Pergamon

Neural Networks, Vol. 10, No. 1, pp. 69–82, 1997

Printed in Great Britain
0893-6080/97 $17.00 + .00

PII: S0893-6080(96)00052-4

CONTRIBUTED ARTICLE

Effective Backpropagation Training with Variable Stepsize

GEORGE D. MAGOULAS,[1] MICHAEL N. VRAHATIS[2] AND GEORGE S. ANDROULAKIS[2]

[1] Department of Electrical and Computer Engineering, and [2] Department of Mathematics, University of Patras

(*Received* 6 *January* 1995; *revised and accepted* 8 *April* 1996)

Abstract—*The issue of variable stepsize in the backpropagation training algorithm has been widely investigated and several techniques employing heuristic factors have been suggested to improve training time and reduce convergence to local minima. In this contribution, backpropagation training is based on a modified steepest descent method which allows variable stepsize. It is computationally efficient and possesses interesting convergence properties utilizing estimates of the Lipschitz constant without any additional computational cost. The algorithm has been implemented and tested on several problems and the results have been very satisfactory. Numerical evidence shows that the method is robust with good average performance on many classes of problems. Copyright © 1996 Elsevier Science Ltd.*

Keywords—Feedforward neural network, Numerical optimization techniques, Steepest descent, Error backpropagation.

1. INTRODUCTION

The efficient supervised training of feedforward neural networks (FNNs) is a subject of considerable ongoing research and numerous algorithms have been proposed to this end. The backpropagation (BP) algorithm (Rumelhart & McClelland, 1986) is one of the most common supervised training methods. Although BP training has proved to be efficient in many applications, it uses a constant stepsize, its convergence tends to be very slow, and it often yields suboptimal solutions (Baldi & Hornik, 1989).

Attempts to speed up training and reduce convergence to local minima have been made in the context of gradient descent by Cater (1987), Chan and Fallside (1987), Jacobs (1988), Vogl et al. (1988), Battiti (1989), Darken and Moody (1990), and Silva and Almeida (1990). However, these methods are based on the use of heuristic factors to dynamically adapt the stepsize.

In this contribution, BP training with variable stepsize (BPVS) is presented. The BPVS method is based on a modification of the deterministic steepest descent that allows variable stepsize as a consequence of minimization of the objective function and of observation of the trajectory in the weight space. Its convergence is guaranteed, utilizing estimates of the Lipschitz constant that are obtained without additional error function and gradient evaluations. The BPVS method is tested and compared with three popular deterministic steepest descent based training methods, on several application examples.

2. THE BACKPROPAGATION TRAINING

Consider a FNN whose lth layer contains N_l neurons, for $l = 1, \ldots, M$. The network is based on the following equations:

$$\text{net}_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}, \tag{1}$$

$$y_j^l = f(\text{net}_j^l), \tag{2}$$

where net_j^l is, for the jth neuron in the lth layer ($j = 1, \ldots, N_l$), the sum of its weighted inputs. The weights from the ith neuron at the $(l-1)$th layer to the jth neuron at the lth layer are denoted by $w_{ij}^{l-1,l}$, y_j^l is the output of the jth neuron that belongs to the lth layer, and $f(\text{net}_j^l) = (1 + \exp(-\text{net}_j^l))^{-1}$ is the jth's neuron activation function.

Acknowledgements: The authors gratefully acknowledge many stimulating and useful discussions with Professor R. E. King and Professor A. Sideris. Thanks are also due to Dr Y. Karayiannis for helping in the texture classification problem.

Requests for reprints should be sent to G. D. Magoulas, Control Systems Laboratory, Department of Electrical and Computer Engineering, University of Patras, GR-261.10 Patras, Greece; Tel: +30.61.997292; Fax: +30.61.997310; e-mail: magoulas@ee-gw.ee.upatras.gr

Training is realised by minimizing the error function E defined by:

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{N_M} (y_{j,p}^{M} - t_{j,p})^2 = \sum_{p=1}^{P} E_p, \qquad (3)$$

where $(y_{j,p}^{M} - t_{j,p})^2$ is the squared difference between the actual output value at the jth output layer neuron for pattern p and the target output value; p is an index over input–output pairs. This function also provides an error surface over the weight space.

Each pass through the entire training set which contains P representative pairs, to compute E is called an "*epoch*". The minimization of E corresponds to updating the weights by epoch (batch learning). This approach is consistent with the numerical optimization theory since it uses information from all the pairs in the training set, i.e., the true gradient. Moreover, batch learning makes the problem of minimization more interesting since the summation over the entire training set in eqn (3) usually results in a complex error surface (Weier, 1991), increasing the possibility for any steepest descent based training algorithm to be trapped in a local minimum.

In the BP algorithm, minimization of E is attempted using the steepest descent with constant stepsize λ and the computation of the gradient of the sum-of-squared-differences error (SSE) function, ∇E, by applying the chain rule on the layers of the FNN (Rumelhart & McClelland, 1986).

The BP algorithm can be summarized in the following equations:

$$w_{ij}^{l-1,l}|_{k+1} = w_{ij}^{l-1,l}|_{k} - \lambda \delta_j^l y_i^{l-1}, \qquad (4)$$

$$\delta_j^L = (y_{j,p}^{L} - t_{j,p}) f'(\text{net}_j^L), \qquad (5)$$

$$\delta_j^l = f'(\text{net}_j^l) \sum_{n=1} w_{jn}^{l,l+1}|_k \delta_n^{l+1}, \quad \text{for } l = M-1, \ldots, 2, \qquad (6)$$

where eqn (4) is the weight update equation for any set of input–output patterns, in which λ is the constant stepsize, and the notation $|_k$ indicates the kth epoch. Equations (5) and (6) specify the backpropagating error signal for the output layer and the hidden layers respectively, and $f'(\cdot)$ is the derivative of the activation function.

The BP method approximates a local minimum of E and always converges when λ is chosen to satisfy the relation $\sup\|Q(w)\| \leq \lambda^{-1} < \infty$ (Goldstein, 1962) in some bounded region where the relation $E(w) \leq E(w^0)$ holds; $Q(w)$ denotes the Hessian matrix of E with respect to the weight vector w and w_0 denotes the initial weight vector. The behavior of E in the neighborhood of a local minimum is determined by the eigensystem of the matrix Q. In the neural network implementation, the storage and computational requirements of the approximated Hessian for FNNs with several hundred weights make its use impractical (Bello, 1992; Magoulas et al., 1995). Thus, the stepsize is usually chosen according to the relation $0 < \lambda < 1$ in such a way that successive steps in weight space do not overshoot the minimum of the error surface.

The usual approach in BP training methods that permit variable stepsize focuses on two issues: (i) start with a small stepsize and increase it exponentially if successive epochs reduce the error, and rapidly decrease it if a significant error increase occurs (Vogl et al., 1988; Battiti, 1989), (ii) start with a small stepsize and increase it if successive epochs keep the gradient direction fairly constant or rapidly decrease it if the direction of the gradient varies greatly at each epoch (Chan & Fallside, 1987). For each weight an individual stepsize can also be given (Jacobs, 1988; Silva & Almeida, 1990), which increases if the successive changes in the weights are in the same direction and decreases otherwise. In this case, there is no guarantee that the adaptations will be in the proper direction determined by the negative gradient of E.

A different approach are the so-called search-then-converge schedules that combine the desirable features of the standard least-mean-square and traditional stochastic approximation algorithms (Darken & Moody, 1990, 1991; Darken et al., 1992).

3. THE BACKPROPAGATION TRAINING WITH VARIABLE STEPSIZE

3.1. Description of the BPVS Training Method

The minimization of the error function E requires a sequence of weight vectors $\{w^k\}_0^\infty$ (where k indicates epochs) which converges to the point w^* that minimizes E. First, let us make the following assumptions regarding the error function E:

(i) The function E is a real-valued function defined and continuous everywhere in $\mathscr{R}^n$ (real Euclidean n-space), bounded from below in $\mathscr{R}^n$;

(ii) for a given point $w^0 \in \mathscr{R}^n$, and for every w in some region containing the initial weight vector w^0, $S(w^0) = \{w : E(w) \leq E(w^0)\}$, then $E \in C^1$ on $S(w^0)$ and ∇E is Lipschitz continuous on $S(w^0)$, i.e., there exists a Lipschitz constant $K > 0$, such that:

$$\|\nabla E(v) - \nabla E(w)\| \leq K\|v - w\|, \qquad (7)$$

for every pair v, $w \in S(w^0)$;

(iii) $r>0$ implies that $m(r)>0$, where $m(r)=\inf_{w\in S_r(w^0)}\|\nabla E\|$, $S_r(w^0)=S_r\cap S(w^0)$, $S_r=\{w:\|w-w^*\|\geq r\}$, and w^* is any point for which $E(w^*)=\inf_{w\in\mathcal{R}^n}E(w)$ (in case $S_r(w^0)$ is void $m(r)$ is defined to be equal to infinity).

By assuming that the above conditions are fulfilled the following convergence theorem applies:

THEOREM 1 (Armijo, 1966). *If $0<\rho\leq 0.25K^{-1}$, then for any $w\in S(w^0)$, the set $S^*(w,\rho)=\{w_\lambda : w_\lambda = w-\lambda\nabla E(w),\ \lambda>0,\ E(w_\lambda)-E(w)\leq-\rho\|\nabla E(w)\|^2\}$, is a non-empty set of $S(w^0)$ and any sequence of weight vectors $\{w^k\}_0^\infty$, such that $w^{k+1}\in S^*(w^k,\rho)$, $k=0,1,2,\ldots$, converges to the point w^* which minimizes E.*

It is known that the optimal value of the stepsize λ depends on the shape of the N-dimensional error function, and can be obtained using the value of the Lipschitz constant K. So the weight update equation (4) can be written as:

$$w^{k+1}=w^k-0.5K^{-1}\nabla E(w^k),\ k=0,1,2,\ldots \tag{8}$$

where

$$\nabla E(w^k)=\sum_{p=1}^{P}\nabla E_p(w^k).$$

The sequence of weight vectors $\{w^k\}_0^\infty$ is such that $w^{k+1}\in S^*(w^k, 0.25K^{-1})$, and convergence to the point w^* is guaranteed as a consequence of Theorem 1.

In neural network implementation, the value of K is not known *a priori*, and a "small" stepsize is chosen. If a long stepsize is used there is no guarantee that the steepest descent will converge. This problem can be avoided by a proper stepsize tuning in each epoch. The following theorem gives an elegant stepsize tuning procedure:

THEOREM 2 (Armijo, 1966). *Suppose that η_0 is an arbitrary assigned positive number and consider the sequence $\eta_m=\eta_0 2^{1-m}$, $m=1,2,\ldots$. Then the sequence of weight vectors $\{w^k\}_0^\infty$ defined by*

$$w^{k+1}=w^k-\eta_{m_k}\nabla E(w^k),\ k=0,1,2,\ldots \tag{9}$$

where m_k is the smallest positive integer for which:

$$E(w^k-\eta_{m_k}\nabla E(w^k))-E(w^k)\leq-\frac{1}{2}\eta_{m_k}\|\nabla E(w^k)\|^2, \tag{10}$$

converges to the point w^* which minimizes the error function E.

Theorem 2 constitutes an efficient and very useful stepsize adaptation procedure. However, according to simulation experiments we performed, its exponential schedule is fast enough for many neural network applications resulting in faster training when compared with the BP with fixed stepsize but in slower training when compared with certain BP methods with variable stepsize like the one proposed by Vogl et al. (1988).

BPVS exploits all the local information regarding the direction and the stepsize. Thus, BPVS follows the well known steepest descent direction, $-\nabla E(w^k)$, and uses a local approximation of the Lipschitz constant L_k in order to estimate the stepsize $0.5K^{-1}$ at each epoch. The local approximation of the Lipschitz constant L_k can be easily obtained without any additional error function and gradient evaluations by relation (7) for a pair of subsequent weight updates w^k, w^{k-1}, i.e.,

$$L_k=\|\nabla E(w^k)-\nabla E(w^{k-1})\|/\|w^k-w^{k-1}\|. \tag{11}$$

In this way, the stepsize $0.5L_k^{-1}$ will be sensitive to the local shape of the error function and the exponential schedule of Theorem 2 is avoided. If, for some reason, the stepsize $0.5L_k^{-1}$ is very long and successive steps in weight space do not satisfy relation (10), the stepsize adaptation procedure of Theorem 2 is used in order that the subsequent weight updates do not overshoot the minimum of the error surface. On the other hand, if the stepsize $0.5L_k^{-1}$ is smaller than a specific lower bound, BPVS increases it. One such stepsize lower bound can be specified by the value of the desired accuracy in obtaining the final weights w^*. Obviously, if the stepsize is smaller than the desired accuracy, it is impossible to reach the solution rapidly, except in situations where relation (10) is not satisfied. A simple adaptation mechanism to increase the stepsize is to double it. Now, since BPVS fulfills relation (10), it converges to a minimizer of E.

In the next subsection, the BPVS training algorithm is summarized.

3.2. The BPVS Training Algorithm

In the BPVS method, the value of E can be calculated with one forward pass and the value of ∇E with one forward and one backward pass. A high-level description of the batch BPVS training algorithm is outlined below:

Step 0: Initialize by setting the number of epochs $k=0$, the weights w^0 to real random values, the stepsize to an arbitrary real value η_0, the termination condition *(TC)*, the stepsize lower bound *(SLB)*, $L_0=1$, $t_k=1$.

Step 1: Present an input–desired output pair p, and do Steps 1.1–1.3 for all $p \in [1, P]$, in order to compute the gradient of E_p for all pairs.

Step 1.1: Use the predetermined weights and compute the output of the FNN.

Step 1.2: Compute the backpropagating error signal using relations (5) and (6).

Step 1.3: Compute $\nabla_{ij}E_p = \delta_{j,p}y_{i,p}$, where $\nabla_{ij}E_p$ denotes the partial derivative of E_p with respect to each weight w_{ij} connecting the pair (i, j) of nodes, y_i is either the output of node i or an input.

Step 2: Take the local approximation L_k of the Lipschitz constant, using relation (11). Compute $\eta_k = 0.5L_k^{-1}$. If $\eta_k > SLB$, go to next step; otherwise set $t_k = t_k + 1$, $\eta_k = \eta_k 2^{t_k - 1}$ and go to next step.

Step 3: If relation (10) holds, set $m_k = 1$, and go to Step 5; otherwise, set $m_k = m_k + 1$, $t_k = 1$, and go to next step.

Step 4: Set $\eta_k = \eta_0 2^{1-m_k}$, and return to Step 3.

Step 5: Update weights according to:

$$w^{k+1} = w^k - \eta_k \nabla E(w^k),$$

where

$$\nabla E(w^k) = \sum_{p=1}^{P} \nabla E_p(w^k).$$

Step 6: Check $E(w^{k+1}) > TC$, replace k by $k + 1$, go to Step 1; otherwise get the final weight vector w^*, and the corresponding value of E.

REMARK 1. The implementation of a stepsize lower bound does not influence the convergence of the BPVS method and has no crucial effect on the convergence speed.

REMARK 2. Step 2 of the BPVS algorithm gives an adaptation of the stepsize increment in the case $\eta_k < SLB$. In this way, unsatisfactory convergence rate as a consequence of an unsuitable local approximation of the Lipschitz constant is avoided.

REMARK 3. Steps 3–4 constitute an efficient method to determine a safe stepsize without additional gradient evaluations. This method guarantees BPVS convergence to a weight vector w^* which minimizes the error function E.

4. APPLICATION EXAMPLES

The BPVS training algorithm has been applied to several problems. The FNNs have been implemented in Matlab version 3.5 (Moler et al., 1987) and a number of simulations have been performed to evaluate the BPVS and compare its performance with the batch versions of BP, momentum BP (MBP) (Jacobs, 1988), and adaptive BP (ABP) (Vogl et al., 1988).

The algorithms have been tested using the same initial weight vectors, and received the same sequence of training patterns. The FNNs were based on nodes with activation function of the form $(1 + e^{-net})^{-1}$.

In all application examples the stepsize lower bound was taken equal to 0.04. For the BPVS initial stepsize a large value was randomly selected for each application, in order to test the stepsize tuning effectiveness and the robustness in oscillations due to large stepsizes. On the other hand, much effort has been made to properly tune the heuristic factors, but there is no guarantee that our final choice is optimal. However, our experience with simulations indicates that the behavior of the algorithms described in the examples to follow is characteristic.

A consideration that is worth mentioning is the difference between epochs (weight vector updates) and error function evaluations: for the BP, the momentum BP and the adaptive BP training algorithms the number of error function evaluations equals the number of epochs; for the BPVS there is a number of additional error function evaluations due to Steps 3–4 [i.e., when relation (10) does not hold]. Thus, a comparison in terms of error function evaluations is preferable in order to readily obtain the computational efficiency of the BPVS.

4.1. The Case of a Single Neuron

This simple example (Demuth & Beale, 1992) is chosen because it is easy to visualize and illustrate the behaviour of the algorithms in different cases. The problem consists of a neuron with two weights to be learned, on a set of eight, two-dimensional patterns. The two weights, w_1, w_2 are initialized from uniform distributions in the intervals $(-1, 1)$ and $(-2.5, 2.5)$, respectively. The error surface is shown in Figure 1.

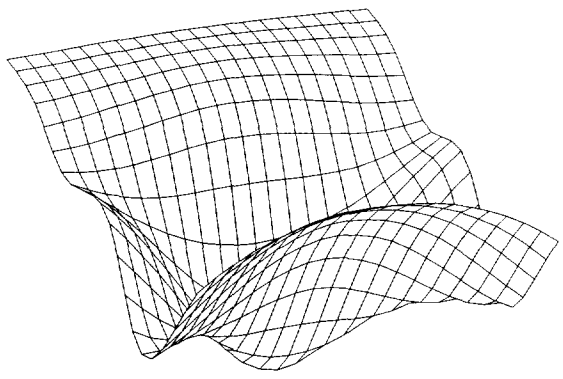

FIGURE 1. Error surface for example 4.1.

The global minimum is located at the center and there are two valleys that lead to local minima.

The behavior of the algorithms is tested in three characteristic cases: (i) when the initial weight vector leads to the global minimum, (ii) when the initial weight vector leads BP to a local minimum, and (iii) when the initial weight vector leads to the global minimum, in the presence of additive weight perturbations in each epoch. Additive weight perturbations may be caused by the imprecision of digital or analog hardware in the implementation of the FNNs and are modeled as a random noise with uniform distribution (Holt & Hwang, 1993). This is especially true when the FNN is to be implemented in real time (Frye et al., 1991) or using fixed-point arithmetic (Sakaue et al., 1993). In the simulations, all weights have been perturbed by a random number uniformly distributed in $(-0.3, +0.3)$. The perturbed weights have been used in all calculations.

In all cases, the stepsize for the adaptive BP and the BPVS is set equal to 0.6; the stepsizer for the BP, and the momentum BP is taken equal to 0.05 (a larger value leads to oscillations). After tuning the heuristics are set as follows: momentum factor = 0.95, error ratio = 1.04, stepsize increment factor = 1.05, and stepsize decrement factor = 0.4.

Figures 2–5 illustrate, in contours of constant error in the w_1, w_2-plane, the behavior of the algorithms when initial weights lead to the global minimum. Under the same conditions, BPVS need 18 error function evaluations, to reach the global minimum, while BP, momentum BP, and adaptive BP need 40, 48, and 26 error function evaluations, respectively. BPVS stepsize η vs error function evaluations is shown in Figure 6a, and the stepsize $0.5\,L_k^{-1}$ due to the local approximation of the Lipschitz constant, is shown in Figure 6b. By comparing Figures 6a and 6b one can see that the stepsize is $\eta_k = 0.5\,L_k^{-1}$ for the cases: 1, 4, 6–11, 15, 17–18; in the situations 5 and 16 the value $0.5\,L_k^{-1}$ is less than the stepsize lower bound, thus $\eta_k = L_k^{-1}$. As a consequence of Step 3 of the BPVS, the stepsize is reduced according to Step 4 in the following situations: 2–3, and 12–14.

When initial conditions lead BP to a local minimum (see Figure 7) momentum BP, adaptive

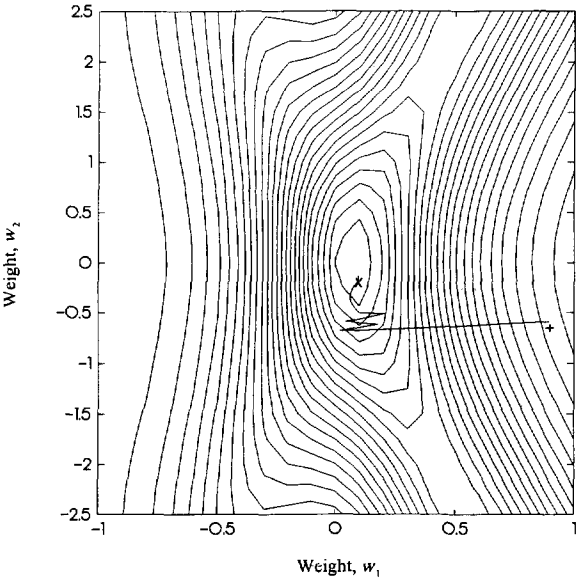

FIGURE 3. Illustration of the momentum backpropagation training algorithm in the case of the global minimum. The ellipses are contours of the surface depicted in Figure 1.

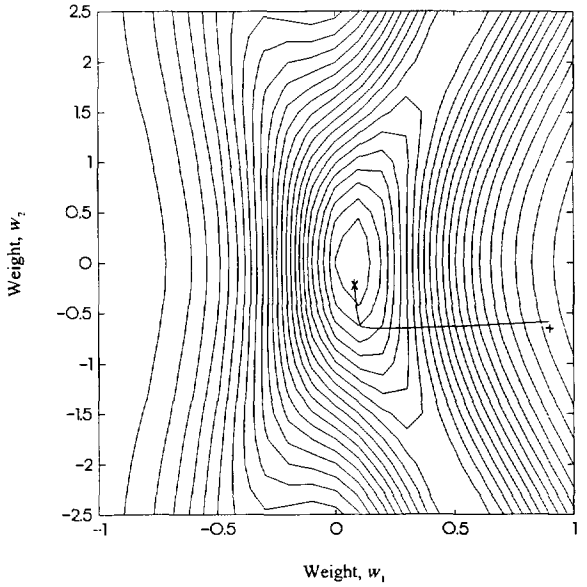

FIGURE 2. Illustration of the backpropagation training algorithm in the case of the global minimum. The ellipses are contours of the surface depicted in Figure 1.

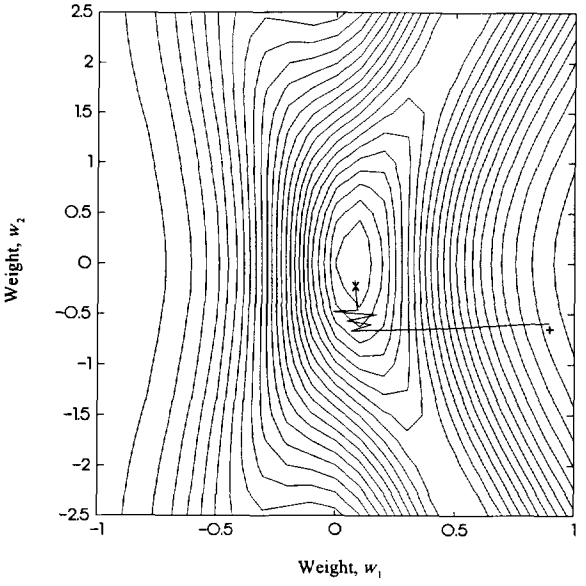

FIGURE 4. Illustration of the adaptive backpropagation training algorithm in the case of the global minimum. The ellipses are contours of the surface depicted in Figure 1.

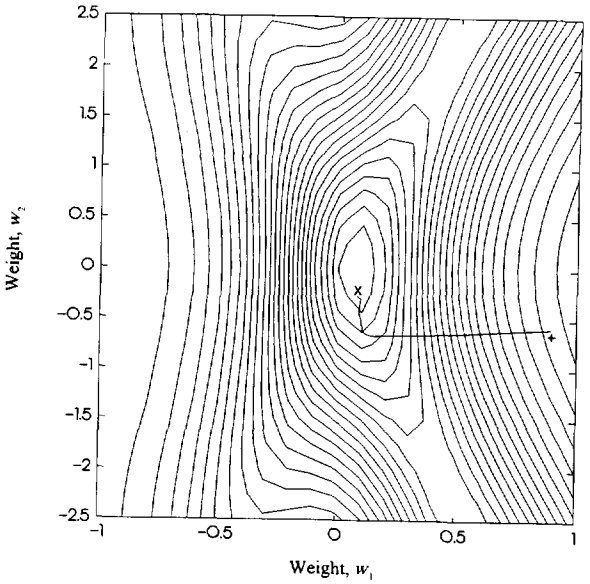

FIGURE 5. Illustration of the backpropagation with variable stepsize in the case of the global minimum. The ellipses are contours of the surface depicted in Figure 1.

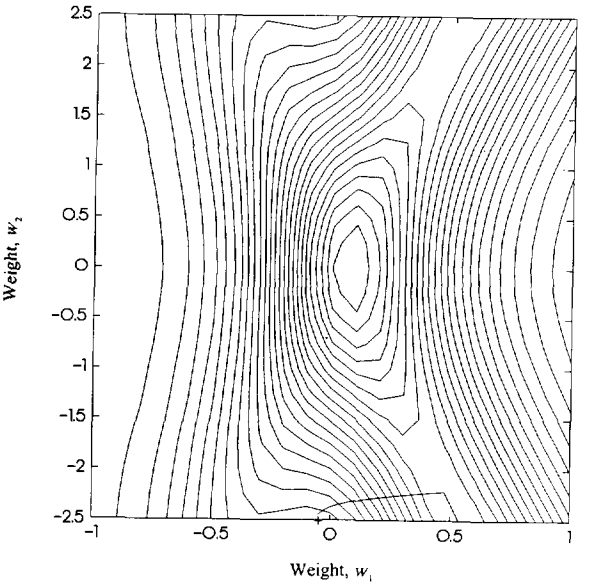

FIGURE 7. Illustration of the backpropagation training algorithm in the case of a local minimum. The ellipses are contours of the surface depicted in Figure 1.

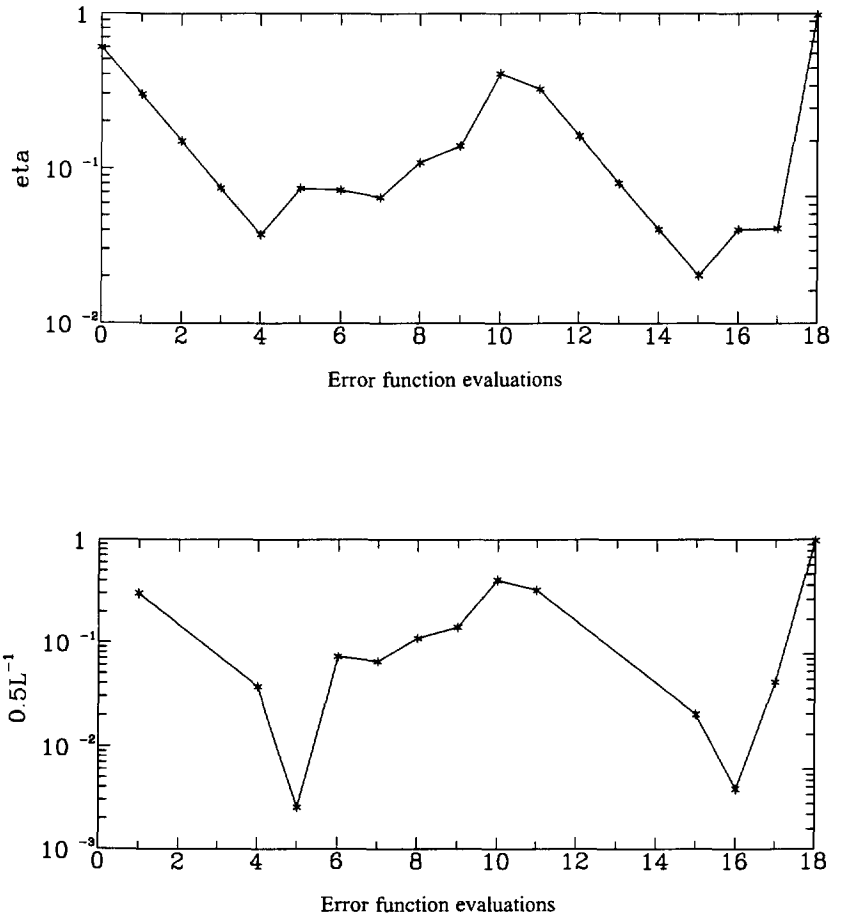

FIGURE 6. The case of a single neuron: (a) stepsize η vs error function evaluations curve for the backpropagation with variable stepsize; (b) the computed stepsize $0.5L_k^{-1}$ vs error function evaluations curve. The curves are for the global minimum case.

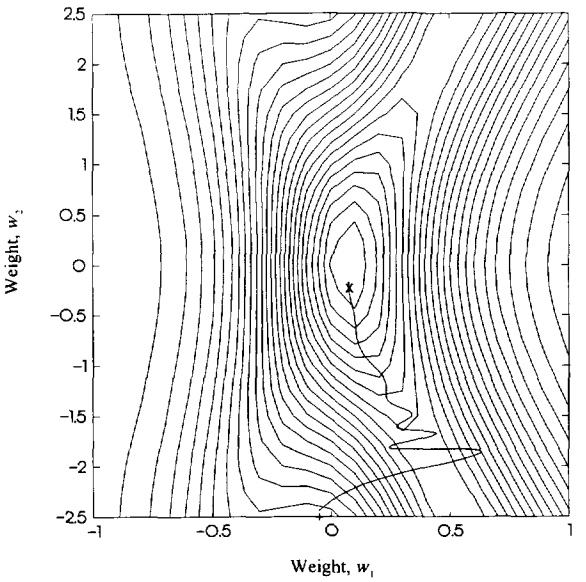

FIGURE 8. Illustration of the momentum backpropagation training algorithm in the case of a local minimum. The ellipses are contours of the surface depicted in Figure 1.

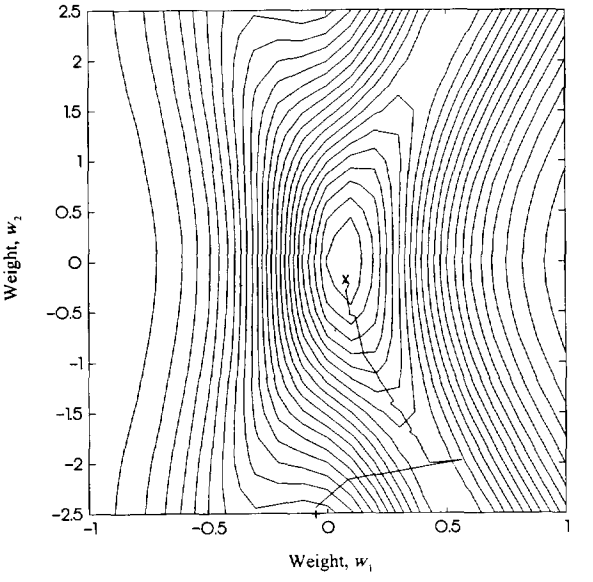

FIGURE 10. Illustration of the backpropagation with variable stepsize in the case of a local minimum. The ellipses are contours of the surface depicted in Figure 1.

BP, and BPVS converge to the global minimum. Figures 8–10 illustrate, in contours of constant error in the w_1, w_2-plane, the behavior of momentum BP, adaptive BP, and BPVS. Although contours are oval-shaped and bent, BPVS has escaped from the local minimum updating the weights using the stepsize $0.5\,L_k^{-1}$. Thus, BPVS has exploited the error surface related information provided by the local approximation of the Lipschitz constant, and converged to the global minimum faster than the momentum BP and the adaptive BP. To be more specific, BPVS needs 76 error function evaluations, while momentum BP and adaptive BP need 177 and 102 error function evaluations, respectively.

The effects on the behavior of the algorithms when perturbing the weights are illustrated in Figures 11–14. As shown in Figure 11, BP oscillates but finally succeeds to converge to the global minimum. The momentum factor (see Figure 12) appears to help limiting the influence of noise. As illustrated in Figure 13, the adaptive BP oscillates when reaching the neighborhood of the global minimum. On the other hand, BPVS trajectory is smooth enough. In general, BPVS had a behavior similar to the adaptive BP but

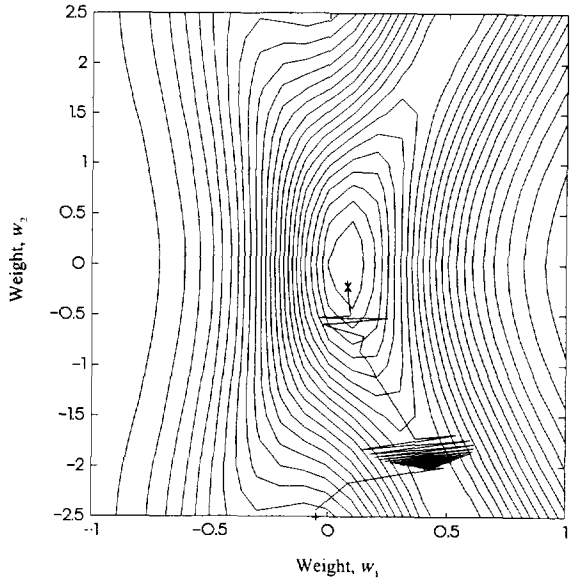

FIGURE 9. Illustration of the adaptive backpropagation training algorithm in the case of a local minimum. The ellipses are contours of the surface depicted in Figure 1.

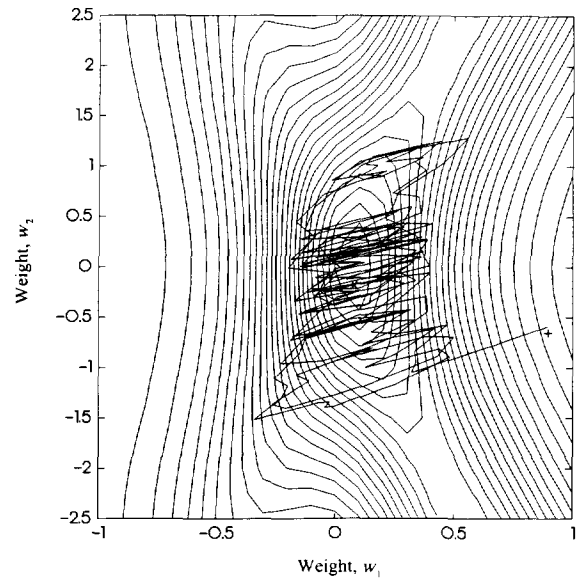

FIGURE 11. Illustration of the backpropagation training algorithm with additive weight perturbations. The ellipses are contours of the surface depicted in Figure 1.

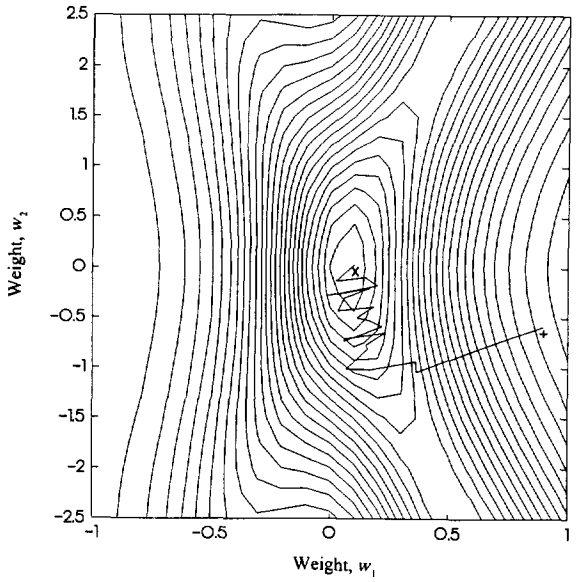

FIGURE 12. Illustration of the momentum backpropagation training algorithm with additive weight perturbations. The ellipses are contours of the surface depicted in Figure 1.

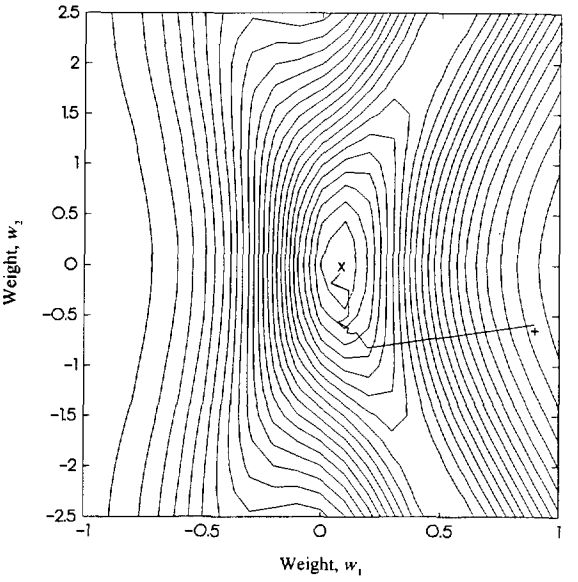

FIGURE 14. Illustration of the backpropagation with variable stepsize with additive weight perturbations. The ellipses are contours of the surface depicted in Figure 1.

when reaching the global minimum the stepsize is kept very small (due to Steps 3–4), and the algorithm does not oscillate (see Figure 14).

4.2. XOR classification problem

Classification of the four XOR patterns into {0, 1} using a 2-2-1 FNN (six weights, three biases) is a classical test problem (Jacobs, 1988; Kollias & Anastassiou, 1989; van Ooyen & Nienhuis, 1992; van der Smagt, 1994). The XOR problem is sensitive to initial weights as well as to stepsize variations and

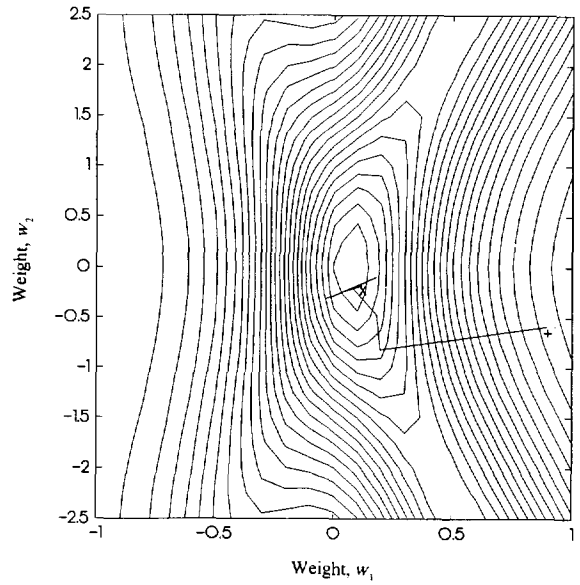

FIGURE 13. Illustration of the adaptive backpropagation training algorithm with additive weight perturbations. The ellipses are contours of the surface depicted in Figure 1.

presents a multitude of local minima with certain weight vectors (Blum, 1989). The weights were initialized using the Nguyen–Widrow (1990) method. BPVS stepsize was taken equal to 12. For the BP, the momentum BP, and the adaptive BP the following standard values were chosen: stepsize = 0.75, momentum factor = 0.9, error ratio = 1.04, stepsize increment factor = 1.05, and stepsize decrement factor = 0.7. The termination condition was $E \leq 0.04$ within 600 error function evaluations.

In all instances, 1000 simulations were run and the results are summarized in Table 1. BPVS and adaptive BP had similar performance: they were faster than the BP and momentum BP, but BPVS had smaller standard deviation than adaptive BP. The aforementioned local minima problem affects the number of successful simulations. However, an improvement to +10% in the success was observed

TABLE 1
Results of Simulations for the XOR Problem

Algorithm	m.n.	s.d.	m.e.	s.e.	suc.
BP	250.50	96.82	39.8	0.11	441
MBP	240.90	113.02	39.6	0.24	434
ABP	78.56	69.88	38.1	1.96	484
BPVS	78.41	55.19	37.8	0.17	485

m.n.=mean number of error function evaluations for simulations that reached solution; s.d.=standard deviation of error function evaluations for simulations that reached solution; m.e.=mean value of error for simulations that reached solution $\times 10^{-3}$; s.e.= standard deviation of error for simulations that reached solution $\times 10^{-3}$; suc.=simulations succeeded out of 1000 within the error function evaluations limit.

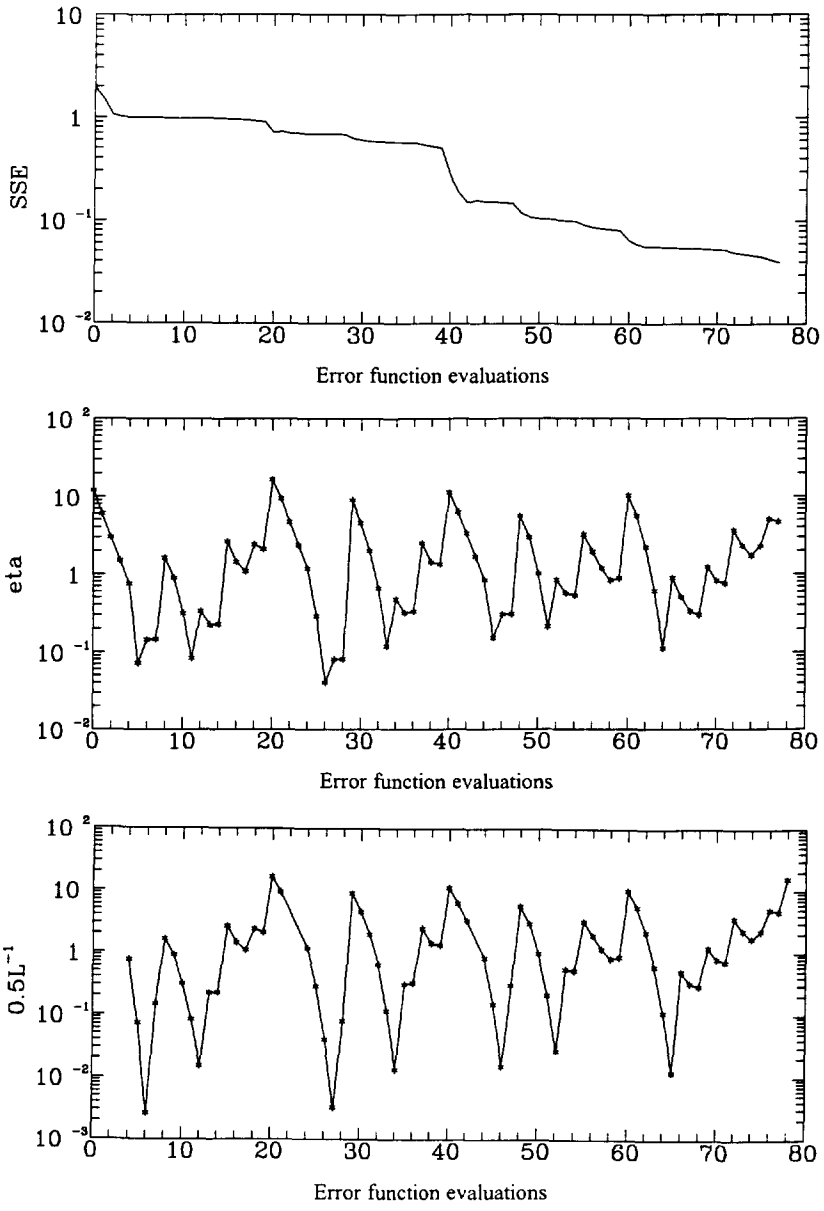

FIGURE 15. XOR problem: (a) BPVS convergence behavior; (b) stepsize η behavior; (c) stepsize $0.5L_k^{-1}$ behavior.

for the BPVS when compared with the BP. The BPVS error function E in a typical run, is shown in Figure 15a. The corresponding stepsize η vs error function evaluations curve is illustrated in Figure 15b, and the stepsize $0.5L_k^{-1}$ due to the local approximation of the Lipschitz constant, is shown in Figure 15c.

4.3. Numeric Font Learning Problem

A 64-6-10 FNN (444 weights, 16 biases) is used as a simple numeral recognizer (Sperduti & Starita, 1993) with an 8 × 8 pixel input and a 10 bit one-hot output representing zero through nine. The weights were initialized using the Nguyen–Widrow (1990) method. The stepsize was set to 1.5 for the BP, the momentum BP, and the adaptive BP, in agreement with results reported in Sperduti & Starita (1993). In the case of BPVS the stepsize was equal to 24. The termination condition was $E \leq 0.04$ within 2000 error function evaluations and the algorithms were tested on 1000 simulation runs. The heuristics were set as follows: momentum factor = 0.9, error ratio = 1.04, stepsize increment factor = 1.05, stepsize decrement factor = 0.7.

TABLE 2
Results of Simulations for the 8 × 8 Numeric Font Problem

Algorithm	m.n.	s.d.	m.e.	s.e.	suc.
BP	859.2	420.67	39.9	0.17	242
MBP	1824.0	0.00	39.9	0.00	2
ABP	814.4	466.94	39.2	1.50	544
BPVS	337.3	84.97	38.7	1.50	999

Abbreviations as for Table 1.

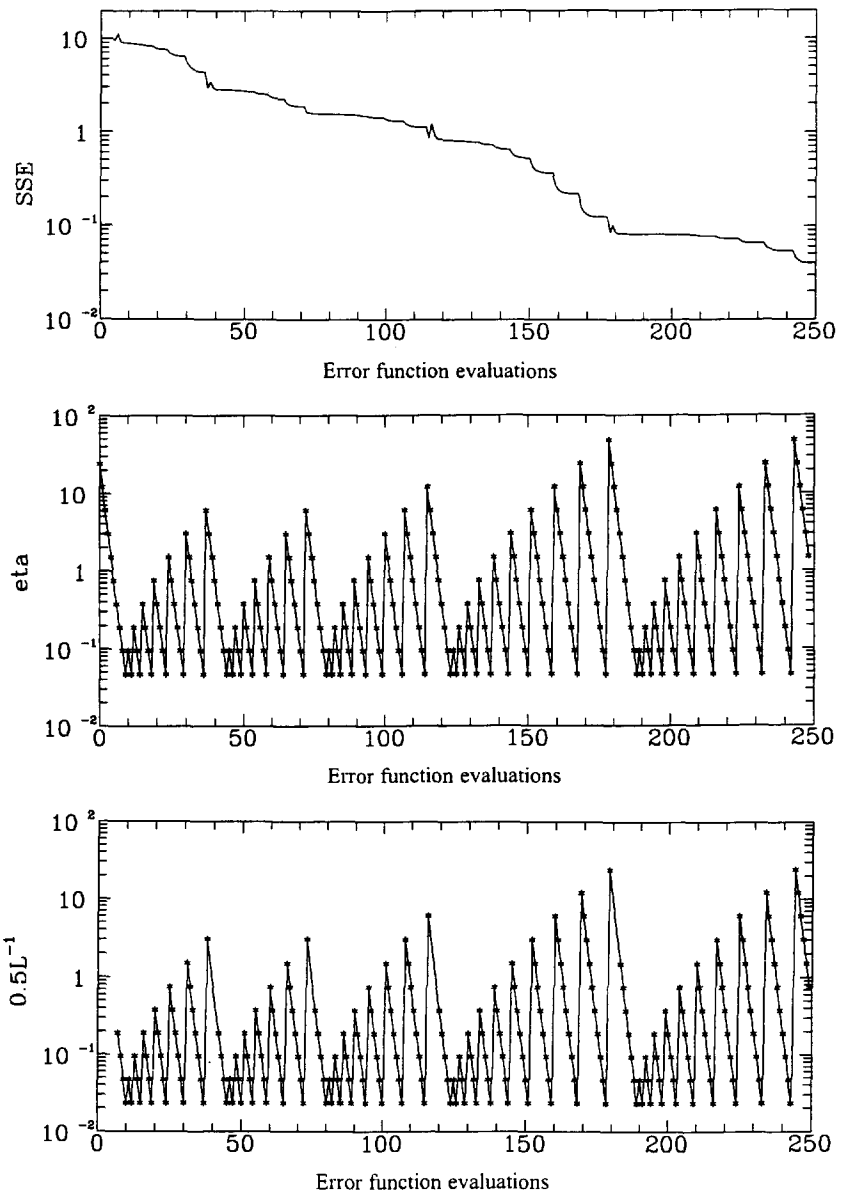

FIGURE 16. Numeric font problem: (a) BPVS convergence behavior; (b) stepsize η behavior; (c) stepsize $0.5L_k^{-1}$ behavior.

The results are summarized in Table 2. BPVS is definitely better than the other algorithms escaping shallow local minima and providing fast training. Figures 16a–c illustrate the error function E, the stepsize η, and the stepsize $0.5L_k^{-1}$ in a typical BPVS run. Note that there was an overshoot of small amplitude in the SSE (see Figure 16a) just before error function evaluation number 120. As soon as the overshoot was detected [i.e., relation (10) was not satisfied] BPVS has stopped following the stepsize $0.5L_k^{-1}$. A safe stepsize was chosen after three error function evaluations according to Steps 3–4 of the BPVS algorithm.

4.4. Numeric Font Generalization Problem

Numerals from zero to nine in eight points standard helvetica font (Figure 17a) form the training patterns for this application example. After being trained with BP, adaptive BP and BPVS as in Application 4.3 the FNN is tested for its generalization capability using helvetica italic (Figure 17b) and helvetica bold (Figure 17c). Note that, the test patterns in italic have 6–14 bits reversed from the training patterns, the test patterns in bold have 6–22 bits reversed from the training patterns, and that 100 FNNs are trained for each case. A test pattern is considered to be correctly recognized if the corresponding output neuron has the greatest value among the output neurons.

BP trained FNNs had similar generalization capability with the adaptive BP trained ones, but as in Application 4.3 more error function evaluations were necessary in order to converge. Therefore, only the performance of the adaptive BP and the BPVS is

0123456789 *0123456789* **0123456789**

(a) standard helvetica (b) helvetica italic (c) helvetica bold

FIGURE 17. Numerals for the numeric font generalization problem: (a) standard helvetica; (b) helvetica italic; (c) helvetica bold.

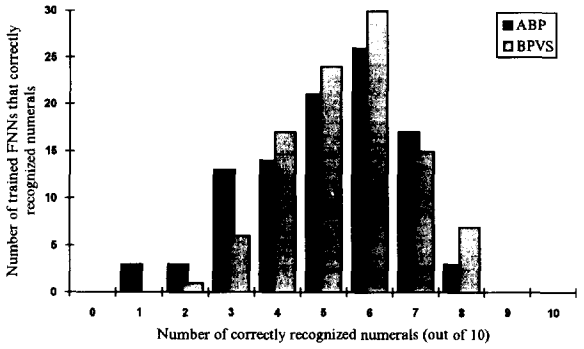

FIGURE 18. Number of trained feedforward neural networks that correctly recognized numerals in helvetica italic.

illustrated in Figures 18 (for the italic) and 19 (for the bold).

As shown in Figure 18, BPVS trained FNNs had slightly better generalization capability than adaptive BP trained ones. For example, seven BPVS trained FNNs recognized eight numerals out of ten. On the other hand, there are only three FNNs trained with adaptive BP that had the same generalization capability.

In the second case, BPVS had definitely better generalization capability than adaptive BP. As shown in Figure 19, BPVS trained FNNs recognized up to nine numerals out of ten. On the other hand,

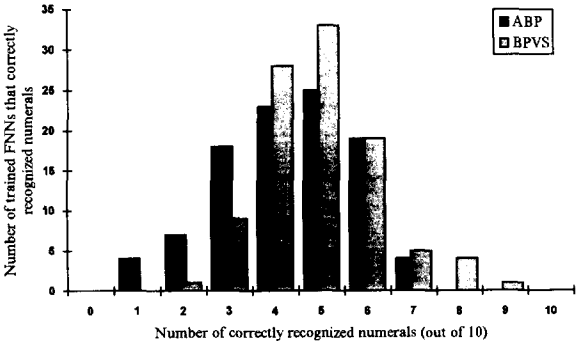

FIGURE 19. Number of trained feedforward neural networks that correctly recognized numerals in helvetica bold.

networks trained with adaptive BP recognized up to seven numerals.

4.5. Texture Classification Problem

A texture classification problem is chosen to compare and evaluate the BP, the adaptive BP, and the BPVS as an example of practical application using continuous-valued training data that contain random noise. Texture classification can be considered as the first step in texture analysis, with important applications in medical imaging, surface or object identification, and processing of satellite images (Haralick, 1979).

A total of 12 Brodatz texture images (Brodatz, 1966): 3, 5, 9, 12, 15, 20, 51, 68, 77, 78, 79, 93 (see Figure 20) of size 512×512 were acquired by a scanner at 150 dpi. From each texture image 10 subimages of size 256 × 256 were randomly selected, and the fractal dimension pattern was computed for

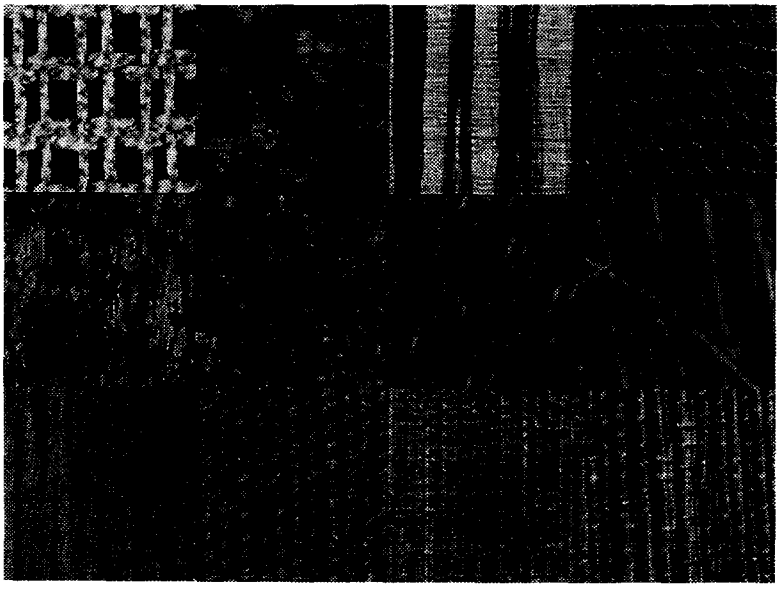

FIGURE 20. Twelve texture patterns obtained from digitizing images found in the "Brodatz Album". Textures: 20, 5, 51, 3, 12, 9, 93, 15, 68, 77, 78, 79.

TABLE 3
Success Rate of Classification by ABP

Texture Type	Input Texture Pattern											
	D20	D5	D51	D3	D12	D9	D93	D15	D68	D77	D78	D79
D20	20											
D5		20										
D51			20									
D3				20		1						
D12					19							
D9					1	17					1	3
D93							20				1	
D15								20				
D68									20			
D77										20		
D78											14	
D79						2					4	17

each one of them. The fractal dimension is an image feature that characterizes the roughness of an image (Pentland, 1984). However, it is possible that two images of different texture and different optical appearance have the same fractal dimension. Thus, its discrimination capability, in some cases is problematic.

In order to alleviate this problem, the fractal dimension was computed in the original subimage, as well as in the first two lower resolution versions of the original subimage and the first two sets of detail subimages, containing higher horizontal and vertical frequency spectral information. The subimages were produced by decomposing the original image through the dyadic wavelet transform (Mallat & Zhong, 1992). The aforementioned feature extraction procedure is originally proposed in Karayiannis & Stouraitis (1995).

Following this procedure ten seven-dimensional training patterns were created from each image. A 7-7-12 FNN (133 weights, 19 biases) was trained to classify the training patterns to 12 texture types. The FNN generalization capability was tested using patterns from 20 subimages of the same size randomly selected from each image. After tuning, stepsize and heuristics were set to the following values: stepsize = 0.001, error ratio = 1.004, stepsize increment factor = 1.005, stepsize decrement factor = 0.4. The termination condition was $E \leq 0.2$, the weights were initialized using the Nguyen–Widrow (1990) method and in all instances ten simulations were run, due to the long training time.

BP never found a global minimum due to oscillations; when BP was approaching a global minimum a smaller stepsize was necessary for the algorithm to continue decreasing the error. Adaptive BP converged to the global minimum needing more than 1.015×10^7 error function evaluations. On the other hand, BPVS converged with an average of 7.1×10^6 error function evaluations. The performance of the two FNNs that had the best training time is presented in Table 3 (adaptive BP trained FNN) and Table 4 (BPVS trained FNN needed 6.242×10^6 error function evaluations), where for each input texture pattern, the count of its classification as a specific texture type by the FNN

TABLE 4
Success Rate of Classification by BPVS

Texture Type	Input Texture Pattern											
	D20	D5	D51	D3	D12	D9	D93	D15	D68	D77	D78	D79
D20	20											
D5		20										
D51			20									
D3				20		1						
D12					20							
D9						17					2	3
D93							20				1	
D15								20				
D68									20			
D77										20		
D78											14	
D79						2					3	17

is given. The total classification rate by BPVS and adaptive BP was significantly high, almost 95%, however BPVS was faster.

5. CONCLUDING REMARKS

BP training with variable stepsize, named BPVS, is presented in this contribution. This algorithm allows arbitrarily large variable stepsize, utilizing estimates of the Lipschitz constant that are obtained without additional error function and gradient evaluations.

The use of the local approximation L_k of the Lipschitz constant at every epoch provides BPVS with valuable information that relates the error function local shape and the subsequent weight updates. This information is used in the stepsize adaptation procedure, in order to achieve satisfactory convergence rate and escape from shallow local minima. In fact, starting with a large stepsize and gradually decreasing it, has an effect similar to annealing (Rumelhart & McClelland, 1986): the algorithm escapes shallow local minima in the early training and converges into a deeper, hopefully global minimum.

The BPVS method has proved to be very effective, when tested and compared with the batch BP, the BP with momentum, and the BP with variable stepsize proposed by Vogl et al. (1988), on several application examples.

The results of the applications suggest that, with no use of heuristic factors the BPVS method successfully adjusts the weights of an FNN. The training algorithm succeeds to converge, within the specified error function evaluation limit, more times than the other algorithms tested and has very small error deviation, which means that it possesses a stable behavior to the desired accuracy. The behavior of the algorithm in the simulations proved to be robust against phenomena such as: oscillations due to large stepsizes and the nearly constant error function value (Lee et al., 1993), by ensuring that the value of the error function E is decreased with every weight update.

Although BPVS is developed for use with deterministic steepest descent, it seems to have a potential for convergence when additive weight perturbations are introduced during training. In a subsequent communication we intend to present the behavior and characteristics of the BPVS convergence under more noise situations and on-line type learning.

REFERENCES

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

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: learning from examples and local minima. *Neural Networks*, **2**, 53–58.

Battiti, R. (1989). Accelerated back-propagation learning: two optimization methods. *Complex Systems*, **3**, 331–342.

Bello, M. G. (1992). Enhanced training algorithms and integrated training/architecture selection for multilayer perceptron networks. *IEEE Transactions Neural Networks*, **3**, 864–874.

Blum, E. K. (1989). Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions. *Neural Computation*, **1**, 532–540.

Brodatz, P. (1966). *Textures—a photographic album for artists and designers*. New York; Dover.

Cater, J. P. (1987). Successfully using peak learning rates of 10 (and greater) in back-propagation networks with the heuristic learning algorithm. *IEEE First International Conference on Neural Networks* (Vol. 11, pp. 645–651).

Chan, L. W., & Fallside, F. (1987). An adaptive training algorithm for backpropagation networks. *Computer Speech Language*, **2**, 205–218.

Darken, C., & Moody, J. (1990). Note on learning rate schedules for stochastic optimization. *Advances in neural information processing systems 3* (pp. 832–838). San Mateo, CA: Morgan Kauffman.

Darken, C., & Moody, J. (1991). Towards faster stochastic gradient search. *Advances in neural information processing systems 4* (pp. 1009–1016). San Mateo, CA: Morgan Kauffman.

Darken, C., Chiang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. *IEEE Second Workshop on Neural Networks for Signal Processing*, 3–12.

Demuth, H., & Beale, M. (1992). *Neural network toolbox user's guide*. Natick, MA: The MathWorks Inc.

Frye, R. C., Rietman, E. A., & Wong, C. C. (1991). Back-propagation learning and nonidealities in analog neural network hardware. *IEEE Transactions Neural Networks*, **2**, 110–117.

Goldstein, A. A. (1962). Cauchy's method of minimization. *Numerische Mathematik*, **4**, 146–150.

Haralick, R. M. (1979). Statistical and structures approaches to texture. *IEEE Proceedings*, **67**, 786–804.

Holt, J. L., & Hwang, J.-N. (1993). Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, **3**, 281–290.

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

Karayiannis, Y. A., & Stouraitis, T. (1995). Texture classification using the fractal dimension as computed in a wavelet decomposed image. In *Proceedings of the IEEE Workshop on Nonlinear Signal and Image Processing* (pp. 186–189). Halkidiki, Greece.

Kollias, S., & Anastassiou, S. (1989). An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Transactions Circuits & Systems*, **36**, 1092–1101.

Lee, Y., Oh, S. H., & Kim, M. (1993). An analysis of premature saturation in backpropagation learning. *Neural Networks*, **6**, 719–728.

Magoulas, G. D., Vrahatis, M. N., Grapsa, T. N., & Androulakis, G. S. (1995). Neural network supervised training based on a dimension reducing method. *Annals of mathematics and artificial intelligence*, to appear (see also *Technical Report CSL-1095*, Department of Electrical & Computer Engineering, University of Patras).

Mallat, S., & Zhong, S. (1992). Characterization of signals from multiscale edges. *IEEE Transactions Pattern Analysis and Machine Intelligence*, **14**, 710–732.

Moler, C., Little, J., & Bangert, S. (1987). *MATLAB user's guide*. Natick, MA: The MathWorks Inc.

Nguyen, D., & Widrow, B. (1990). Improving the learning speed of

2-layer neural networks by choosing initial values of the adaptive weights. *IEEE First International Joint Conference on Neural Networks* (Vol. 3, pp. 21–26).

Pentland, A. P. (1984). Fractal-based description of natural scenes. *IEEE Transactions Pattern Analysis and Machine Intelligence*, **6**, 661–674.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Sakaue, S., Kohda, T., Yamamoto, H., Maruno, S., & Shimeki, Y. (1993). Reduction of required precision bits for backpropagation applied to pattern recognition. *IEEE Transactions Neural Networks*, **4**, 270–275.

Silva, F., & Almeida, L. (1990). Acceleration techniques for the backpropagation algorithm. *Lecture Notes in Computer Science*, **412**, 110–119.

Sperduti, A., & Starita, A. (1993). Speed up learning and network optimization with extended back-propagation. *Neural Networks*, **6**, 365–383.

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

van Ooyen, A., & Nienhuis, B. (1992). Improving the convergence of the back-propagation algorithm. *Neural Networks*, **5**, 465–471.

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., & Alkon, D. L. (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, **59**, 257–263.

Weier, M. K. (1991). A method for self-determination of adaptive learning rates in back-propagation. *Neural Networks*, **4**, 371–379.