

H. Adeli and S. L. Hung*

College of Engineering
and Center for Cognitive Science
The Ohio State University
470 Hitchcock Hall, 2070 Neil Avenue
Columbus, Ohio 43210-1275

Transmitted by Melvin Scott

ABSTRACT

An adaptive conjugate gradient learning algorithm has been developed for training of multilayer feedforward neural networks. The problem of arbitrary trial-and-error selection of the learning and momentum ratios encountered in the momentum backpropagation algorithm is circumvented in the new adaptive algorithm. Instead of constant learning and momentum ratios, the step length in the inexact line search is adapted during the learning process through a mathematical approach. Thus, the new adaptive algorithm provides a more solid mathematical foundation for neural network learning. The algorithm has been implemented in *C* on a SUN-SPARCstation and applied to two different domains: engineering design and image recognition. It is shown that the adaptive neural networks algorithm has superior convergence property compared with the momentum backpropagation algorithm.

1. INTRODUCTION

The momentum backpropagation (BP) learning algorithm [14] is widely used for training multilayer neural networks for classification problems. This algorithm, however, has a slow rate of learning. In order to improve the convergence rate and reduce the total number of iterations and consequently the execution time, more effective neural network learning algorithms have to

*Adeli is a professor and Hung is a Ph.D. candidate.

655 Avenue of the Americas, New York, NY 10010 0096-3003/94/$7.00

be developed. Kollias and Anastassiou [10] developed an adaptive least-squares learning algorithm for multilayer neural networks. Douglas and Meng [5] developed an adaptive linearized least-squares learning algorithm for training of multilayer feedforward neural networks. These two algorithms achieved better convergence rate than the momentum BP learning algorithm by using second order derivatives of the error function with respect to the network weights. However, in both of these algorithms, the Hessian matrix (H), containing the second order derivatives of the network weights, is needed, requiring a large amount of memory storage and additional computations. These two algorithms are efficient only when the input data set is small.

The conjugate gradient method, originally proposed by Fletcher and Reeves [6], has been recognized as one of the few practical methods for solving large optimization problems because it does not require any large matrix storage and its iteration cost is relatively low [13]. In this work, an adaptive conjugate gradient learning algorithm has been developed for training of multilayer feedforward neural networks and implemented in C on a SUN-SPARCstation. Two examples have been used to test the performance of the developed learning algorithm. The first example is taken from the domain of engineering design. A small neural network with 50 links is used for this example. The other example is from the domain of image recognition. A large neural network with 5950 links is used for this example.

2. SUPERVISED LEARNING FOR FEEDFORWARD NEURAL NETWORKS

A classification system is said to be supervised when, given a set of objects with known classifications, it can classify unknown objects based on the information acquired by training. Mathematically, it may be stated as follows: given a set of real input vectors $X_k \in R^{n_I}$ as well as real output vector $Y_k \in R^{n_N}$ ($K = 1, 2, \ldots, p$,) find the function f: $R^{n_I} \rightarrow R^{n_N}$ that maps X_k to Y_k. The total number of training instances is denoted by p. R^{n_I} and R^{n_N} are sets of real numbers with dimensions n_I and n_N, respectively. In our domain of interest, n_I and n_N are the total numbers of patterns in the input and output sets, respectively. The mapping process can be classified as a mathematical optimization problem: find an optimum decision vector W to minimize an objective function. The objective function for this optimization problem is the sum of squared error function (the difference between the desired and computed outputs) as

$$E(X_k, W) = \frac{1}{2p} \sum_{k=1}^{p} \sum_{m=1}^{n_N} (y_{km} - o_{km})^2, \tag{1}$$

where y_{km} and o_{km} are the desired and computed outputs for the mth output and the kth training instance, respectively.

Feedforward neural networks have been widely used to perform this kind of mapping [5, 10, 14]. A multilayer feedforward neural network is shown in Figure 1. It contains n_1 input nodes, n_N output nodes, and n_i nodes in the $(i-1)$st hidden layer, respectively. The nodes between the ith and $(i+1)$st layers are fully connected. A weight $w_{j,k}$ is assigned to the link between nodes j and k in two connected layers. An input vector is propagated feedforwardly through the network. Except for the nodes of input layer, the input to each node is the sum of the weighted output of the prior layer modified by a nonlinear sigmoidal function. Following the notation of the neural network in Figure 1, the output vectors of the kth training instance for layers 1 to N can be represented by the following equations:

$$O_k^{(1)} = \begin{bmatrix} X_k \\ 1 \end{bmatrix} \tag{2}$$

$$O_k^{(i+1)} = \begin{bmatrix} G\left(W^{(i)} \cdot O_k^{(i)}\right) \\ 1 \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, N-1. \tag{3}$$

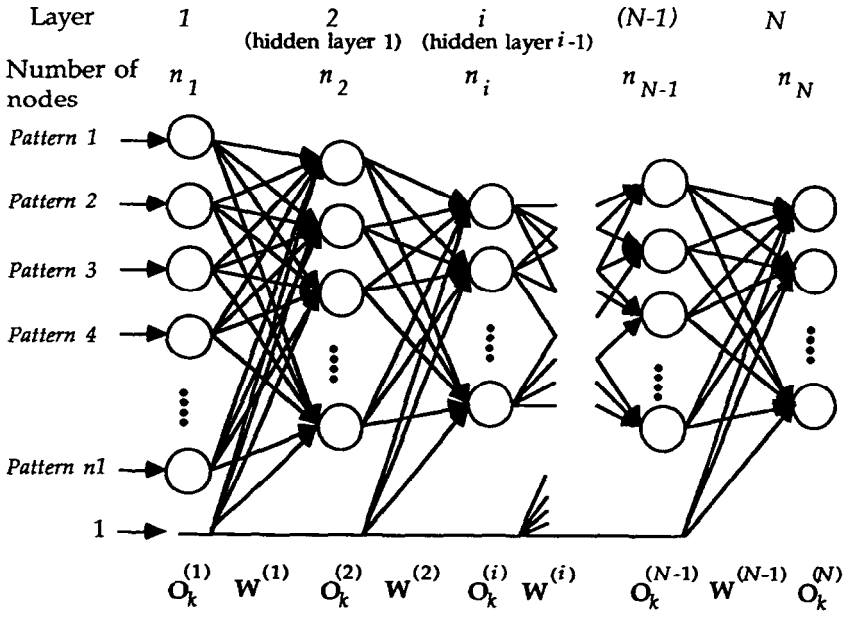


FIG. 1. A multilayer feedforward neural network.

The matrix of weights between layers i and $i + 1$ can be written as

$$W^{(i)} = \begin{bmatrix} w_{1,1}^{(i)} & \cdots & w_{1,n_i+1}^{(i)} \\ \vdots & & \vdots \\ w_{n_{i+1},1}^{(i)} & \vdots & w_{n_{i+1},n_i+1}^{(i)} \end{bmatrix}_{(n_{i+1})(n_i+1)} = \begin{bmatrix} W_1^{(i)T} \\ \vdots \\ W_{n_{i+1}}^{(i)T} \end{bmatrix} \tag{4}$$

where T indicates the transpose of a matrix and $w_{t,n_{i+1}}^{(i)}$ is the threshold value for node t in the ith layer. $W_j^{(i)T}$ is a row vector containing the weights of the links connected to the jth node in the ith layer. The following sigmodial function is used as the threshold function:

$$g(u) = \frac{1}{1 + e^{-u}}. \tag{5}$$

Therefore,

$$G\left(W^{(i)} \cdot O_k^{(i)}\right) = \begin{bmatrix} g\left(\sum_{j=1}^{n_i+1} \left(w_{1,j}^{(i)} o_{kj}^{(i)}\right)\right) \\ g\left(\sum_{j=1}^{n_i+1} \left(w_{2,j}^{(i)} o_{kj}^{(i)}\right)\right) \\ \vdots \\ g\left(\sum_{j=1}^{n_i+1} \left(w_{n_{i+1},j}^{(i)} o_{kj}^{(i)}\right)\right) \end{bmatrix}. \tag{6}$$

Now, we define the aforementioned vector $W \in R^L$ containing all the weights of the feedforward neural network represented in (4) as

$$\begin{aligned} W &= \left\{W_1^{(1)T} W_2^{(1)T} \cdots W_{n_i}^{(i-1)T} W_1^{(i)T} \cdots W_{(n_N-1)}^{(N-1)T} W_{n_N}^{(N-1)T}\right\}^T \\ &\equiv \left\{w_{1,1}^{(1)} w_{1,2}^{(1)} \cdots w_{n_2,n_1+1}^{(1)} w_{1,1}^{(2)} \cdots w_{j,k}^{(i)} \cdots w_{1,1}^{(N-1)} w_{1,2}^{(N-1)} \cdots w_{n_N,n_{N-1}+1)}^{(N-1)}\right\}^T, \end{aligned} \tag{7}$$

where L is the total number of links given by

$$L = \sum_{i=1}^{N-1} (n_{i+1})(n_i + 1). \tag{8}$$

Note that W is a column matrix with L elements. The functional mapping between the input vector X_k and the computed vector O_k for the kth training instance can be written as

$$f(X_k, W) = O_k. \tag{9}$$

The vector W in (9) can be solved either iteratively or directly by finding the pseudoinverse matrix of X_k. However, the latter approach is inconsistent with the biological learning process. The reason is that the learning process in a biological neural network is an improvement process rather than an analytical procedure. Therefore, in order to simulate the human learning process, (9) is solved iteratively in order to find an optimum weight vector W^* minimizing the total system error represented by (1).

3. EFFECTS OF LEARNING AND MOMENTUM RATIOS ON THE CONVERGENCE

Most of the gradient-based nonlinear unconstrained optimization algorithms consist of two stages. One stage is finding the search direction. Determination of the vector of search direction d is dependent on the chosen optimization algorithm. In the steepest gradient descent search, the vector of search direction is set as the steepest descent direction $d = -\nabla E(W)$. In this work, the search direction is set as the conjugate direction of the steepest descent search direction, as defined in the following section. The other stage is the step length size determination that seeks a scalar λ^* to minimize the function $E(W + \lambda d)$. Once the search direction is determined, the step size becomes a one-dimensional minimization problem.

In the momentum BP learning algorithm a constant step length (λ) is used. Thus, it consists of finding the direction of search only. Such an algorithm has a slow and irregular convergence rate. Moreover, the convergence rate is highly dependent on the choice of the values of learning ratio (λ) and momentum ratio (α). The proper values of these two ratios are dependent on the type of problem. As an example, the convergence rates for the problem of minimum weight design of steel beams using four different values for the pair (λ, α) are shown in Figure 2. It is observed that convergence rate is quite unacceptable for some values of the pair (λ, α).

4. INEXACT LINE SEARCH ALGORITHM

The stage of the step length size determination (line search) has a great effect on the efficiency of the mathematical optimization algorithm. A very

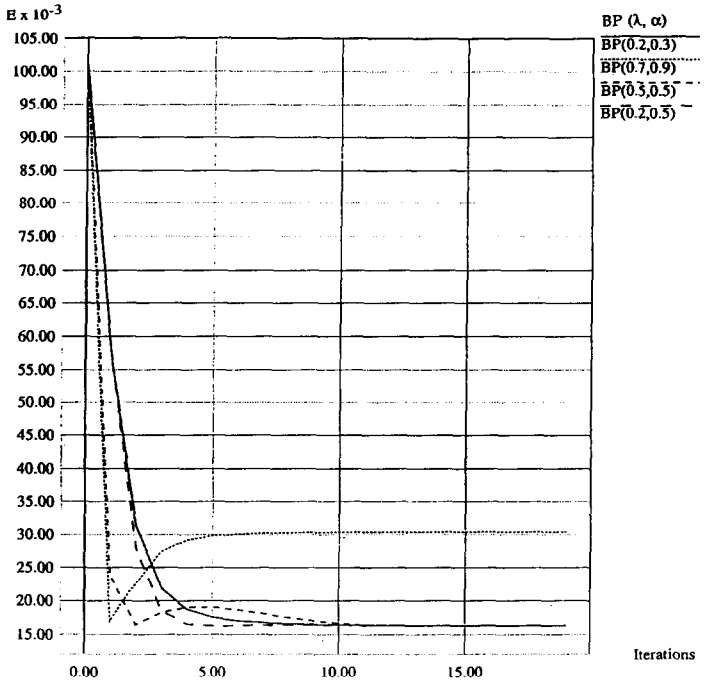


FIG. 2. Learning performance for the problem of minimum weight design of steel beams by momentum BP learning algorithm with different pairs of (λ, α).

accurate line search algorithm may reduce the total number of iterations but usually needs many function evaluations. Therefore, if the computation of the objective function for the optimization problem is expensive, the performance of an optimization algorithm using an inexact line search strategy is better than that of an optimization algorithm using an exact line search strategy. In our previous work, we found that the computation of the objective function in the feedforward neural network is the most time-consuming step in each iteration. Therefore, in this work, an inexact line search has been utilized.

An inexact (approximate) line search algorithm can determine the search parameter (step length) λ within a small percentage of its true value. Two useful criteria for terminating an inexact line search process have been proposed by Armijo [3] and Goldstein [7]. These two criteria are used to guarantee that the selected step length λ is neither too large nor too small. According to the first criterion [3], in order to ensure that the selected step

length λ is not too large, it has to satisfy the condition

$$E(W^{(k)} + \lambda d^{(k)}) \leqslant E(W^{(k)}) + \beta\lambda\left(\nabla E(W^{(k)})^T d^{(k)}\right),$$

$$\beta \in (0, 1) \text{ and } \lambda > 0. \quad (10)$$

The superscript k refers to the kth iteration.

According to the second criterion [7], in order to ensure that the selected step length λ is not too small, it has to satisfy the condition

$$\nabla E(W^{(k)} + \lambda d^{(k)})^T d^{(k)} \geqslant \theta\left(\nabla E(W^{(k)})^T d^{(k)}\right), \qquad \theta \in (\beta, 1) \text{ and } \lambda > 0. \quad (11)$$

The condition, $1 > \theta > \beta > 0$, guarantees that (10) and (11) can be satisfied simultaneously [4]. However, the two conditions represented by (10) and (11) do not guarantee that descent directions are always generated. Therefore, we apply a third condition to ensure that the selected step length λ satisfies the descent condition [11]:

$$\nabla E(W^{(k)} + \lambda d^{(k)})^T d^{(k+1)} < 0. \quad (12)$$

The three aforementioned conditions are described graphically in Figure 3. The acceptable step length λ is located in a region that satisfies the three conditions. Dennis and Schnable [4] developed an inexact line search algorithm by backtracking, using successive parabolic and cubic interpolations. Our inexact line search algorithm is based on the integration of (10)–(12) and backtracking by successive parabolic and cubic interpolations.

5. AN ADAPTIVE CONJUGATE GRADIENT NEURAL NETWORK LEARNING ALGORITHM

Proposed by Fletcher and Reeves [6] and modified by Polak-Ribiére [12], the conjugate gradient method is an effective modification of the steepest descent method. Furthermore, Powell [13] showed that the unrestarted Polak-Ribiére method with exact line search may fail to converge for nonconvex problems and proposed a more robust algorithm to ensure convergence. We present an adaptive conjugate gradient neural network learning algorithm

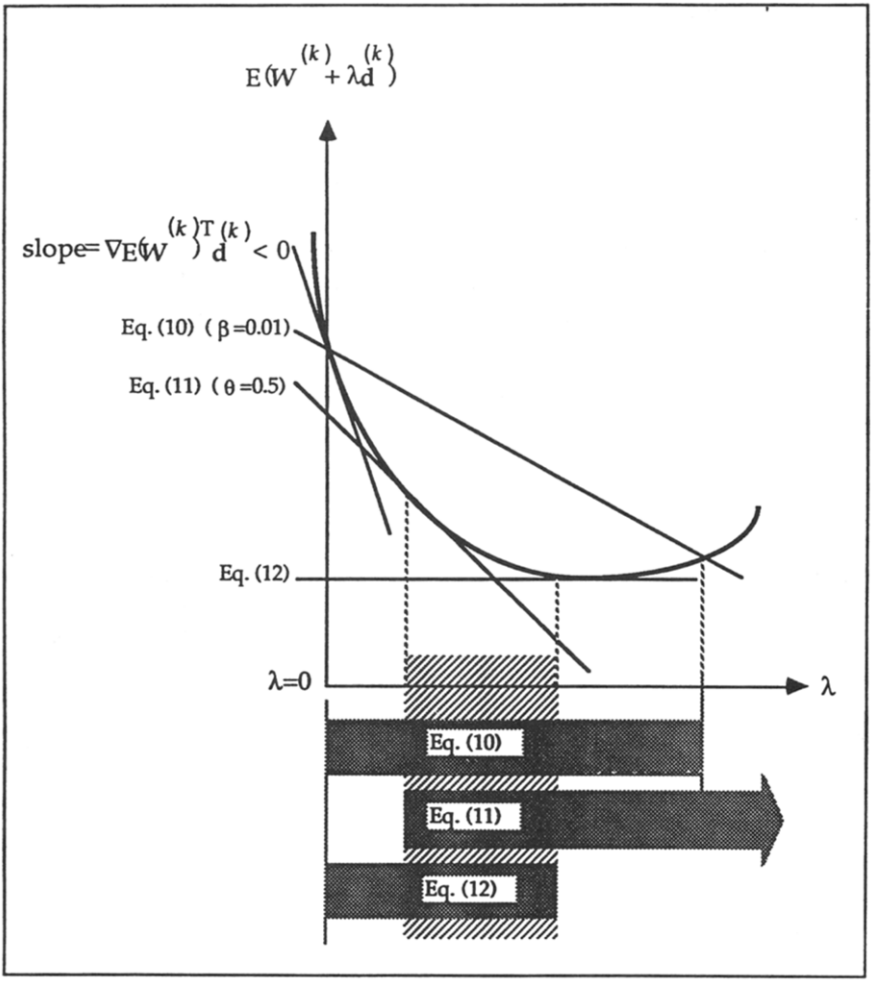


FIG. 3. Permissible values of λ under three line search conditions.

by using the Powell's modified conjugate gradient algorithm for minimizing the system error in neural networks with the inexact line search algorithm described in the previous section.

Our optimization problem has L decision variables.

Step 1. Generate a starting weight vector $W^0 \in R^L$ randomly. Set the iteration counter $n = 1$ and three different stopping criteria: the

convergence parameter for the gradient vector $\varepsilon(=10^{-6})$, maximum number of iterations (*MaxIter*), and the minimum system error (*minerr*). Set the initial search direction $d^{(0)} = \{0\}$. Set STOP1 = 0 and *iter* = 0. Set the acceptable minimum and maximum step length as *minlen* (= 0.00001) and *maxlen* (= 1000). Set β (= 0.9) and θ (= 0.001).

Step 2. For $k = 1$ to p, perform the following for the kth training instance:

Step 2.1. Perform feedforward procedure of the neural network. Set

$$O_k^{(1)} = \begin{bmatrix} X_k \\ 1 \end{bmatrix}.$$

For $i = 1$ to $N - 1$, calculate the output vector in $(i + 1)$st layer

$$O_k^{(i+1)} = \begin{bmatrix} G\left(W^{(i)} \cdot O_k^{(i)}\right) \\ 1 \end{bmatrix} \tag{13}$$

$$G\left(W^{(i)} \cdot O_k^{(i)}\right) = \begin{bmatrix} \dfrac{1}{1 + e^{-\sum_{j=1}^{n_i+1} (w_{1,j}^{(i)} o_{kj}^{(i)})}} \\ \dfrac{1}{1 + e^{-\sum_{j=1}^{n_i+1} (w_{2,j}^{(i)} o_{kj}^{(i)})}} \\ \vdots \\ \dfrac{1}{1 + e^{-\sum_{j=1}^{n_i+1} (w_{n_{i+1},j}^{(i)} o_{kj}^{(i)})}} \end{bmatrix}. \tag{14}$$

Step 2.2. Calculate the system error for the kth training instance

$$E_k(X_k, W) = \sum_{m=1}^{n_N} (y_{km} - o_{km})^2. \tag{15}$$

Step 2.3. Calculate the deltas in the output layer for the kth training instance

$$\delta_{kr}^{(N)} = \left(y_{kr} - o_{kr}^{(N)}\right)\left(1 - o_{kr}^{(N)}\right)o_{kr}^{(N)}, \qquad r = 1, 2, \ldots, n_N. \tag{16}$$

Step 2.4. For $r = N - 1$ down to 1, calculate the deltas in the hidden layers:

$$\delta_{kq}^{(r)} = o_{kq}^{(r)}\left(1 - o_{kq}^{(r)}\right) \sum_{t=1}^{n_{r+1}} \left(\delta_{kt}^{(r+1)} w_{q,t}^{(r)}\right), \qquad q = 1, 2, \ldots, n_r. \tag{17}$$

Step 2.5. For $i = 1$ to $N - 1$, calculate the gradient vector for the kth training instance

$$\nabla E_k(W^{(n)}) = \frac{\partial E(W)}{\partial w_{q,r}^{(i)}} = \delta_{kq}^{(i+1)} o_{kr}^{(i)},$$

$$q = 1, 2, \ldots, n_{(i+1)} \text{ and } r = 1, 2, \ldots, n_i + 1. \quad (18)$$

Step 3. Calculate the total system error:

$$E(X_k, W) = \frac{1}{2p} \sum_{k=1}^{p} E_k(X_k, W). \quad (19)$$

If $E(X_k, W) < minerr$, set STOP1 = 1 and go to Step 19. Otherwise, go to the next step.

Step 4. Calculate the gradient vector of the total neural network system error:

$$\nabla E(W^{(n)}) = \sum_{k=1}^{p} \nabla E_k(W^{(n)}). \quad (20)$$

Assign the search direction as

$$d^{(n)} = -\nabla E(W^{(n)}). \quad (21)$$

If $|\nabla E(W^{(n)})| < \varepsilon$, set STOP1 = 1 and go to Step 19. In this case, $W^{(n)}$ is the optimum solution ($|X|$ is the length of vector X). Otherwise, continue.

Step 5. Set $iter = iter + 1$. If $iter \geqslant L$, set $iter = 0$. If $iter = 1$, set $\alpha_n = 0$ and go to the next step. Otherwise, calculate the new conjugate direction as

$$d^{(n)} = -\nabla E(W^{(n)}) + \alpha_n d^{(n-1)}, \quad (22)$$

where

$$\alpha_n = \max\left\{0, \frac{\nabla E(W^{(n)})^T v^{(n-1)}}{|\nabla E(W^{(n-1)})|^2}\right\} \tag{23}$$

and

$$v^{(n-1)} = \nabla E(W^{(n)}) - \nabla E(W^{(n-1)}). \tag{24}$$

Step 6. Perform the inexact line search algorithm to calculate λ.
Set the stopping criterion STOP2 ($= 0$). Initialize $\lambda = 1$.

Step 7. Calculate $E(W^{(n)} + \lambda d^{(n)})$ and $E(W^{(n)}) + \beta\lambda(\nabla E(W^{(n)})^T d^{(n)})$.
If $E(W^{(n)} + \lambda d^{(n)}) \leqslant E(W^{(n)}) + \beta\lambda(\nabla E(W^{(n)})^T d^{(n)})$, go to the next step. Otherwise, go to Step 15.

Step 8. Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)}$ and $\theta(\nabla E(W^{(n)})^T d^{(n)})$.
If $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)} < \theta(\nabla E(W^{(n)})^T d^{(n)})$, go to the next step. Otherwise, go to Step 13.

Step 9. If $\lambda = 1$, go to Step 10. Otherwise, go to step 11.

Step 10. Set $\lambda = \min(2\lambda, \mathit{maxlen})$.
Calculate the new search direction $d^{(n+)}$.
Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}$.
If $(\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}) < 0$, calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)}$ and $\theta(\nabla E(W^{(n)})^T d^{(n)})$.
If $(\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}) \geqslant 0$,
or $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)} \geqslant \theta(\nabla E(W^{(n)})^T d^{(n)})$,
or $\lambda \geqslant \mathit{maxlen}$, go to step 11. Otherwise, go to Step 10.

Step 11. If $\lambda < 1$ or ($\lambda > 1$ and $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)} \geqslant 0$), go to Step 12. Otherwise, go to Step 17.

Step 12. Perform backtracking using parabolic interpolation to find a new λ.
Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)}$ and $\theta(\nabla E(W^{(n)})^T d^{(n)})$.
Calculate the new search direction $d^{(n+)}$.
Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}$.
If both conditions, $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n)} \geqslant \theta(\nabla E(W^{(n)})^T d^{(n)})$ and $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)} < 0$, hold simultaneously, set STOP2 = 1 and go to Step 17. Otherwise, go to the beginning of this step.

Step 13. Calculate the new search direction $d^{(n+)}$.
Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}$.
If $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)} \geqslant 0$, go to Step 14. Otherwise, set STOP2 = 1 and go to step 17.

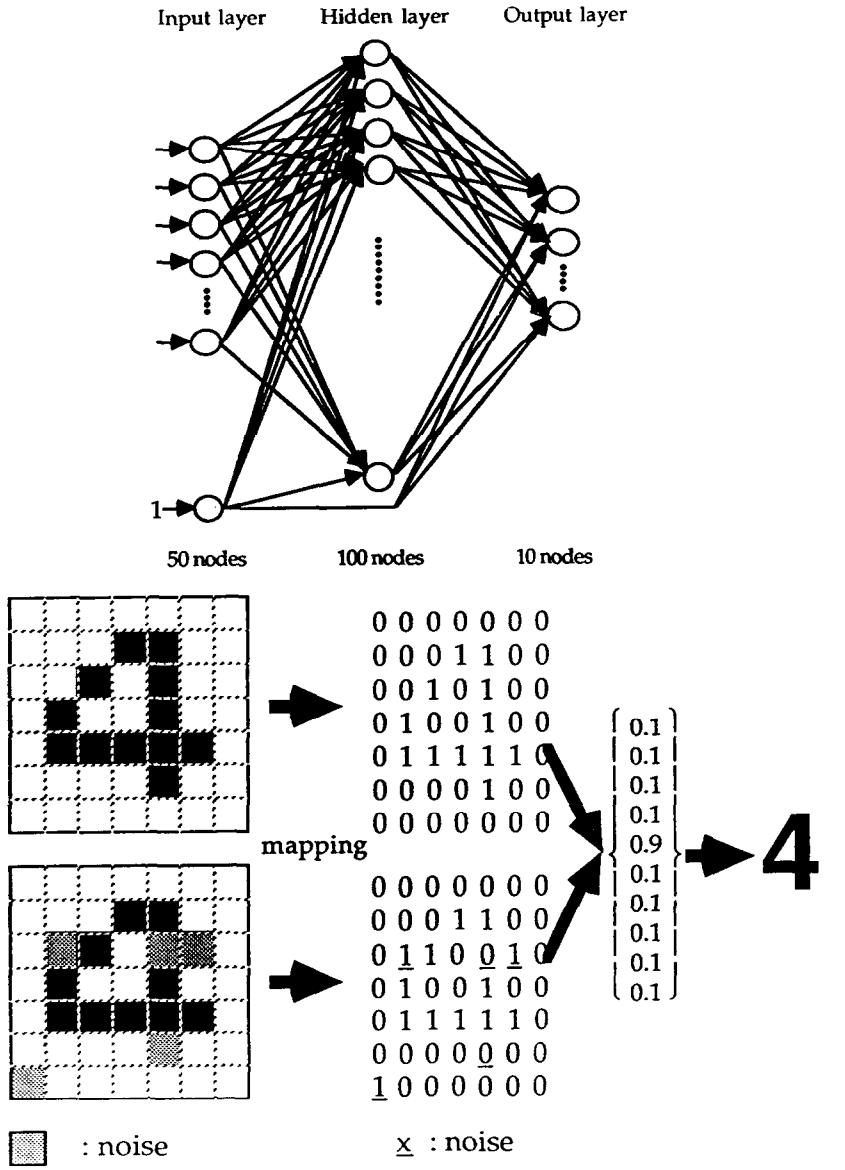


FIG. 4. Three-layer neural network for the 7 × 7 binary image (numerals 0–9) problem.

Step 14. Perform backtracking using parabolic interpolation to find a new λ. Calculate the new search direction $d^{(n+)}$.
Calculate $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)}$.
If $\nabla E(W^{(n)} + \lambda d^{(n)})^T d^{(n+)} < 0$, set STOP2 = 1 and go to Step 17. Otherwise go to the beginning of this step.

Step 15. If $\lambda < minlen$, then set $\lambda = 0$, set STOP2 = 1 and go to Step 17. Otherwise, go to the next step.

Step 16. If $\lambda = 1$, perform backtracking using parabolic interpolation to find a new λ. Otherwise, perform backtracking using cubic interpolation to find a new λ. Go to the next step.

Step 17. If STOP2 = 1, set $\lambda_n = \lambda$, stop the iterations of inexact line search algorithm, and go to the next step. Otherwise, go to Step 6.

Step 18. Update the weight vector as

$$W^{(n+1)} = W^{(n)} + \lambda_n d^{(n)}. \tag{25}$$

Set $n = n + 1$. If $n > MaxIter$, set STOP1 = 1 and go to next step.

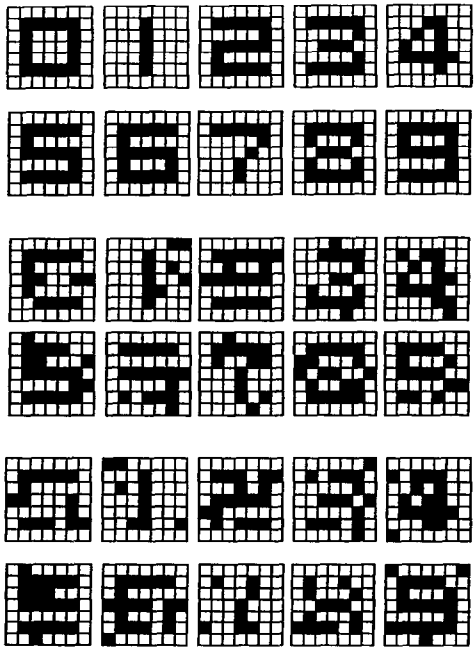


FIG. 5. 30 training instances for the 7 × 7 binary image (numerals 0–9) problem.

Step 19. If STOP1 = 1, stop the iteration of the adaptive conjugate gradient learning algorithm and $W^{(n)}$ is the optimum weight vector. Otherwise, go to Step 2.

The algorithm is restarted every L iterations by setting $\alpha_k = 0$.

6. APPLICATIONS

We apply the adaptive conjugate gradient neural network learning algorithm presented in this paper to two different domains: engineering design and image recognition. Two examples are presented, one in the domain of engineering design and one in the domain of image recognition.

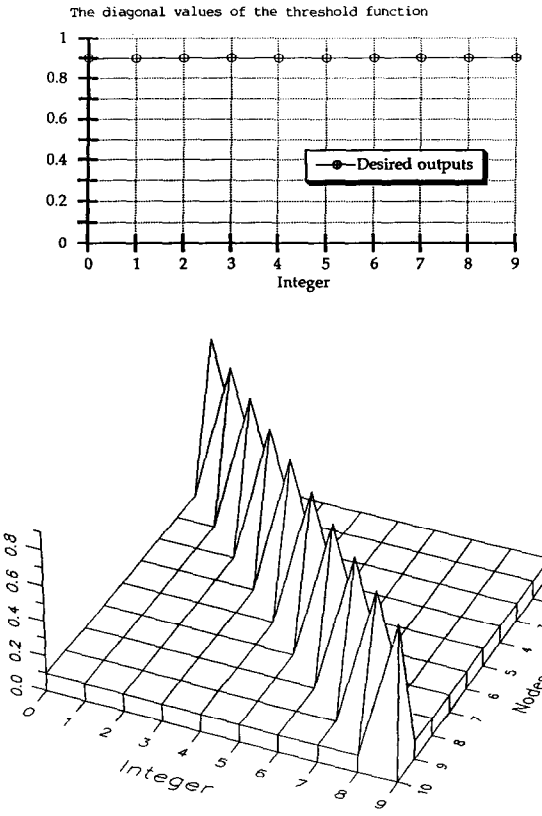


FIG. 6. Three-dimensional contour surface of the desired outputs for the 7×7 binary image problem.

6.1. *Example 1: Engineering Design*

This example is the selection of a minimum weight steel beam from American Institute of Steel Construction (AISC) Load and Resistance Factor Design (LRFD) wide-flange (W) shape database [2] for a given loading condition. This example has been solved by Adeli and Yeh [1] and Hung and Adeli [9]. For details of this example, refer to these references. Each instance consists of five input patterns: the member length (L), the unbraced length (L_b), the maximum bending moment in the member (M_{max}), the maximum shear force (V_{max}), and the bending coefficient (C_b). The output pattern is the plastic modulus (Z_x) of the corresponding least weight member.

A four-layer feedforward neural network with two hidden layers was used to learn this problem. The numbers of nodes in the input layer, the first and second hidden layers, and the output layer are 5, 5, 3, and 1, respectively. There are 52 links in this neural network, resulting in 52 decision variables in the corresponding mathematical optimization problem. For the momentum BP learning algorithm, the learning and momentum ratios were chosen as 0.2

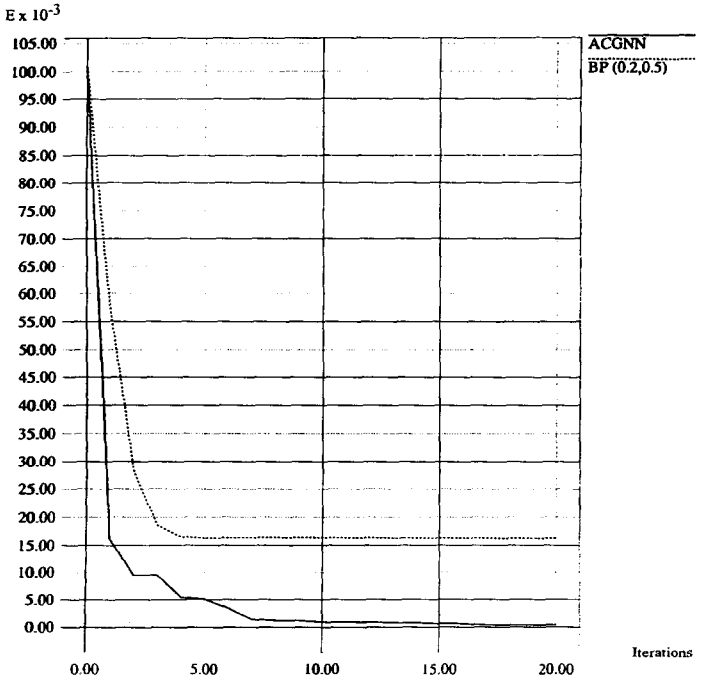


FIG. 7. System error for the minimum weight steel beam problem.

and 0.5, respectively. We used ten training instances in this example. The total number of iterations for the learning process is limited to 20.

6.2. Example 2: Image Processing for 7 × 7 Binary Images

This example is recognition of seven by seven (7 × 7) binary images of the numerals 0 to 9. A three-layer neural network (with one hidden layer) was used to learn this problem as in Figure 4. The numbers of nodes in the input layer, the hidden layers, and the output layer are 49, 99, and 10, respectively. The total number of links in this two-layer neural networks is 5950. Thus, there are 5950 decision variables in the corresponding mathematical optimization problem. This is a very large-scale optimization problem. For the momentum BP learning algorithm, the learning and momentum ratios are chosen as 0.2 and 0.3, respectively. The total number of iterations for the learning process is limited to 200. We used 30 training instances in this example: 10 noiseless image instances and 20 image instances with about 10% noise (5 pixels out of 49 pixels) as in Figure 5.

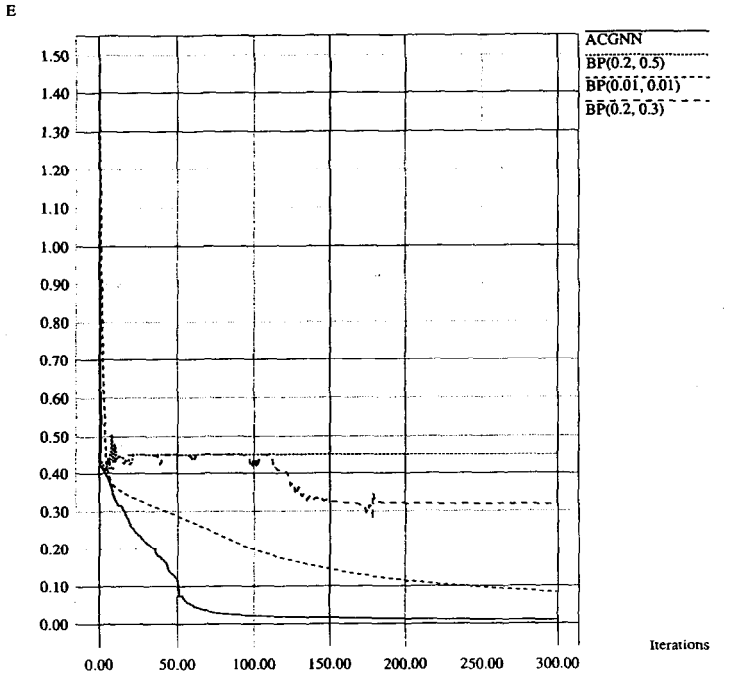


FIG. 8. System error for the 7 × 7 binary image recognition problem.

The value of the threshold function (5) can vary between 0 (FALSE for the binary output) and 1 (TRUE for the binary output), but can never equal these two values. In implementation, we set $g = 0.1$ and $g = 0.9$ as FALSE and TRUE values, respectively. The desired output in this training example is one of the ten integers 0–9. For the simplicity of encoding, we use ten output nodes to represent the ten integers. Thus, the desired output vector, say for the integer 4, is {0.1, 0.1, 0.1, 0.1, 0.9, 0.1, 0.1, 0.1, 0.1, 0.1} (Figure 4).

The outputs can be presented visually in a 3-dimensional contour surface shown in Figure 6. The two horizontal axes represents the ten nodes and ten possible integer outputs (0–9). The vertical axis represents the value of the threshold function. Figure 6 represents the desired output for this training example.

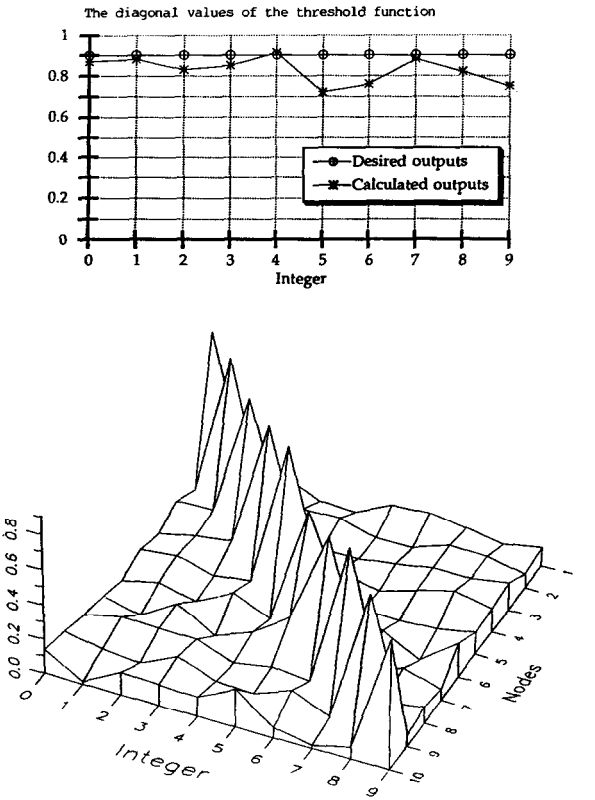


FIG. 9. Three-dimensional contour surface of the calculated outputs for the 10 noiseless training instances shown in Figure 5.

7. LEARNING RESULTS

7.1. *Example 1*

The system error for this example using the momentum BP algorithm and the adaptive conjugate gradient neural network (ACGNN) learning algorithm are shown in Figure 7. The learning performance of the adaptive conjugate gradient neural network learning algorithm is substantially better than that of the momentum BP learning algorithm. After 20 iterations, the adaptive conjugate gradient neural network learning algorithm converges to a very small value of 10^{-5}. But, after the same number of iterations, the momentum BP algorithm (with $\lambda = 0.2$, $\alpha = 0.5$, the best values among different sets we tried) converges to 0.017.

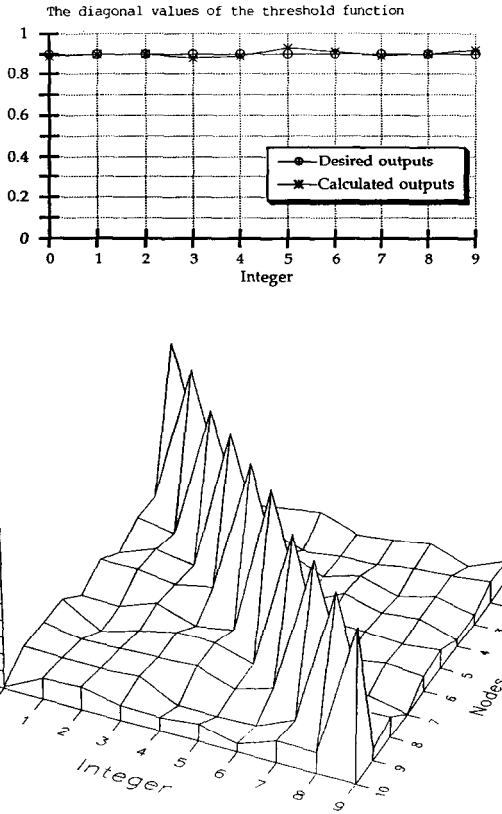


FIG. 10. Three-dimensional contour surface of the calculated outputs for the first 10 noisy training instances shown in Figure 5.

7.2. Example 2

The system error for this example using the momentum BP algorithm with three different pairs of (λ, α) and the adaptive conjugate gradient neural network learning algorithm are shown in Figure 8. The system error for the new learning algorithm is less than 0.1 after 50 iterations and error-free image recognition is achieved. The calculated outputs for the ten noiseless and twenty noisy training instances given in Figure 5 are represented in 3-dimensional contour surfaces in Figures 9–11.

To achieve the same system error with the momentum BP learning algorithm with the best values of the pair (λ, α) requires about 300 iterations. But, using the other values of the pair of (λ, α), the momentum BP learning algorithm can recognize only 45% of the images after 300 iterations, as shown in Figure 12.

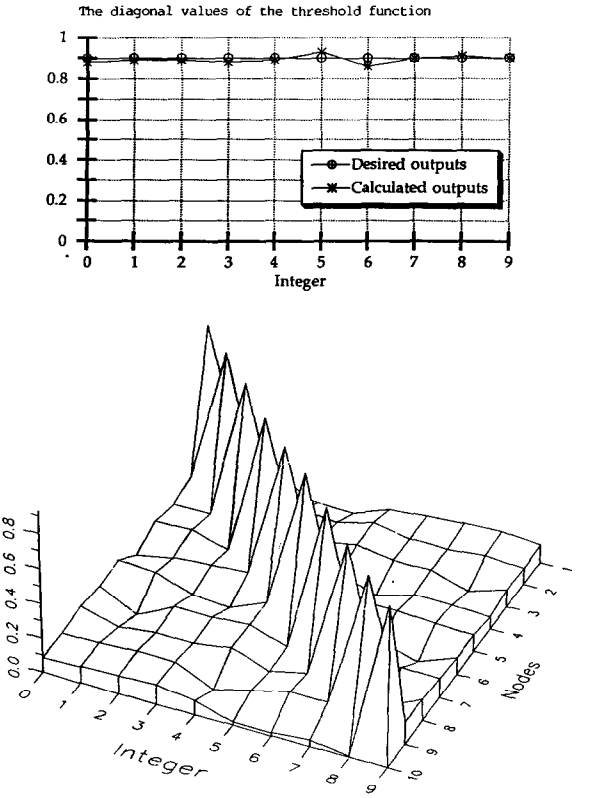
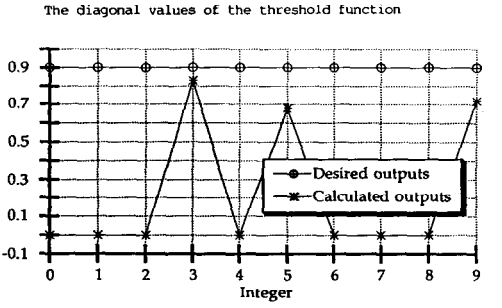


FIG. 11. 3-dimensional contour surface of the calculated outputs for the second ten noisy training instances shown in Figure 5.



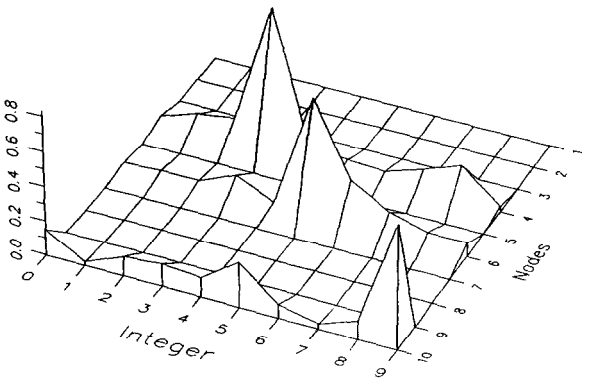


FIG. 12. Three-dimensional contour surface of the calculated outputs for the 10 noiseless training instances (shown in Figure 5) using the momentum BP learning algorithm with ($\lambda = 0.2$, $\alpha = 0.5$).

8. CONCLUDING REMARKS

An adaptive conjugate gradient neural network learning algorithm has been developed and implemented in *C* on a SUN-SPARCstation. The learning algorithm has been applied to two different domains, engineering design and image recognition. The performance of the algorithm has been evaluated by applying it to two examples. The following observations are made and conclusions drawn:

1. The problem of arbitrary trial-and-error selection of the learning ratio (λ) and momentum ratio (α) encountered in the momentum backpropaga-

tion algorithm is circumvented in the new adaptive algorithm. Instead of constant learning and momentum ratios, the step length in the inexact line search is adapted during the learning process through a mathematical approach. Thus, the new adaptive algorithm provides a more solid mathematical foundation for neural network learning.

2. The adaptive conjugate gradient neural network learning algorithm presented in this paper converges much faster than the momentum backpropagation algorithm.

3. The results of neural network learning are sensitive to the initial values of the weight vector. To overcome this problem, we proposed a new hybrid algorithm combining genetic algorithm with neural networks [8]. In order to improve the convergence performance of the hybrid learning algorithm, we are currently integrating the adaptive learning algorithm presented in this paper with a genetic algorithm.

ACKNOWLEDGMENT

This work has been sponsored by the National Science Foundation Grant MSS-9222114, which is gratefully acknowledged.

REFERENCES

1 H. Adeli and C. Yeh, Perceptron learning in engineering design, *Microcomput. Civil Eng.* 4 (4):247–256 (1989).

2 AISC, *Manual of Steel Construction—Load and Resistance Factor Design*, American Institute of Steel Construction, Chicago, IL, 1986.

3 L. Armijo, Minimization of functions having Lipschitz-continuous first partial derivatives, *Pacific J. Math.* 16(1):1–3 (1966).

4 J. E. Dennis, Jr and R. B. Schnable, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, 1983.

5 S. C. Douglas and T. H.-Y Meng, Linearized least-squares training of multilayer feedforward neural networks, in *IEEE International Joint Conference on Neural Networks*, Vol. 1, IEEE Service Center. Piscataway, NJ, 1991, pp. I307–I312.

6 R. Fletcher and R. M. Reeves, Function minimization by conjugate gradients, *Comput. J.* 7(2):149–160 (1964).

7 A. A. Goldstein, *Constructive Real Analysis*, Harper & Row, New York, 1967.

8 S. L. Hung and H. Adeli, A hybrid learning algorithm for distributed memory multicomputers, *Heuristics—The Journal of Knowledge Engineering* 4(4):58–68 (1991).

9 S. L. Hung and H. Adeli, A Model of perception learning with a hidden layer for engineering design, *Neurocomputing* 3(1):3–14 (1991).

10 S. Kollias and D. Anastassiou, An adaptive least squares algorithm for the efficient training of artificial neural networks, *IEEE Trans. Circuits Systems* 36(8):1092–1101 (1989).

11 J. Nocedal, The performance of several algorithms for large scale unconstrained optimization, in *Large-Scale Numerical Optimization* (T. F. Coleman and Y. Li, Eds.), Society for Industrial and Applied Mathematics, Philadelphia, 1990, pp. 138–151.

12 E. Polak and Ribiére, Note sur la convergence de methodes de directions conjugées, *Rev. Francaise Informat. Recherche Opérationelle* 3(16):35–43 (1969).

13 M. J. D. Powell, Convergence properties of algorithms for nonlinear optimization, *SIAM Rev.* 28(4):487–500 (1986).

14 D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representation by error propagation, in *Parallel Distributed Processing* (D. E. Rumelhart et al., Eds.), MIT Press, Cambridge, 1986, pp. 318–362.