PERGAMON

Neural Networks 12 (1999) 1285–1299

Neural Networks

www.elsevier.com/locate/neunet

Contributed article

Accelerating neural network training using weight extrapolations

S.V. Kamarthi*, S. Pittner

Department of Mechanical, Industrial and Manufacturing Engineering, Northeastern University, 334 Snell Eng. Center, Boston, MA 02115, USA

Received 27 June 1997; received in revised form 16 June 1999; accepted 16 June 1999

Abstract

The backpropagation (BP) algorithm for training feedforward neural networks has proven robust even for difficult problems. However, its high performance results are attained at the expense of a long training time to adjust the network parameters, which can be discouraging in many real-world applications. Even on relatively simple problems, standard BP often requires a lengthy training process in which the complete set of training examples is processed hundreds or thousands of times. In this paper, a universal acceleration technique for the BP algorithm based on extrapolation of each individual interconnection weight is presented. This extrapolation procedure is easy to implement and is activated only a few times in between iterations of the conventional BP algorithm. This procedure, unlike earlier acceleration procedures, minimally alters the computational structure of the BP algorithm. The viability of this new approach is demonstrated on three examples. The results suggest that it leads to significant savings in computation time of the standard BP algorithm. Moreover, the solution computed by the proposed approach is always located in close proximity to the one obtained by the conventional BP procedure. Hence, the proposed method provides a real acceleration of the BP algorithm without degrading the usefulness of its solutions. The performance of the new method is also compared with that of the conjugate gradient algorithm, which is an improved and faster version of the BP algorithm.

Keywords: Multilayer neural networks; Backpropagation algorithm; Convergence acceleration; Extrapolation methods; Parameter estimation; Linear regression; Conjugate gradient method; Relative entropy

1. Introduction

The capability of multilayer perceptrons for approximating continuous functions with arbitrary accuracy has been demonstrated through several results (Chen, Chen & Liu, 1995; Mhaskar & Micchelli, 1992). The advent of the backpropagation (BP) algorithm (Rumelhart & McClelland, 1986)—an adaptation of the steepest descent method (Bazaraa, Sherali & Shetty, 1993)—opened avenues for the application of multilayer neural networks for many problems of practical interest (see, e.g. Salehi, Lacroix & Wade, 1998; Sejnowski & Rosenberg, 1987). In this algorithm, an initial weight (or parameter) vector $\mathbf{w}_0$ of a feedforward neural network is iteratively adapted according to the recursion

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta \mathbf{d}_k \tag{1}$$

to find an optimal weight vector. This adaptation is performed by presenting to the network sequentially a set pairs of input and target vectors. The positive constant of η, which is selected by the user, is called the *learning rate*. The direction vector $\mathbf{d}_k$ is the negative of the gradient of the output error function E

$$\mathbf{d}_k = -\nabla E(\mathbf{w}_k). \tag{2}$$

The standard learning scheme for the BP algorithm in which the weights of the network are updated immediately after the presentation of each pair of input and target patterns is called *on-line learning*. An alternative training method treats all the pairs of patterns in the training set as a batch, and it updates the weights only after all training pairs in the batch have been processed, although this can result in slow convergence if the set of training patterns is large. This approach is referred to as *batch learning*. In either case, the vector $\mathbf{w}_k$ contains the weights computed during the kth iteration, and the output error function E is a multivariate function of the weights in the network

$$E(\mathbf{w}_k) = \begin{cases} E_{\mathbf{p}}(\mathbf{w}_k) & \text{for on-line learning,} \\ \sum_{\mathbf{p}} E_{\mathbf{p}}(\mathbf{w}_k) & \text{for batch learning,} \end{cases}$$

where $E_{\mathbf{p}}(\mathbf{w}_k)$ denotes the half-sum-of-squares error function of the network outputs for a certain input pattern $\mathbf{p}$.

The objective of supervised learning (or training) is to

* Corresponding author. Tel.: +1-617-373-3070; fax: +1-617-373-2921.
E-mail address: sagar@coe.neu.edu (S.V. Kamarthi)

0893-6080/99/$ - see front matter
PII: S0893-6080(99)00072-6

Nomenclature

a, b, c	the parameters of a model function
$\mathbf{d}_k$	the direction vector for updating weights in the $(k+1)$th iteration
e	Euler's number
E	the multivariate half-sum-of-squares error function ($E_\mathbf{p}$ is the error function associated with a single input pattern $\mathbf{p}$)
f	a univariate model function
$\mathbf{g}_k$	the gradient used in the $(k+1)$th iteration
H	the relative entropy error measure
i, j	variable indices in the range from 1 to n
I, J	index sets
k	a variable iteration or epoch number
K	a fixed epoch number
l	the index of an extrapolation
L	an upper limit for the learning rate
log	the natural logarithm
M	a certain number of epochs, which is also written as M_l to indicate M at the lth extrapolation
n	the dimension of the weight space
N	the number of epochs considered for a prediction, which is also written as N_l
$\mathbb{N}$	the set of natural numbers
$\mathbb{N}_0$	the set of natural numbers together with 0
O	a function $F : \mathbb{R}^m \to \mathbb{R}$ is said to have the property $F = \mathrm{O}(G)$ as $G \to 0$ for a function $G : \mathbb{R}^m \to \mathbb{R}$ if there exist positive constants D and ϵ such that $\lvert F(x_1, x_2, \ldots, x_m)\rvert \leq D \cdot G(x_1, x_2, \ldots, x_m)$ whenever $G(x_1, x_2, \ldots, x_m) \leq \epsilon$
p_i	a quadratic polynomial
$\mathbf{p}$	a single input pattern
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^n$	the weight space of real-valued column vectors of length n
$\mathbb{R}^+$	the set of positive real numbers
sgn	a function which returns 1 for positive arguments, -1 for negative arguments and 0 for argument 0
t_α	an element of a special sequence $\mathbf{t}$ of real numbers
T	the transposition symbol used for vectors
$\mathbf{v}, \mathbf{w}, \tilde{\mathbf{w}}$	vectors of the weight space; a weight vector $\mathbf{w}$ at epoch k is denoted as $\mathbf{w}_k$ and the optimal weight vector by $\mathbf{w}_*$
w_k	an individual weight at epoch k
$w^i, w_k^i, \tilde{w}^i, w_*^i$	the ith component of a vector $\mathbf{w}$, $\mathbf{w}_k$, $\tilde{\mathbf{w}}$ or $\mathbf{w}_*$, respectively
x	an argument of a univariate function
y_α	an element of a special sequence $\mathbf{y}$ of real numbers
β_k	the momentum rate at iteration $k+2$
η	the learning rate; η_k denotes a dynamic learning rate at iteration $k+1$
ξ	a real-valued variable
$\sum$	the summation sign
χ_I	the characteristic function of a set I
$\forall$	the universal quantifier
∂	the partial derivative symbol
$\lVert\cdot\rVert$	a commonly used norm of a vector such as $\lVert\cdot\rVert_\infty$, which stands for the value of the maximum absolute value of all vector components
∇E	the gradient of a function E

select the set of weights that minimizes E, the deviation between the network outputs and the target patterns, over the complete set of training pairs. Geometrically, the function E specifies an error surface defined over the weight space. Every weight vector $\mathbf{w}_k$ is viewed as a point in the weight space. In this sense, at each iteration the weight update is performed in the direction that yields locally a maximal error reduction. Every cycle in which each one of the training patterns is presented once to the neural network is called an epoch. The learning process continues until E is less than a preset value at the end of an epoch.

The obvious risk of finding only a local minimum for the weight vector or experiencing no convergence is often concealed by the drawback that a large number of iterations is required even for small sized networks. The rate of convergence of the BP algorithm for a practical example

can be expected to be asymptotically linear (Tesauro, He & Ahmad, 1989). This slow rate of convergence is a detriment especially in cases where a network has to continually adapt its parameters in the face of perpetually changing conditions in the application. Reducing the training time while maintaining the framework of BP learning with its simplicity and success as reported in the literature is the aim of the current paper.

This work is structured as follows. In Section 2 earlier techniques modifying the update rules (1) and (2) for accelerating the BP algorithm are reported, and weight extrapolation methods for the same purpose are motivated. Section 3 describes a novel acceleration technique using weight extrapolation. This method, abbreviated as BPWE (backpropagation by weight extrapolation), is shown to reduce the required number of iterations for several problems at the expense of only a few extrapolation steps embedded in the standard BP algorithm. We use a numerical function fitting technique to determine the parameters of extrapolation functions and then use these functions to project weights into the future. In Section 4 the conjugate gradient algorithm, a popular alternative training procedure for multilayer perceptrons is analyzed. The performance of BPWE is evaluated in Section 5 by applying the method to three different examples. We compare these results with those given by the standard BP as well as the conjugate gradient algorithm. Although the paper focuses its applications on multilayer perceptrons and the BP algorithm, the BPWE method can be readily applied to improve any numerical (optimization) procedure, in which real numbers or vectors of real numbers gradually approach a final solution—all it needs is the sequence $(\mathbf{w}_k)$ of weight vectors. One of the most famous examples of this kind is Newton's procedure for finding zeros of differentiable functions. Accordingly, the robustness of BPWE has also been studied in Section 5 by applying the algorithm to example problems using a modified error function as well as different activation functions in both the hidden layer and the output node of the neural networks. The conclusions are reported in Section 6.

2. Possibilities for accelerating backpropagation

Several researchers have investigated different modifications to speed up the convergence of the BP algorithm:

1. Dynamical adaptation of the learning rate:
   1. Line search in the gradient direction, i.e. at each iteration $k+1$ choosing $\eta = \eta_k$ such that it gives the smallest nonnegative local minimum of the function $E(\mathbf{w}_k + \eta \mathbf{d}_k)$ (Dahl, 1987; Hush & Salas, 1988; Jones, Lustig & Kornhauser, 1990; Roy, 1993).
   2. Dynamically adjusting the learning rate, either commonly for all weights (Cater, 1987; Codrington & Mohandes, 1994; Mohandes, Codrington & Gelfand, 1994; Nachtsheim, 1994; Salomon & van Hemmen, 1996; Weir, 1991) or separately for each weight (Jacobs, 1988; Pirez & Sarkar, 1993; Silva & Almeida, 1990).
2. Dynamical adaptation of the weight adjustments expressed by the vectors $\mathbf{d}_k$:
   1. Determining $\mathbf{d}_k$ for $k \geq 1$ by a relation of the form $\mathbf{d}_k = -\nabla E(\mathbf{w}_k) + \beta_{k-1}\mathbf{d}_{k-1}$, i.e. by adding a "momentum term" of the form $\beta_{k-1}\mathbf{d}_{k-1}$ to the current weight adjustment vector $\mathbf{d}_k$ (Rumelhart & McClelland, 1986; Yu, Loh & Miller, 1993). The nonnegative parameters β_{k-1} are called *momentum rates*, which determine the effect of past weight changes on the current direction of movement in the weight space. This tends to magnify descent in steady downhill directions, while having a "stabilizing" effect for rapidly changing directions.
   2. Other methods for the determination of $\mathbf{d}_k$ which include only first partial derivatives of the error function E (Fahlman, 1989; Kanda, Fujita & Ae, 1994).
3. Alternating computation of several iterations of the BP algorithm and the extrapolation of the computed sequence $(\mathbf{w}_k)$ of weight vectors (Cho & Kim, 1990; Dewan & Sontag, 1990; Yamada, Pecharanin, Taguchi, Iijima & Sone, 1997). The special feature of these additional iterations is that updates of a weight vector $\mathbf{w}_k$ are computed without using information about the error function E. This property holds true for the method presented in the current paper also.

Many other dramatical alterations of the BP algorithm, some of which are based on combinations of the items listed above, have been proposed. However, some of these approaches require complex and costly calculations at each iteration, which offset their faster rate of convergence.

A widespread class of acceleration methods which are considerably more complex are the second-order methods. They are divided into the Newton method and the quasi-Newton methods. The idea behind the Newton method is that during each iteration the error function E is approximated locally by a second-degree Taylor polynomial, which is minimized subsequently. This minimization requires the knowledge of a Hessian of the error function E and the solution of a system of linear equations, which is computationally prohibitive (Battiti, 1992; Bishop, 1995). To overcome this situation, quasi-Newton methods (Dennis & Schnabel, 1983; Robitaille, Marcos, Veillette & Payre, 1996) are preferred, which compute approximations of the Hessian or its inverse in an iterative process. Both kinds of second-order methods can lead to considerable storage requirements.

Another disadvantage of most acceleration techniques is that they must often be "tuned" to fit the particular application. One of the most popular alternatives to the BP scheme, the conjugate gradient algorithm, is based on the items 1.1 and 2.1 and is discussed in Section 4. A

systematic comparison of some of the acceleration methods for BP was done by Pfister and Rojas (1993).

In this work, we use the idea described in item 3. However, instead of calculating weight extrapolations every now and then during the BP algorithm, our method first watches if the weights change smoothly during a time interval of the training process, and only then are the proposed extrapolations computed. Furthermore, our model function differs considerably from the existing approaches.

3. Weight extrapolation strategy

In contrast to minimization techniques such as the conjugate gradient algorithm that rely on second-order partial derivatives of the error function E, the BP algorithm is strikingly slow, particularly when approaching the final optimum. The improvement of this slow behavior at an advanced phase of training is the goal of this section.

Consider that in a given learning process a certain component of the gradient ∇E is either positive and monotonically decreasing or negative and monotonically increasing for several iterations. This fact suggests that the error surface has a smooth variation along the respective axis, and therefore extrapolations for those particular components should be possible. For performing an extrapolation, the weights are recorded at the end of each epoch. By examining in the BP learning the convergence behavior of each network weight w individually (see Dewan & Sontag, 1990), it seems adequate to use extrapolation by a function of the general form

$$f(x) = a - be^{-cx} \tag{3}$$

with $b \neq 0$, $c > 0$ and an arbitrary constant a. The argument x represents an epoch number, i.e. for epoch number k

$$f(k) = w_k. \tag{4}$$

We use this observation as the basis for the speed-up technique proposed in this paper. With this technique, the network weights for a future epoch are predicted by weight extrapolation. This prediction relies on M preceding epochs during which every individual component behaves according to the extrapolation function f. This behavior, which triggers our extrapolation procedure, is summarized by the following definition.

Definition 1. Let M be a specified number of epochs. Then a sequence $(\mathbf{w}_k)$ of weight vectors is called quasi-exponentially convergent for the preceding M epochs at a certain epoch $K \geq M$ if

1. the sequence $(w_{K-M+1}, w_{K-M+2}, \ldots, w_K)$ of every individual component of $\mathbf{w}_k$ is strictly monotonically either increasing or decreasing, and
2. the distances between consecutive elements in every such sequence are strictly monotonically decreasing, i.e.

$$|w_k - w_{k+1}| > |w_{k+1} - w_{k+2}|$$

for $k = K - M + 1, K - M + 2, \ldots, K - 2$.

It is easy to show that a function f as defined in Eq. (3) leads, for all positive integers K and M with $K \geq M$, to a sequence $(w_{K-M+1}, w_{K-M+2}, \ldots, w_K)$ that satisfies conditions 1 and 2 of Definition 1. In fact, the stated sequence is strictly monotonically increasing for $b > 0$ and strictly monotonically decreasing for $b < 0$.

Condition 1 of Definition 1 seems to be very restrictive since it demands that all individual weights have a monotonic behavior simultaneously. However, one of the main reasons for the slow convergence of the BP algorithm can be attributed to the fact that in general the error surface E contains an extremely flat "valley" in a vicinity (of small radius) of the final weight vector (Balakrishnan & Honavar, 1992; Parekh, Balakrishnan & Honavar, 1993). In this region, the decrease in the error function during any iteration is very small compared to the learning rate η. It will become clear from the proof of Proposition 2 that under these circumstances all individual weight sequences are monotonic. Nevertheless, an extension of the presented BWPE algorithm for nonmonotonic weight behavior will be addressed in Section 3.4.

3.1. Selection of values for M and N

At the beginning of a training process (when K is small), the convergence of a weight vector $\mathbf{w}_K$ can substantially be improved with a coarse extrapolation using only a short preceding sequence (i.e. M is small) of the form $(\mathbf{w}_{K-M+1}, \mathbf{w}_{K-M+2}, \ldots, \mathbf{w}_K)$. On the other hand, if $\mathbf{w}_k$ is positioned close to a local minimum at a more advanced phase of training, a very accurate extrapolation of $(\mathbf{w}_K)$ using a lengthy sequence of previous weight vectors will be necessary because of the danger of "overshooting" the goal. Therefore, we decided to double the value of M each time an extrapolation has been performed. With the starting value $M_1 = 8$ for the first extrapolation, the size M at the lth extrapolation is in fact given by

$$M = M_l = 2^{l+2}. \tag{5}$$

The weights are predicted from the past M weight values only every now and then during the iterative training process.

At the lth extrapolation, the method predicts the result of training for a specified number of future epochs N_l by processing the information of the weight vectors computed during the preceding M_l epochs. The extrapolated weights are used to "jump" over N_l epochs of the standard BP algorithm, so that significant computational savings are expected. The danger of overshooting a local minimum is a reason that it is not favorable to predict the final optimal

weight vector $\mathbf{w}_\infty$. Instead we fixed N_l as

$$N = N_l = 70M_l = 35 \times 2^{l+3}.$$

With this choice, where the factor 70 was determined experimentally, the number N_l of future epochs considered for the prediction is larger for larger M_l, because more accurate predictions are expected from the larger number of previous weight vectors.

3.2. Extrapolation algorithm

Now we consider a sequence $(\mathbf{w}_k)$ that is quasi-exponentially convergent for the preceding $M = M_1$ epochs at a certain epoch K according to Definition 1. We select an arbitrary component of the weight vector and consider the corresponding sequence $(w_{K-M+1}, w_{K-M+2}, \ldots, w_K)$ for illustrating extrapolation calculations. The extrapolation routine is the same for every other component. By the properties 1 and 2 of Definition 1, the sequence $(w_{K-M+1}, w_{K-M+2}, \ldots, w_K)$ follows the exponential trend of the function f in the sense of Eq. (4), which means that

$$w_k \approx a - be^{-ck} \tag{6}$$

for every $k = K - M + 1, K - M + 2, \ldots, K$ with unknown parameters a, b and c that have to be determined. It follows that the successive differences of these weights satisfy the relation

$$|w_k - w_{k+1}| = |b| \cdot |e^{-ck} - e^{-c(k+1)}| = |b|e^{-c(k+1)}(e^c - 1), \tag{7}$$

because c is positive. Taking the logarithm of both sides yields that

$$\log|w_k - w_{k+1}| = d - c(k+1) \tag{8}$$

with the constant $d := \log|b| + \log(e^c - 1)$ for $k = K - M + 1, K - M + 2, \ldots, K - 1$. The parameters c and d can be fit via linear regression. Given these parameters, we find b through

$$|b| = e^d(e^c - 1)^{-1}. \tag{9}$$

According to the remark following Definition 1, the constant b is positive for a strictly monotonically increasing sequence $(w_{K-M+1}, w_{K-M+2}, \ldots, w_K)$ and negative if this sequence is strictly monotonically decreasing. Motivated by the transformation

$$a \approx w_k + be^{-ck}$$

of relation (6) we can calculate a as the arithmetic mean

$$a = \frac{1}{M}\Big(\sum_{k=K-M+1}^{K} w_k + b \sum_{k=K-M+1}^{K} e^{-ck}\Big)$$

$$= \frac{1}{M}\left(\sum_{k=K-M+1}^{K} w_k + be^{-c(K-M+1)}\frac{1 - e^{-Mc}}{1 - e^{-c}}\right). \tag{10}$$

The completely specified function f is used to predict a future weight after $N = N_1$ additional epochs, and we set the weight w_{K+1} equal to this prediction,

$$w_{K+1} = f(K + N). \tag{11}$$

After every single projected weight value w_{K+1} is calculated, the weight vector $\mathbf{w}_{K+1}$ is fed back into the BP algorithm. The values of M and N are updated to $M = M_2$ and $N = N_2$. Thereupon, the standard BP iterations continue until the next quasi-exponentially convergent sequence of length M_2 is found, and so on. The iterative BP training process is interrupted for an extrapolation step very sporadically, namely when the conditions for extrapolation are met. This method is referred to as BPWE during the rest of this paper.

The main part of every extrapolation consists of the approximate solution of the $M - 1$ equations specified by Eq. (8) for every single weight. Afterwards, the parameters of the corresponding functions of the form $f(x) = a - be^{-cx}$ are determined through Eqs. (9) and (10), and the extrapolation of the weights is performed as defined in relation (11).

3.3. An alternative parameter estimation

It should be noted that a similar performance of the proposed method BPWE is expected, if one uses the following method instead of the linear regression approach in the calculation of the parameters b and c. It follows from Eq. (7) that

$$\frac{|w_k - w_{k+1}|}{|w_{k+1} - w_{k+2}|} = \frac{e^{-c(k+1)}}{e^{-c(k+2)}} = e^c,$$

for $k = K - M + 1, K - M + 2, \ldots, K - 2$, i.e. the parameters b and c can be estimated with

$$c = \frac{1}{M-2}\sum_{k=K-M+1}^{K-2}(\log|w_k - w_{k+1}| - \log|w_{k+1} - w_{k+2}|),$$

$$|b| = \frac{1}{(M-1)(e^c - 1)}\sum_{k=K-M+1}^{K-1} e^{c(k+1)}|w_k - w_{k+1}|.$$

3.4. A simple theoretical example

The next proposition demonstrates the effect of our proposed extrapolation scheme on the BP algorithm (for batch learning) on the basis of a class of error functions, which are so simple that they allow one to demonstrate the behavior of the BP and BPWE algorithms near the final weight vector only coarsely. These error functions can be obtained by using the first three terms of the Taylor series

approximation

$$E(\mathbf{w}) = E(\tilde{\mathbf{w}}) + (\nabla E(\tilde{\mathbf{w}}))^{\mathrm{T}}(\mathbf{w} - \tilde{\mathbf{w}}) + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^{\mathrm{T}} D(\mathbf{w} - \tilde{\mathbf{w}}) + \frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{\partial^2 E}{\partial w^i \partial w^j}(\tilde{\mathbf{w}})(w^i - \tilde{w}^i)(w^j - \tilde{w}^j) + \cdots \tag{12}$$

of more general functions $E : \mathbb{R}^n \to \mathbb{R}$ at an arbitrary point $\tilde{\mathbf{w}} \in \mathbb{R}^n$. In relation (12) we have $\mathbf{w} = (w^1, w^2, \ldots, w^n)^{\mathrm{T}}$, $\tilde{\mathbf{w}} = (\tilde{w}^1, \tilde{w}^2, \ldots, \tilde{w}^n)^{\mathrm{T}}$ and $D = (d_{ij})$ is an $n \times n$-diagonal matrix with diagonal elements $d_{ii} = \partial^2 E/\partial(w^i)^2$.

Proposition 2. *Assume that the function* $E : \mathbb{R}^n \to \mathbb{R}$ *has the form*

$$E(\mathbf{w}) = p_1(w^1) + p_2(w^2) + \cdots + p_n(w^n) + C, \tag{13}$$

where $\mathbf{w} = (w^1, w^2, \ldots, w^n)$, $C \in \mathbb{R}$, *and every* $p_i : \mathbb{R} \to \mathbb{R}$ $(i = 1, 2, \ldots, n)$ *is a quadratic polynomial of the form*

$$p_i(x) = A_i x^2 + B_i x + C_i$$

with $A_i, B_i, C_i \in \mathbb{R}$. *Then this function E has an isolated local minimum if and only if every* A_i *is positive. In this case, which is assumed during the rest of this proposition, there exists in fact only one local minimum* $\mathbf{w}_*$ *which is also the global minimum of E.*

The BP algorithm with batch learning started at a weight vector $\mathbf{w}_0 \neq \mathbf{w}_*$ *converges if and only if the learning rate* η *satisfies the inequality*

$$\eta < L, \tag{14}$$

where $L := (\max_{i \in I} A_i)^{-1}$. *The set I is defined by* $I := \{i \in \{1, 2, \ldots, n\} : w_0^i \neq w_*^i\}$, *where* w_0^i *and* w_*^i *are the i*th *components of* $\mathbf{w}_0$ and $\mathbf{w}_*$, *respectively. This also holds true for BPWE. Moreover, for the case where* inequality (14) *is satisfied, BP and BPWE have the following convergence behavior:*

1. *The BP algorithm converges in an infinite learning process to the global minimum* $\mathbf{w}_*$ *of E with a linear rate of convergence. The only exception is the case where*

$$\eta = \frac{1}{2A_i} \qquad \forall i \in I. \tag{15}$$

   In this situation, the minimum is reached after one epoch.
2. *The BPWE algorithm applies extrapolations if and only if* $\eta < L/2$ *with L as defined above and*

$$I = \{1, 2, \ldots, n\} \tag{16}$$

   Otherwise its behavior is identical to that of the BP algorithm described in item 1. If $\eta < L/2$ *and* condition (16) *is valid, then BPWE converges faster than BP to the global minimum* $\mathbf{w}_*$ *with a speed-up factor between* 25 *and* 71 *after* 8 $(= M_1)$ *epochs, which only depends on the number of epochs until termination.*

It is easy to show that the form of E in Eq. (13) of Proposition 2 is in fact equivalent to the form

$$E(\mathbf{w}) = C_0 + \mathbf{v}^{\mathrm{T}}(\mathbf{w} - \tilde{\mathbf{w}}) + \frac{1}{2}(\mathbf{w} - \tilde{\mathbf{w}})^{\mathrm{T}} \times \begin{pmatrix} A_1 & & & 0 \\ & A_2 & & \\ & & \ddots & \\ 0 & & & A_n \end{pmatrix} (\mathbf{w} - \tilde{\mathbf{w}})$$

with arbitrary vectors $\mathbf{v}, \tilde{\mathbf{w}} \in \mathbb{R}^n$ and arbitrary constants $A_1, A_2, \ldots, A_n, C_0 \in \mathbb{R}$. The proof of Proposition 2 is presented in Appendix A.

By allowing also weight sequences that are alternately nonincreasing and nondecreasing in item 1 of Definition 1, the BPWE method could be extended to model functions of the more general form

$$f(x) = a - b(\pm e^{-c})^x$$

with $a, b \in \mathbb{R}$ and $c > 0$. The formulas for parameter estimation with this alternative BPWE method could be derived similar to Section 3.2. Moreover, this alternative BPWE procedure also speeds up the BP for a function E of the form stated in Proposition 2 by a factor between 25 and 71, where our version of BPWE behaves like the BP (the only exception is the case where both methods compute the minimum with one epoch).

3.5. Time and storage requirements

The essential storage requirement of the BP algorithm consists of all n weights of the feedforward neural network. According to the description in Section 3.2 and the formula for linear regression (Ryan, 1997) an efficient implementation of the BPWE method will update the following values during each iteration K in which a quasi-exponentially convergent weight sequence (starting at iteration k_1) is encountered:

$$\sum_{k=k_1}^{K} \log|w_k - w_{k+1}|, \qquad \sum_{k=k_1}^{K} (k+1)\log|w_k - w_{k+1}|$$

for the linear regression (8), $\sum_{k=k_1}^{K} w_k$ for formula (10), and $w_k - w_{k+1}$ for the next comparisons according to Definition 1. Since no additional past weight values have to be stored, the essential extra storage space required by BPWE amounts to $4n$ units, which can be considered quite favorable.

For the determination of the computational complexity of BPWE relative to the BP, three different cases have to be distinguished. If the weight sequence is not quasi-exponentially convergent for the iteration under consideration, the

time complexity of the BP and BPWE can be considered as the same. The computational complexity per iteration of the BP is around n multiplications for the forward pass and around $2n$ multiplications for the backward pass, and n multiplications for multiplying the gradient $\mathbf{d}_k$ with the learning rate η (Hush & Salas, 1988). In summary, we get four multiplications per iteration per weight.

If the weight sequence is quasi-exponentially convergent but has not reached the length M_l required by BPWE, the extra computation of the BPWE algorithm is just that of one multiplication and the computation of a logarithm per iteration per weight. However, if the criteria of Definition 1 are satisfied for M_l iterations, the time complexity of an iteration with BPWE is approximately eight times that of the BP. However, the doubling of the value M after each extrapolation as defined in Eq. (5) keeps such costly iterations rare.

After the description of this acceleration technique for the BP algorithm, our aim is to illustrate its advantages from a practical point of view. For this purpose, the conjugate gradient method allows a good comparison since it highly optimizes each individual step of the BP algorithm. This principle is common for most existing acceleration methods, but is different from the strategy used in the BPWE algorithm where highly optimized steps are applied only rarely. The principal structure and characteristics of the conjugate gradient method are briefly described in the next section.

4. Conjugate gradient algorithm

One of the reasons for the slow convergence of the BP algorithm is that an iteration that reduces the error $E_{\mathbf{p}}$ with respect to one pattern $\mathbf{p}$ will not, in general, result in an error reduction with respect to all the patterns together. Such a step may misdirect the optimization path and thus may increase considerably the number of iterations required for convergence. With the following conjugate gradient algorithm for training feedforward neural networks, this problem can be overcome. The existing empirical evidence that it converges faster than BP training is strengthened in this paper.

Conjugate gradient (CG) algorithm

Step 1. Initialize the weight vector $\mathbf{w}_0$ with random values.
Step 2. Compute the gradient $\nabla E(\mathbf{w}_0)$ and test it. If $\|\nabla E(\mathbf{w}_0)\|$ is small enough then STOP and return $\mathbf{w}_0$ as the desired minimum; otherwise continue.
Step 3. Set $\mathbf{g}_0 := \nabla E(\mathbf{w}_0)$ and set the direction $\mathbf{d}_0 := -\mathbf{g}_0$.
Step 4. Initialize $k = 0$.
Step 5. Calculate

$$\eta_k = \max\{\eta \geq 0 : E(\mathbf{w}_k + \xi \mathbf{d}_k)$$

is strictly monotonically decreasing for $\xi \in [0, \eta]\}$.
Step 6. Set $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta_k \mathbf{d}_k$.
Step 7. Compute the gradient $\nabla E(\mathbf{w}_{k+1})$ and test it. If $\|\nabla E(\mathbf{w}_{k+1})\|$ is small enough then STOP and return $\mathbf{w}_{k+1}$ as the desired minimum; otherwise continue.
Step 8. Set $\mathbf{g}_{k+1} := \nabla E(\mathbf{w}_{k+1})$ and set the direction $\mathbf{d}_{k+1} := -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$.
Step 9. Increase k by 1 and go to step 5.

Similar to the BP method, the conjugate gradient algorithm is iterative, but the error function E is evaluated and weights are modified only after the presentation of the complete set of training patterns. This means that the vectors $-\mathbf{g}_k$ show the direction of steepest descent for the sum-of-squares error E over all the training patterns. This is in contrast to the standard BP algorithm, where the vectors $\mathbf{d}_k$ indicate the direction of steepest descent only with respect to the partial error functions $E_{\mathbf{p}}$.

Note that the basic structure of the weight update rule in step 6 of the CG procedure is the same as for the BP algorithm in relation (1). Furthermore, the CG method can be regarded as an extension of the BP training by automatically selecting an appropriate learning rate in each epoch as well as including a momentum term as described in Section 2.

4.1. Line search

It is obvious that the line search procedure in step 5 assures either a reduction in the error function E or constancy for $\eta_k = 0$. Note that this latter situation can occur, because in contrast to the BP algorithm with batch learning, the obtained directions $\mathbf{d}_k$ are not necessarily local descent directions. Therefore, this one-dimensional function minimization prevents (in general) cycling on the error surface and overshooting of a local minimum, two phenomena, which are frequently observed in the BP. On the other hand in the CG method, the sequence $(\mathbf{w}_k)$ of weight vectors may converge to a bad local minimum, because the CG algorithm moves towards the bottom of whatever valley it reaches. The reason is that "escaping" a local minimum requires an increase in the overall error function E, which is excluded by the line search procedure. As will also be demonstrated by the simulations in Section 5, the standard BP algorithm allows this possibility for two reasons:

1. For the BP, a constant learning rate η regardless of the shape of the error surface is used, which may lead to a "jump" of $(\mathbf{w}_k)$ over the local minimum.
2. The updating direction $\mathbf{d}_k$ in relation (1), which uses partial gradients $\nabla E_{\mathbf{p}}$ rather than the total gradient ∇E, is determined by a single training pattern. Such a rule does not take into account that another pattern would change the weight vector in a completely different direction. This may also lead to an increase in the error function E during the iterations.

However, it becomes clear from the theoretical derivation of the CG method (Pittner, 1996; Polak, 1971) that step 5 is a fundamental part of the underlying concept.

The difficulty and cost of finding the exact value of the first minimum of $E(\mathbf{w})$ along the direction $\mathbf{d}_k$ starting at

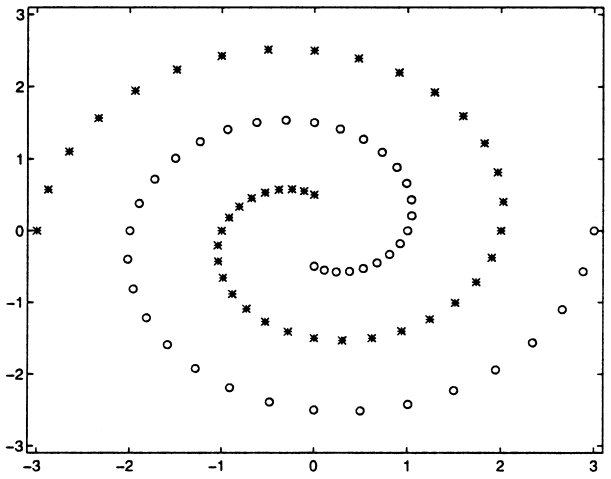

Fig. 1. Spiral training points where $*$ denotes the spiral with target value 0 and $\circ$ denotes the spiral with target value 1.

$E(\mathbf{w}_k)$ resulted in the common inclusion of inexact line searches in implementations of the traditional CG method. Leonard and Kramer (1990) demonstrated that the CG algorithm converges most quickly when the line search in step 5 is done only coarsely. Therefore, we performed this one-dimensional minimization in our simulations based on an iterative method of Brent (1973) that uses derivatives of the function $E(\mathbf{w}_k + \xi\mathbf{d}_k)$ and allowed a relatively large tolerance (Press, Teukolsky, Vetterling & Flannery, 1992).

4.2. Choice of β_k

The only parameters that remain to be defined for the CG algorithm are the values of the sequence (β_k), which we have taken as

$$\beta_k = \frac{\mathbf{g}_{k+1}^{\mathbf{T}}(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^{\mathrm{T}}\mathbf{g}_k}.$$

This assignment is due to Polak and Ribière (1969) and is generally considered to be one of the best choices from a practical point of view (e.g., Fitch, Lehmann, Dowla, Lu, Johansson & Goodman, 1991). A profound demonstration of the fast convergence of the CG method for feedforward neural networks with this choice was first provided by Kramer and Sangiovanni-Vincentelli (1988) as well as Charalambous (1992). Moreover, efforts have been undertaken for improving these kind of methods in order that fast convergence can also be assured theoretically (Gilbert & Nocedal, 1992; Powell, 1977).

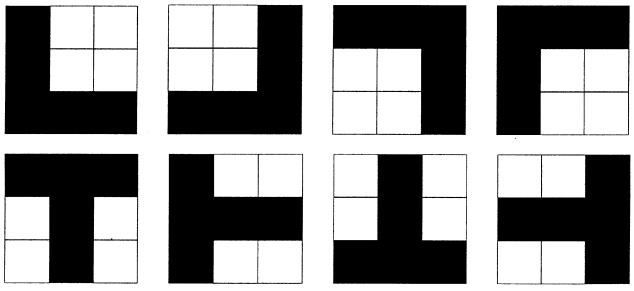

Fig. 2. Eight learning patterns used for the third simulation example.

4.3. Quality of the conjugate gradient method

While the storage requirement for the CG algorithm is about four times that for the BP, the computation time per cycle is more significantly increased because of the line search necessary for determining an appropriate step size. Determining the learning rates with a line search involves several evaluations of either the error function E or its derivative, both of which raise the computational complexity per epoch, since evaluating E includes the presentation of all input patterns to the network.

It has been confirmed empirically that in general the local minimum achieved with the BP algorithm will in fact be a global minimum, or at least a solution that is good enough for most purposes. In contrast, the CG algorithm will lead the network to often settle down in a bad local minimum from which it cannot escape and consequently it will impair the generalization ability of the network (Towsey, Alpsan & Sztriha, 1995). This is a serious limitation of the CG method.

5. Simulation results

In this section the convergence behavior of the BPWE, standard BP and CG algorithms are compared on three example problems. The learning rate was fixed at $\eta = 0.9$ in BPWE and BP algorithms for all three example problems. For all tests reported, a bipolar activation function $\sigma_{\mathrm{h}}(x) = [1 - \exp(-x)]/[1 + \exp(-x)]$ was used for the hidden nodes and a logistic activation function $\sigma_{\mathrm{o}}(x) = [1 + \exp(-x)]^{-1}$ for the output nodes of a fully connected feedforward neural network. The convergence of the learning process is measured by taking the half-sum-of-squared errors as the objective function. We used sum-of-squared-error less than or equal to 10^{-4} as the stopping criterion, and the same initial weights for all three methods. However, the CG method was terminated once it was stranded in a local minimum with no further error reduction possible. All three networks were simulated in C on a Sun SPARC-4 Ultra 5 workstation.

5.1. Experimental details and results

XOR problem. The first test was performed with the exclusive-or (XOR) problem, which is the most popular benchmark for neural network training. The network architecture used for this problem consisted of two input units, one hidden layer with three units and one output unit. The network mapped each of the four pairs of input patterns into the corresponding output target value.

Two spirals problem. For the second training task we

Table 1
Simulation results for three example problems

		BP algorithm	BP algorithm with weight extrapolations	Conjugate gradient method
XOR problem	Epochs required	16 718	4886	14
	Learning time (s)	2.6	1.0	0.1
	Total error of solution	0.0001	0.00008	0.012
	Number of extrapolations	–	5	–
Two spirals problem	Epochs required	11 188	3798	17
	Learning time (s)	59.0	20.4	1.6
	Total error of solution	0.0001	0.0001	18.16
	Number of extrapolations	–	4	–
L–T problem	Epochs until solution	1940	1811	5
	Learning time (s)	0.7	0.6	0.1
	Total error of solution	0.00001	0.00001	0.011
	Number of extrapolations	–	1	–

selected the "two spirals separation problem" examined by Lang and Witbrock (1989). The difficulty of this problem has been demonstrated in many attempts to solve the problem with backpropagation and several elaborated modifications. The input pattern set consists of the pairs of coordinates describing the points of two intertwined spirals in the x–y-plane. The network is trained to discriminate points lying on these two separate spirals. The membership of an input point to one or the other spiral was indicated by the target values 0 and 1, respectively. Our simulation tests use only the original patterns within a radius of three units. Since the BP algorithm is unable to locate more than suboptimal solutions of the two spirals problem for networks with one hidden layer (see Baum & Lang, 1991), a network with two hidden layers was used. The network was built up of an input layer of two units, each one representing a coordinate, a first hidden layer with eight units, a second hidden layer with two units and an output layer with one unit. A training set of 82 input–output pairs was created by randomly merging the points of the spirals depicted in Fig. 1.

L–T problem. The third experiment conducted a simple L–T letter recognition task. The network had nine input units, two hidden units and a single output unit, and was trained to recognize the letters L and T. Each input pattern was a 3×3 pixel binary image of a letter. The training set was formed by eight patterns, all four orientations for each letter (see Fig. 2). The letters L and T were indicated by the target values 0.05 and 0.95, respectively, for the output unit. For this case, the original BP method required 1940 iterations while the modified method BPWE required 1811 iterations. Since the BP algorithm converged very fast for this problem, the termination tolerance was decreased from 10^{-4} to 10^{-5}.

We trained neural networks and studied the convergence behavior in these three simulation experiments. The simulation results are summarized in Table 1 for all three examples. The results demonstrate the effectiveness of the proposed method, since the BPWE algorithm requires a much smaller number of iterations than the standard BP procedure. This is also reflected by a marked reduction of CPU time with the exception of the third example. These examples show that the proposed new approach spends some additional time for storing computed weights and

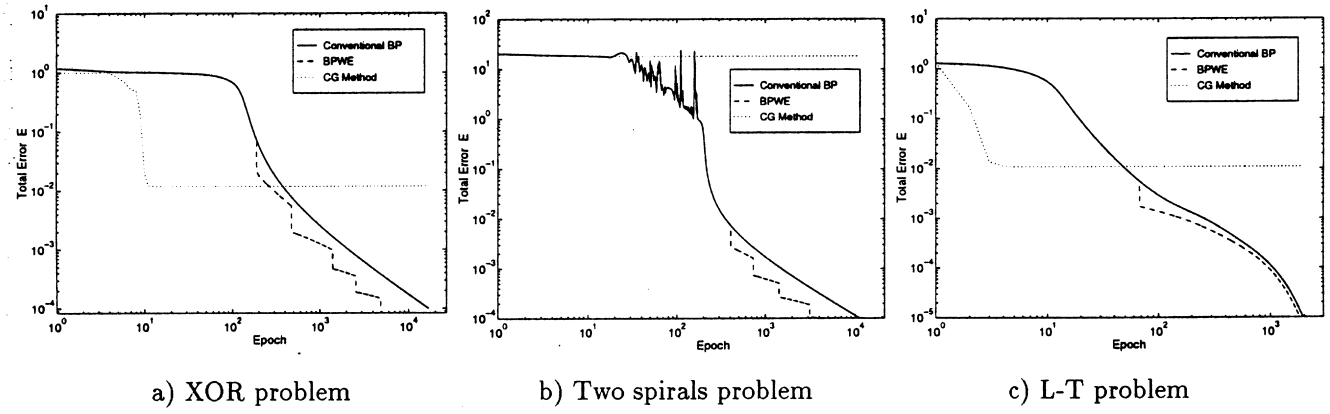

Fig. 3. Evolution of error as a function of epoch number when network is being trained by backpropagation (BP), backpropagation with weight extrapolations (BPWE), and conjugate gradient (CG) method.

gradient vectors and for computing accelerating weight extrapolations, but saves time in an overcompensating manner by reducing the number of epochs required to achieve a desired error value. This means that the proposed improvement of the BP algorithm can be achieved at the cost of an insignificant increase in computational load per epoch.

In Fig. 3 the convergence behavior of the total error E during the training process is compared for three algorithms on all three example problems. Both axes of the plots are logarithmic. Since BP and BPWE update the weights after each presentation of a training pattern, and the CG algorithm only after the whole training set has been presented, error updates were counted in terms of epochs. As can be observed from the graphs, the proposed BPWE algorithm leads to a sharp decrease in the total error E compared with that of standard BP after the activation of extrapolations. In the BPWE algorithm the activations of extrapolations are performed only after E shows a steady tendency to approach zero, i.e. when a smooth part of the error surface containing a practically useful local minimum has been reached. It is clear from Fig. 3 that in this phase of training there is a high potential for performing weight extrapolations, since the BP algorithm makes a very slow progress in getting closer to the minimum.

The CG algorithm had a completely different course of learning. It was only able to train the networks to an error value not smaller than 0.01, which means that it could not reach an acceptable solution. This was accomplished within a few seconds, but then the error remained constant such that the stated convergence condition could never be reached. This behavior means that the CG method has settled down in a bad local minimum.

5.2. Repeated runs of the conjugate gradient method

As was already done by Fitch et al. (1991) we applied the CG method several times with different random starting points $\mathbf{w}_0$ for the weight vector for all three example problems. The final weight vector, which gave the minimum error level was taken as the result. To keep the computation time of the CG method the smallest among the three training algorithms, four different initial settings were used for each problem, resulting in an average reduction of the final error by a factor of six. Still, the convergence conditions have never been reached.

5.3. Relative entropy error measure

Besides the standard half-sum-of-squares error function E, several other error measures are occasionally used in neural network training, especially for classification problems. One of these error measures is called relative entropy H and is based on the following lemma (Jelinek, 1968). In this lemma, and during the rest of this section we assume that $0\cdot\log 0 := 0$ and $0/0 := 1$.

Lemma 3. *Let $(q_1, q_2, \ldots, q_m)$ and $(r_1, r_2, \ldots, r_m)$ be two nonzero sequences of the same length m which satisfy the properties $q_\alpha \geq 0$, $r_\alpha > 0$ for $\alpha = 1, 2, \ldots, m$ and*

$$\sum_{\alpha=1}^{m} q_\alpha = \sum_{\alpha=1}^{m} r_\alpha.$$

Then the inequality

$$\sum_{\alpha=1}^{m} q_\alpha \log \frac{q_\alpha}{r_\alpha} \geq 0$$

is satisfied, where equality holds if and only if $q_\alpha = r_\alpha$ for all α.

For a neural network with a single output node with target values $t(\mathbf{p}) = \pm 1$ and the actual network outputs $y(\mathbf{p})$ contained in the interval $[-1,1]$ the relative entropy error measure H (Hertz, Krogh & Palmer, 1991) is formulated as

$$H = \frac{1}{2}\sum_{\mathbf{p}}\left[(1 + t(\mathbf{p}))\log\frac{1 + t(\mathbf{p})}{1 + y(\mathbf{p})} + (1 - t(\mathbf{p}))\log\frac{1 - t(\mathbf{p})}{1 - y(\mathbf{p})}\right],$$

where the sum runs over all input patterns. For the hyperbolic tangent function $\sigma_o(x) = \tanh x$ the values of H are always finite and the gradient computation in the BP algorithm gets the simplest possible form. In fact, this function is the rescaled version $\tanh x = \sigma_h(2x)$ of the bipolar activation function σ_h used in our previous neural network training.

The following lemma, whose proof is given in Appendix A, is the foundation for comparing the values of the error functions E and H.

Lemma 4. *Consider the half-sum-of-squares difference*

$$E(\mathbf{t}, \mathbf{y}) = \frac{1}{2}\sum_{\alpha=1}^{P}(t_\alpha - y_\alpha)^2$$

and the relative entropy measure

$$H(\mathbf{t}, \mathbf{y}) = \frac{1}{2}\sum_{\alpha=1}^{P}\left[(1 + t_\alpha)\log\frac{1 + t_\alpha}{1 + y_\alpha} + (1 - t_\alpha)\log\frac{1 - t_\alpha}{1 - y_\alpha}\right]$$

for arbitrary sequences $\mathbf{t} = (t_1, t_2, \ldots, t_P)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_P)$ with $|t_\alpha| = 1$ and $y_\alpha \in [-1, 1]$ for $\alpha = 1, 2, \ldots, P$. The values of these error functions E and H satisfy the relations

$$E = O(H^2) \qquad \text{for } H \to 0 \tag{17}$$

and

$$H = O(\sqrt{E}) \qquad \text{for } E \to 0. \tag{18}$$

It follows from the example

$$\mathbf{t} = (1, 1), \quad \mathbf{y} = (1, -1),$$

Table 2
Simulation results for alternative error and activation functions with the backpropagation algorithm (BP) and backpropagation with weight extrapolations (BPWE)

		Training of type A	Training of type B	Training of type C
XOR problem with BP	Epochs required	1135	447	137
	Learning time (s)	0.34	0.21	0.17
	Total error of solution	0.01	0.0001	0.00009
XOR problem with BPWE	Epochs required	554	336	95
	Learning time (s)	0.24	0.19	0.16
	Total error of solution	0.01	0.00008	0.00009
	Number of extrapolations	2	3	1
Two spirals problem with BP	Epochs until solution	7671	503 077	431 742
	Learning time (s)	56.99	2598.29	2206.25
	Total error of solution	0.01	0.0001	0.0001
Two spirals problem with BPWE	Epochs required	3854	432 015	227 162
	Learning time (s)	21.06	2230.57	1178.30
	Total error of solution	0.01	0.0001	0.00008
	Number of extrapolations	3	7	9
L–T problem with BP	Epochs until solution	970	220	186
	Learning time (s)	0.43	0.19	0.18
	Total error of solution	0.003	0.00001	0.00001
L–T problem with BPWE	Epochs required	252	150	143
	Learning time (s)	0.23	0.17	0.17
	Total error of solution	0.002	0.00001	0.000002
	Number of extrapolations	2	1	1

where $E(\mathbf{t}, \mathbf{y}) = 2$ and $H(\mathbf{t}, \mathbf{y}) = \infty$, together with the proof of Lemma 4 that the error measures H^2 and E are "equivalent" only for $H \in [0, D]$ and $E \in [0, 2 - \epsilon]$ if D, $\epsilon > 0$ are fixed constants.

5.4. Training with alternative error and activation functions

In order to demonstrate the flexibility of BPWE, the examples of Section 5.1 were solved by the BP and BPWE algorithms with different training conditions.

Training of type A. First, the half-sum-of-squares error function E was replaced by the relative entropy error function H specified in Section 5.3. According to the discussion in the previous section, the logistic activation function in the output nodes of the respective networks was replaced by the function $\sigma_o(x) = \tanh x$ and the target values were modified to 1 and -1. To achieve convergence of the BP training algorithm to an acceptable solution, the learning rate η had to be decreased from 0.9 to 0.1 for the XOR and the L–T problem and to 0.01 for the two spirals problem. We call the resulting training processes training of type A. Here, the final error value 0.01 was used for the objective function H of the XOR problem and the two spirals problem according to Lemma 4. Accordingly, the training for the LT problem was terminated as soon as H fell below 0.003.

Training of type B. During training of type B the activation function in the output layer has been removed completely. This has the effect that the BP algorithm again takes its simplest form mentioned in Section 5.3. We used the same learning rates as in the training of type A, only for the two spirals problem $\eta = 0.3$ has been selected.

Training of type C. The only difference between type C and type B consisted in the activation function σ_h in the hidden layer of the neural network, which was now chosen as the hyperbolic tangent function used already in type A training. This activation function is attractive from a practical point of view (Bishop, 1995). The learning rate was chosen as 0.1 for all three examples.

The simulation results for all nine training situations are given in Table 2 for both BP and BPWE algorithm. Although the BP algorithm converges extremely fast with these modified network structures and error functions for the XOR and the L–T problem (only a few hundred epochs are required to reach the stopping criteria), the BPWE procedure still managed to decrease the number of iterations considerably. In addition, a decrease in the CPU time and in half of the cases also in the final error of the training process could be achieved. This demonstrates the robustness of the extrapolation idea at least for the BP algorithm used with feedforward neural networks even with the results for the two spirals problem not fully living up to the expectations.

However, it is worth mentioning that during BPWE training of type B for the XOR problem the second extrapolation resulted in an increase of the error function. This is in contrast to what is expected of the BPWE algorithm, but did not have any negative effect on this training example.

5.5. Discussion

It can be recognized from all these simulation results that the improvement obtained with BPWE is more remarkable on complex problems on which the BP scheme gives very slow convergence than on simple problems. The reason

seems to be that when the training time is long there are more possibilities for applying the weight extrapolation procedure than when the training time is short. The Euclidean distances between the final weight vectors obtained by the BP algorithm and BPWE algorithm are 0.27, 0.03 and 0.08 for the three example problems in Section 5.1. Similar values were obtained for the training experiments in Section 5.4. Even for the example where the error function increased through an extrapolation step, the Euclidean distance between the final weight vectors of BP and BPWE was just 0.77. These small values indicate that with a few extrapolations between the iterations of the BP algorithm, the convergence rate is increased without altering the convergence path. Other proposed improvements such as the CG method differ significantly from the original BP algorithm, and consequently the path, which the sequence $(\mathbf{w}_k)$ follows in the weight space usually also differs significantly.

6. Conclusions

In this paper, we focused on improving the BP algorithm for training feedforward neural networks. We introduced a new speed-up method called BPWE for the standard BP algorithm based on the concept of extrapolating computed network weights. By extrapolating the weights, it is possible to economize on the epochs required by BP learning before an acceptable weight vector is reached. Furthermore, the experimental results with the BPWE algorithm show that this algorithm offers much higher speed of convergence than the basic BP algorithm. Consequently, the improvement presented in this paper can be considered as a valuable and viable alternative to existing training methods. The degree of improvement with BPWE for difficult problems (for which the BP algorithm converges slowly) is greater than that for simple problems (for which the BP algorithm converges quickly). According to the simulation experiments described in the preceding section, the CPU time as well as the number of epochs required by the BP training process can be reduced by a factor of 3 for complex problems for which the BP algorithm converges slowly. In Section 4 we introduced the CG algorithm as another alternative to BP learning, and demonstrated how it can be employed for training feedforward neural networks by appropriately selecting the learning and momentum rate during each iteration. It turned out that modifying an existing BP training procedure to get the CG method is not a significant task. However, the line search involved must be considered more sophisticated than the extensions to BP arising from our extrapolation procedure based on the well-known formulas for linear regression that do not need additional evaluations of the error function or its derivative. Moreover, concerning the quality of the solutions, the simulation experiments described in Section 5 reveal tremendous advantages of the BPWE method. The proposed extrapolation method is very simple, effective, and contains no problem-dependent parameters. Furthermore, it has the flexibility to be used also for other iterative learning algorithms and/or network architectures without modifications. Since our work is complementary to other efforts, consideration of the proposed improved BP algorithm with any other variant is possible.

Appendix A

Proof of Proposition 2. *Step* 1. First we want to show that the function E has a single local minimum which is also its global minimum if $A_i > 0$ for $i = 1, 2, \ldots, n$. It can be shown using elementary calculus that every function p_i has a unique global minimum at $x = -B_i/(2A_i)$. Consequently, the vector $\mathbf{w}_* := (-B_1/(2A_1), -B_2/(2A_2), \ldots, -B_n/(2A_n))$ constitutes a unique global minimum for E.

Step 2. The converse of the statement in step 1 can be shown indirectly. Suppose that E has an isolated local minimum $\mathbf{w}_* = (w_*^1, w_*^2, \ldots, w_*^n)$ and $A_{i_0} \leq 0$ for some i_0 with $1 \leq i_0 \leq n$. It follows that p_{i_0} does not have an isolated local minimum. Consequently, for every $\epsilon > 0$ there exists a value $w_\epsilon^{i_0}$, different from $w_*^{i_0}$ with $|w_\epsilon^{i_0} - w_*^{i_0}| < \epsilon$ and $p_{i_0}(w_\epsilon^{i_0}) \leq p_{i_0}(w_*^{i_0})$, which leads to the contradiction

$$E(w_*^1, w_*^2, \ldots, w_*^{i_0-1}, w_\epsilon^{i_0}, w_*^{i_0+1}, \ldots, w_*^n) \leq E(\mathbf{w}_*).$$

Step 3. The convergence behavior of BP and BPWE learning is revealed through the consideration of every individual component w^i of the weight vectors $\mathbf{w}$. By using the more convenient representation

$$p_i(w^i) = A_i\left(w^i + \frac{B_i}{2A_i}\right)^2 + C_i - \frac{B_i^2}{4A_i}$$

of the corresponding polynomial p_i and the relations (1) and (2), it turns out that the development of the sequence (w_k^i) is determined by

$$\begin{aligned} w_{k+1}^i - w_*^i &= w_k^i - 2A_i\eta(w_k^i - w_*^i) - w_*^i \\ &= (w_k^i - w_*^i)(1 - 2A_i\eta). \end{aligned} \tag{A1}$$

This recursion leads to

$$|w_k^i - w_*^i| = |w_0^i - w_*^i| \cdot |1 - 2A_i\eta|^k,$$

so that every single weight sequence (w_k^i) converges to the optimum $w_*^i = -B_i/2A_i$ if and only if $w_0^i = w_*^i$, or $w_0^i \neq w_*^i$ and

$$|1 - 2A_i\eta| < 1. \tag{A2}$$

This is equivalent to

$$\chi_I(i)\eta < A_i^{-1}$$

because of $A_i > 0$, where $\chi_I(i) = 1$ if $i \in I$ and 0 otherwise. Since the BP algorithm converges only if all its weights converge, a necessary and sufficient convergence condition for BP has the form

$$\eta < \min_{i \in I} A_i^{-1} = (\max_{i \in I} A_i)^{-1} = L.$$

For the rest of the proof, we assume that η satisfies this convergence condition.

Step 4. During this step of the proof, we want to derive the rate of convergence and the number of epochs until the minimum $\mathbf{w}_*$ is reached by the BP algorithm. In the case where condition (15) of Proposition 2 is satisfied, BP reaches the minimum in one epoch, because we have with Eq. (A1) that

$$w_1^i - w_*^i = (w_0^i - w_*^i)(1 - 2A_i\eta) = 0$$

for each index i, where $w_0^i - w_*^i = 0$ for $i \notin I$ and $1 - 2A_i\eta = 0$ for $i \in I$.

On the contrary, if condition (15) is not valid, we define the nonempty set J by $J := \{i \in I : 1/(2A_i) \neq \eta\}$, a positive constant $D_1 := \min_{i\in J} |1 - 2A_i\eta|$, and a constant $D_2 := \max_{i\in I} |1 - 2A_i\eta|$ with $D_2 < 1$ according to inequality (A2). The use of relation (A1) now yields

$$\begin{aligned} D_1\|\mathbf{w}_k - \mathbf{w}_*\|_\infty &= D_1 \cdot \max_{i\in I} |w_k^i - w_*^i| \\ &\leq \max_{i\in I} (|w_k^i - w_*^i| \cdot |1 - 2A_i\eta|) = \max_{i\in I} |w_{k+1}^i - w_*^i| \\ &= \|\mathbf{w}_{k+1} - \mathbf{w}_*\|_\infty \end{aligned}$$

and

$$\begin{aligned} D_2\|\mathbf{w}_k - \mathbf{w}_*\|_\infty &= D_2 \cdot \max_{i\in I} |w_k^i - w_*^i| \\ &\geq \max_{i\in I} (|w_k^i - w_*^i| \cdot |1 - 2A_i\eta|) = \max_{i\in I} |w_{k+1}^i - w_*^i| \\ &= \|\mathbf{w}_{k+1} - \mathbf{w}_*\|_\infty. \end{aligned}$$

In summary

$$D_1\|\mathbf{w}_k - \mathbf{w}_*\|_\infty \leq \|\mathbf{w}_{k+1} - \mathbf{w}_*\|_\infty \leq D_2\|\mathbf{w}_k - \mathbf{w}_*\|_\infty$$

for every $k \in \mathbb{N}_0$ with constants D_1 and D_2 that satisfy $0 < D_1 \leq D_2 < 1$, which expresses the linear convergence of the BP algorithm.

Step 5. The statements of Proposition 2 concerning the convergence of the BPWE method remain to be shown. From step 3 we conclude that extrapolations are not activated if

$$1 - 2A_i\eta < 0,$$

i.e. if $\eta > 1/2A_i$ for some index i. The reason is that the values of the corresponding sequence (w_k^i) oscillate around the optimum w_*^i. Therefore, the strict monotonicity demanded in item 1 of Definition 1 is never valid for this weight component. This means that for $\eta > L/2$ the BPWE routine performs simply the BP algorithm. In the case when property (16) of Proposition 2 is not satisfied or $\eta = L/2$, it follows from step 3 that a weight sequence (w_k^i) is constant.

For the remaining case we have $\eta < L/2$ and

$$I = \{1, 2, \ldots, n\}.$$

From step 3 we get

$$w_k^i = w_*^i + (w_0^i - w_*^i)(1 - 2A_i\eta)^k$$

for every index i, where the factors $w_0^i - w_*^i$ and $1 - 2A_i\eta$ are nonzero and positive, respectively. The development of w_k^i is therefore described by the function f with

$$f(k) := a - be^{-ck}, \quad a := w_*^i, \quad b := w_*^i - w_0^i,$$

$$c := -\log(1 - 2A_i\eta).$$

According to Section 3, an extrapolation of the computed weight vectors is activated after $M_1, M_1 + M_2, M_1 + M_2 + M_3, \ldots$ epochs. Every extrapolation predicts exactly the weights in the BP algorithm after $71M_1, 71(M_1 + M_2), 71(M_1 + M_2 + M_3), \ldots$ epochs. This leads to a faster convergence of the BPWE method compared with BP, where it holds for the speed-up factor s after eight epochs that

$$\frac{71\cdot\sum_{l=1}^{k} M_l + M_{k+1} - 1}{\sum_{l=1}^{k} M_l + M_{k+1} - 1} \leq s \leq \frac{71\cdot\sum_{l=1}^{k} M_l}{\sum_{l=1}^{k} M_l} \qquad (k \in \mathbb{N}).$$

With the relation $\sum_{l=1}^{k} M_l = 8(2^k - 1)$ it follows that the lower bound is equal to

$$\frac{568\cdot(2^k - 1) + 2^{k+3} - 1}{8\cdot(2^k - 1) + 2^{k+3} - 1} > \frac{576\cdot(2^k - 1)}{2^{k+4} - 9} \geq \frac{576}{23} > 25.$$

Therefore s is contained in the interval (25, 71], and depends on the epoch after which BPWE is stopped. □

Proof of Lemma 4. Let $\mathbf{t}$ and $\mathbf{y}$ be two sequences as in Lemma 4 and let E and H be the corresponding error measures.

Step 1. To prove relation (17) we consider an arbitrary index α from the interval $[1, P]$, and consider the value H to be, e.g. less or equal to 1.

Let us first assume that $t_\alpha = 1$. Then it follows that

$$\log \frac{2}{1 + y_\alpha} \leq H,$$

or equivalently

$$y_\alpha \geq \frac{2}{e^H} - 1$$

and

$$\frac{1}{2}(t_\alpha - y_\alpha)^2 \leq 2\left(1 - \frac{1}{e^H}\right)^2. \tag{A3}$$

If conversely $t_\alpha = -1$, then we get

$$\log \frac{2}{1 - y_\alpha} \leq H$$

and inequality (A3) follows with an argumentation similar to the one above.

Using the inequality

$$e^x \leq 1 + (e-1)x \qquad \text{for } x \in [0,\ 1],$$

which is an exercise in basic calculus, for relation (A3), it turns out that

$$\frac{1}{2}(t_\alpha - y_\alpha)^2 \leq 2\left(\frac{(e-1)H}{1+(e-1)H}\right)^2 \leq 2(e-1)^2H^2$$

for every index α, and in summary

$$E \leq 2P(e-1)^2H^2.$$

Step 2. For the proof of relation (18) we assume, e.g. that $E \leq 1$ and let α be an arbitrary index. If $t_\alpha = 1$, then

$$\tfrac{1}{2}(1-y_\alpha)^2 \leq E,$$

which is equivalent to

$$y_\alpha \geq 1 - \sqrt{2E} \Leftrightarrow \log\frac{2}{1+y_\alpha} \leq \log\frac{2}{2-\sqrt{2E}}.$$

The analogous inequality

$$\log\frac{2}{1-y_\alpha} \leq \log\frac{2}{2-\sqrt{2E}}$$

follows for $t_\alpha = -1$ and it implies with the well-known inequality

$$\log x \leq x - 1 \qquad \text{for } x \in \mathbb{R}^+$$

that

$$\log\frac{2}{1+\operatorname{sgn}(t_\alpha)y_\alpha} \leq \frac{\sqrt{2E}}{2-\sqrt{2E}} \leq \frac{1}{\sqrt{2}-1}\sqrt{E}$$

for $\alpha = 1, 2, \ldots, P$. This leads to

$$H \leq \frac{P}{\sqrt{2}-1}\sqrt{E},$$

which concludes the proof. □

References

Balakrishnan, K., & Honavar, V. G. (1992). Improving convergence of back propagation by handling flat-spots in the output layer. In I. Aleksander & J. Taylor (Eds.), (pp. 1003–1009). *Artificial neural network*, 2. North-Holland: Elsevier.

Battiti, R. (1992). First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, *4* (2), 141–166.

Baum, E. B., & Lang, K. J. (1991). Constructing hidden units using examples and queries. In R. P. Lippmann & J. E. Moody & D. S. Touretzky (Eds.), (pp. 904–910). *Advances in neural information processing systems*, 3. San Mateo, CA: Morgan Kaufmann.

Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (1993). *Nonlinear programming—theory and algorithms*, New York: Wiley.

Bishop, C. M. (1995). *Neural networks for pattern recognition*, Oxford: Clarendon Press.

Brent, R. P. (1973). *Algorithms for minimization without derivatives*, Englewood Cliffs, NJ: Prentice-Hall.

Cater, J. P. (1987). Successfully using peak learning rates of 10 (and greater) in back-propagation networks with the heuristic learning algorithm. (pp. 645–651). *Proceedings of the IEEE International Conference on Neural Networks*, 2. San Diego, CA: Institute of Electrical and Electronic Engineers.

Charalambous, C. (1992). Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings-G*, *139* (3), 301–310.

Chen, T., Chen, H., & Liu, R. -W. (1995). Approximation capability in $C(\bar{\mathbb{R}}^n)$ by multilayer feedforward networks and related problems. *IEEE Transactions on Neural Networks*, *6* (1), 25–30.

Cho, S. -B., & Kim, J. H. (1990). An accelerated learning method with backpropagation. (pp. 605–608). *Proceedings of the International Joint Conference on Neural Networks*, 1. Washington, DC: Lawrence Erlbaum.

Codrington, C. W., & Mohandes, M. (1994). Projection-based methods for stepsize adaptation and their application to the training of feedforward artificial neural networks. (pp. 72–77). *Proceedings of the IEEE International Conference on Neural Networks*, 1. Orlando, FL: Institute of Electrical and Electronic Engineers.

Dahl, E. D. (1987). Accelerated learning using the generalized delta rule. (pp. 523–530). *Proceedings of the IEEE International Conference on Neural Networks*, 2. San Diego, CA: Institute of Electrical and Electronic Engineers.

Dennis, J. E., & Schnabel, R. B. (1983). *Numerical methods for unconstrained optimization and nonlinear equations*, Englewood Cliffs, NJ: Prentice-Hall.

Dewan, H. M., & Sontag, E. D. (1990). Extrapolatory methods for speeding up the BP algorithm. (pp. 613–616). *Proceedings of the International Joint Conference on Neural Networks*, 1. Washington, DC: Lawrence Erlbaum.

Fahlman, S. E. (1989). Faster-learning variations on back-propagation: an empirical study. *Proceedings of the 1988 Connectionist Models Summer School*, (pp. 38–51). Pittsburgh, PA: Morgan Kaufmann.

Fitch, J. P., Lehmann, S. K., Dowla, F. U., Lu, S. Y., Johansson, E. M., & Goodman, D. M. (1991). Ship wake-detection procedure using conjugate gradient trained artificial neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *29* (5), 718–726.

Gilbert, J. C., & Nocedal, J. (1992). Global convergence properties of conjugate gradient methods for optimization. *SIAM Journal on Optimization*, *2* (1), 21–42.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation*, Redwood City, CA: Addison-Wesley.

Hush, D. R., & Salas, J. M. (1988). Improving the learning rate of back-propagation with the gradient reuse algorithm. (pp. 441–447). *Proceedings of the IEEE International Conference on Neural Networks*, 1. San Diego, CA: Institute of Electrical and Electronic Engineers.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, *1* (4), 295–307.

Jelinek, F. (1968). *Probabilistic information theory—discrete and memoryless models*, New York: McGraw-Hill.

Jones, K. L., Lustig, I. J., & Kornhauser, A. L. (1990). Optimization techniques applied to neural networks: line search implementation for back propagation. (pp. 933–939). *Proceedings of the International Joint Conference on Neural Networks*, 3. Washington, DC: Lawrence Erlbaum.

Kanda, A., Fujita, S., & Ae, T. (1994). Acceleration by prediction for error back-propagation algorithm of neural network. *Systems and Computers in Japan*, *25* (1), 78–87.

Kramer, A. H., & Sangiovanni-Vincentelli, A. (1988). Efficient parallel learning algorithms for neural networks. In D. S. Touretzky (Ed.), (pp. 40–48). *Advances in neural information processing systems*, 1. San Mateo, CA: Morgan Kaufmann.

Lang, K. J., & Witbrock, M. J. (1989). Learning to tell two spirals apart. *Proceedings of the 1988 Connectionist Models Summer School*, (pp. 52–59). Pittsburgh, PA: Morgan Kaufmann.

Leonard, J., & Kramer, M. A. (1990). Improvement of the backpropagation algorithm for training neural networks. *Computers and Chemical Engineering*, *14* (3), 337–341.

Mhaskar, H. N., & Micchelli, C. H. (1992). Approximation by superposition of sigmoidal and radial basis functions. *Advances in Applied Mathematics*, *13* (3), 350–373.

Mohandes, M., Codrington, C. W., & Gelfand, S. B. (1994). Two adaptive stepsize rules for gradient descent and their application to the training of feedforward artificial neural networks. (pp. 555–560). *Proceedings of the IEEE International Conference on Neural Networks*, 1. Orlando, FL: Institute of Electrical and Electronic Engineers.

Nachtsheim, P. R. (1994). A first order adaptive learning rate algorithm for back propagation networks. (pp. 257–262). *Proceedings of the IEEE International Conference on Neural Networks*, 1. Orlando, FL: Institute of Electrical and Electronic Engineers.

Parekh, R., Balakrishnan, K., & Honavar, V. (1993). Empirical comparison of flat-spot elimination techniques in back-propagation networks. (pp. 55–60). *SPIE Proceedings (3rd Workshop on Neural Networks)*, 1721.

Pfister, M., & Rojas, R. (1993). Speeding-up backpropagation—a comparison of orthogonal techniques. (pp. 517–523). *Proceedings of the International Joint Conference on Neural Networks*, 1. Nagoya, Japan: Japanese Neural Network Society.

Pirez, Y. M., & Sarkar, D. (1993). Back-propagation algorithm with controlled oscillation of weights. (pp. 21–26). *Proceedings of the IEEE International Conference on Neural Networks*, 1. San Francisco, CA: Institute of Electrical and Electronic Engineers.

Pittner, S. (1996). *Basics for conjugate gradient minimum search*. Technical Report, Northeastern University, Boston, USA.

Polak, E. (1971). *Computational methods in optimization: a unified approach*, New York: Academic Press.

Polak, E., & Ribière, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, *16*, 35–43.

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, *12*, 241–254.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical recipes in C: the art of scientific computing*, 2. Cambridge: Cambridge University Press.

Robitaille, B., Marcos, B., Veillette, M., & Payre, G. (1996). Modified quasi-Newton methods for training neural networks. *Computers and Chemical Engineering*, *20* (9), 1133–1140.

Roy, S. (1993). Near-optimal dynamic learning rate for training backpropagation neural networks. In D. W. Ruck (Ed.), (pp. 277–283). *Science of Artificial Neural Networks*, II. Bellingham, WA: Society of Industrial and Applied Mathematics.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition: foundations*, 1. Cambridge, MA: MIT Press.

Ryan, T. P. (1997). *Modern regression methods*, New York: Wiley.

Salehi, F., Lacroix, R., & Wade, K. M. (1998). Effects of learning parameters and data presentation on the performance of backpropagation networks for milk yield prediction. *Transactions of the ASAE*, *41* (1), 253–259.

Salomon, R., & van Hemmen, J. L. (1996). Accelerating backpropagation through dynamic self-adaptation. *Neural Networks*, *9* (4), 589–601.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*, *1* (1), 145–168.

Silva, F. M., & Almeida, L. B. (1990). In L. B. Almeida & C. J. Wellekens (Eds.), *Acceleration techniques for the backpropagation algorithm*, (pp. 110–119). *Lecture Notes in Computer Science*, 412. Berlin: Springer.

Tesauro, G., He, Y., & Ahmad, S. (1989). Asymptotic convergence of backpropagation. *Neural Computation*, *1* (3), 382–391.

Towsey, M., Alpsan, D., & Sztriha, L. (1995). Training a neural network with conjugate gradient methods. (pp. 373–378). *Proceedings of the IEEE International Conference on Neural Networks*, 1. Perth, Australia: Institute of Electrical and Electronic Engineers.

Weir, M. K. (1991). A method for self-determination of adaptive learning rates in back propagation. *Neural Networks*, *4* (3), 371–379.

Yamada, K., Pecharanin, N., Taguchi, A., Iijima, N., & Sone, M. (1997). Escape from the slowdown area on training of back-propagation algorithm by using a jumping method. *Systems and Computers in Japan*, *28* (4), 48–57.

Yu, X., Loh, N. K., & Miller, W. C. (1993). A new acceleration technique for the backpropagation algorithm. (pp. 1157–1161). *Proceedings of the IEEE International Conference on Neural Networks*, 3. San Francisco, CA: Institute of Electrical and Electronic Engineers.