Neural Networks, Vol. 6, pp. 525–533, 1993
Printed in the USA. All rights reserved.

0893-6080/93 $6.00 + .00

ORIGINAL CONTRIBUTION

A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning

MARTIN FODSLETTE MØLLER

University of Aarhus

(*Received* 10 *January* 1991; *accepted* 13 *November* 1991)

Abstract—*A supervised learning algorithm (Scaled Conjugate Gradient, SCG) is introduced. The performance of SCG is benchmarked against that of the standard back propagation algorithm (BP) (Rumelhart, Hinton, & Williams,* 1986), *the conjugate gradient algorithm with line search (CGL) (Johansson, Dowla, & Goodman,* 1990) *and the one-step Broyden-Fletcher-Goldfarb-Shanno memoriless quasi-Newton algorithm (BFGS) (Battiti,* 1990). *SCG is fully-automated, includes no critical user-dependent parameters, and avoids a time consuming line search, which CGL and BFGS use in each iteration in order to determine an appropriate step size. Experiments show that SCG is considerably faster than BP, CGL, and BFGS.*

Keywords—Feedforward neural network, Supervised learning, Optimization, Conjugate gradient algorithms.

1. INTRODUCTION

1.1. Motivation

Several adaptive learning algorithms for feedforward neural networks have recently been discovered (Hinton, 1989). Many of these algorithms are based on the gradient descent algorithm well known in optimization theory. They usually have a poor convergence rate and depend on parameters which have to be specified by the user, as no theoretical basis for choosing them exists. The values of these parameters are often crucial for the success of the algorithm. An example is the standard back propagation algorithm (BP) (Rumelhart, Hinton, & Williams, 1986), which often behaves very badly on large-scale problems and whose success depends of the user dependent parameters *learning rate* and *momentum constant*. The aim of this paper is to develop a supervised learning algorithm that eliminates some of these disadvantages.

From an optimization point of view, learning in a neural network is equivalent to minimizing a global error function, which is a multivariate function that depends on the weights in the network. This perspective gives some advantages in the development of effective learning algorithms because the problem of minimizing a function is well known in other fields of science, such as conventional numerical analysis (Watrous, 1987).

Since learning in realistic neural network applications often involves adjustment of several thousand weights, only optimization methods that are applicable to large-scale problems are relevant as alternative learning algorithms. The general opinion in the numerical analysis community is that especially one class of optimization methods, called the *Conjugate Gradient Methods*, are well suited to handle large-scale problems in an effective way (Fletcher, 1975; Gill, Murray, & Wright, 1980; Hestenes, 1980; Powell, 1977). Several conjugate gradient algorithms have recently been introduced as learning algorithms in neural networks (Battiti, 1989; Johansson, Dowla, & Goodman, 1990; Møller, 1990). Johansson, Dowla, and Goodman describe the theory of general conjugate gradient methods and how to apply the methods in feedforward neural networks. They conclude that the standard conjugate gradient method with line search (CGL) is an order of magnitude faster than the standard BP when tested on the parity problem. Battiti and Masulli have introduced

Acknowledgment: I would like to thank Brian Mayoh (AAU), Kim Plunkett (University of Oxford), and Ole Østerby (AAU) for many good discussions and advice. The work was supported by Statens Teknisk-Videnskabelige Forkningsråd, Denmark and The Carlsberg Foundation, Denmark.

Requests for reprints should be sent to Martin F. Møller, Computer Science Department, University of Aarhus, Ny Munkegade, Building 540, Denmark.

a variation of the standard conjugate gradient method, the one-step Broyden-Fletcher-Goldfarb-Shanno memoriless quasi-Newton algorithm (BFGS), as an alternative learning algorithm (Battiti & Masulli, 1990). They conclude that BFGS also yields an acceleration of about one order of magnitude compared to BP when tested on the parity problem. Both CGL and BFGS raise the calculation complexity per learning iteration considerably since they have to perform a line search in order to determine an appropriate step size. A line search involves several calculations of either the global error function or its derivative, both of which raise the complexity.

This paper introduces a new variation of the conjugate gradient method (Scaled Conjugate Gradient, SCG), which avoids the line search per learning iteration by using a Levenberg-Marquardt approach (Gill, Murray, & Wright, 1980) in order to scale the step size. During the development of SCG, a tutorial to the theory of conjugate gradient related algorithms is given.

1.2. Notation

Let an arbitrary feedforward neural network be given. The weights in the network will be expressed in vector notation. A *weight vector* is a vector in the real euclidean space $\mathcal{R}^N$, where N is the number of weights and biases in the network. A weight vector will often be referred to as a point in $\mathcal{R}^N$ or just a point in weight space. Let $\tilde{w}$ be the weight vector defined by

$$\tilde{w} = (\ldots, w_{ij}^{(l)}, w_{i+1j}^{(l)}, \ldots, w_{N_lj}^{(l)}, \theta_j^{(l+1)}, w_{ij+1}^{(l)}, w_{i+1j+1}^{(l)}, \ldots), \tag{1}$$

where $w_{ij}^{(l)}$ is the weight from unit number i in layer number 1 to unit number j in layer number $l + 1$, N_l is the number of units in layer 1, and $\theta_j^{(l+1)}$ is the bias for unit number j in layer number $l + 1$. We assume that a global error function $E(\tilde{w})$ depending on all the weights and biases is attached to the neural network. $E(\tilde{w})$ could be the standard least square function or any other appropriate error function. $E(\tilde{w})$ can be calculated with one forward pass and the gradient $E'(\tilde{w})$ with one forward and one backward pass (Rumelhart, Hinton, & Williams, 1986). According to Hinton (1989), it is reasonable to assume that the number of patterns to be learned is proportional to the number of weights.[1] Using this assumption the complexity of calculating $E(\tilde{w})$ and $E'(\tilde{w})$ is $O(N^2)$ and $O(3N^2)$, respectively, where N is the number of weights and biases (Yoshida, 1991). $E'(\tilde{w})$ is given by

$$E'(\tilde{w}) = \left(\ldots, \sum_{p=1}^{P} \frac{dE_p}{dw_{ij}^{(l)}}, \sum_{p=1}^{P} \frac{dE_p}{dw_{i+1j}^{(l)}}, \ldots, \sum_{p=1}^{P} \frac{dE_p}{dw_{N_lj}^{(l)}}, \sum_{p=1}^{P} \frac{dE_p}{d\theta_j^{(l+1)}}, \sum_{p=1}^{P} \frac{dE_p}{dw_{ij+1}^{(l)}}, \ldots\right), \tag{2}$$

where P is the number of patterns presented to the network during training and E_p is the error associated with pattern p.

We are now able to define some of the weight vector operations needed. The coordinates in a weight vector is referred to by superscript so that w^i denotes the i's weight in weight vector w. When matrix operations are used, a weight vector $\tilde{w}$ is a column-vector and w^T, the transpose of $\tilde{w}$, will then be a row-vector. The ordering of the coordinates in the weight vectors are not important as long as the chosen ordering is consistent throughout the implementation. Weight vectors and scalars are respectively indicated by Roman and Greek letters.

The *weight addition, weight subtraction,* and *weight product* are defined, respectively, as

$$\begin{aligned} \tilde{w} + \tilde{y} &= (w^1 + y^1, \ldots, w^i + y^i, \ldots, w^N + y^N)^T \\ \tilde{w} - \tilde{y} &= (w^1 - y^1, \ldots, w^i - y^i, \ldots, w^N - y^N)^T \\ \tilde{w}^T\tilde{y} &= \sum_{j=1}^{N} w^i y^i. \end{aligned} \tag{3}$$

The *weight length* is defined as

$$|\tilde{w}| = \left[\sum_{j=1}^{N} (w^i)^2\right]^{1/2}. \tag{4}$$

It might also be useful to recall that the error function $E(\tilde{w})$ in a given point $(\tilde{w} + \tilde{y})$ in $\mathcal{R}^N$ can be expressed by the well known Taylor expansion

$$E(\tilde{w} + \tilde{y}) = E(\tilde{w}) + E'(\tilde{w})^T\tilde{y} + \tfrac{1}{2}\tilde{y}^T E''(\tilde{w})\tilde{y} + \ldots. \tag{5}$$

A $N \times N$ matrix A is said to be *positive definite* if

$$\tilde{y}^T A \tilde{y} > 0 \quad \forall \tilde{y} \in \mathcal{R}^N. \tag{6}$$

Let $\tilde{p}_1, \ldots, \tilde{p}_k$ be a set of nonzero weight vectors in $\mathcal{R}^N$. The set is said to be a *conjugate system* with respect to a nonsingular symmetric $N \times N$ matrix A if the following holds (Hestenes, 1980)

$$\tilde{p}_i^T A \tilde{p}_j = 0 \quad (i \neq j, i = 1, \ldots, k). \tag{7}$$

The set of points $\tilde{w}$ in $\mathcal{R}^N$ satisfying

$$\tilde{w} = \tilde{w}_1 + \alpha_1\tilde{p}_1 + \ldots + \alpha_k\tilde{p}_k, \quad \alpha_i \in \mathcal{R}, \tag{8}$$

where $\tilde{w}_1$ is a point in weight space and $\tilde{p}_1, \ldots, \tilde{p}_k$ is a subset of a conjugate system, is called a *k-plane* or π_k (Hestenes, 1980).

2. OPTIMIZATION STRATEGY

Most of the optimization methods used to minimize functions are based on the same strategy. The mini-

[1] Throughout this paper, this will be assumed if not stated otherwise.

mization is a local iterative process in which an approximation to the function in a neighbourhood of the current point in weight space is minimized. The approximation is often given by a first or second order Taylor expansion of the function. The idea of the strategy is illustrated in the pseudo algorithm presented below, which minimizes the error function $E(\tilde{w})$.

1. Choose initial weight vector $\tilde{w}_1$ and set $k = 1$.
2. Determine a search direction $\tilde{p}_k$ and a step size α_k so that $E(\tilde{w}_k + \alpha_k\tilde{p}_k) < E(\tilde{w}_k)$.
3. Update vector: $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k\tilde{p}_k$.
4. If $E'(\tilde{w}_k) \neq \tilde{0}$ then set $k = k + 1$ and go to 2 else return $\tilde{w}_{k+1}$ as the desired minimum.

Determining the next current point in this iterative process involves two independent steps. First a *search direction* has to be determined, i.e., in what direction in weight space do we want to go in the search for a new current point. Once the search direction has been found we have to decide how far to go in the specified search direction, i.e., a *step size* has to be determined.

3. THE BACK PROPAGATION ALGORITHM

If the search direction $\tilde{p}_k$ in the above pseudo algorithm is set to the negative gradient $-E'(\tilde{w})$ and the step size α_k to a constant ε, then the algorithm becomes the gradient descent algorithm (Gill, Murray, & Wright, 1980). In the context of neural networks, this is the BP without a momentum term (Rumelhart, Hinton, & Williams, 1986). Minimization by gradient descent is based on the linear approximation $E(\tilde{w} + \tilde{y}) \approx E(\tilde{w}) + E'(\tilde{w})^T y$, which is the main reason why the algorithm often shows poor convergence. Another reason is that the algorithm uses a constant step size, which in many cases is inefficient and makes the algorithm less robust. The inclusion of a momentum term in the BP is an *ad hoc* attempt to force the algorithm to use second order information from the network. Unfortunately, the momentum term is not able to speed up the algorithm considerably, and causes the algorithm to be even less robust because of the inclusion of another user dependent parameter, *the momentum constant*. Back propagation, including the momentum term, will be referred to as BP.

Usually two versions of BP are considered, the "offline" version and the "online" version. They differ in how often the weights are updated. The "offline" version updates the weights after all patterns have been propagated through the network, i.e., using information from all the patterns in the training set. The "online" version updates after every single pattern, i.e., using only information from one pattern. The "online" version is not consistent with the optimization theory but it has nevertheless shown to be superior to the "offline" version on some specific problems. These problems seems to be characterized by big training sets containing a lot of redundant information (Le Cun, 1989) and by error surfaces that are not too complex in structure. The "offline" version is, however, superior on problems which does not have these properties.[2] This paper will use the "offline" version of BP in the comparison with other algorithms. For a more detailed discusssion and comparison with the "online" version of BP, see Møller (1991).

4. CONJUGATE DIRECTION METHODS

Conjugate direction methods are also based on the above general optimization strategy but choose the search direction and the step size more carefully by using information from the second order approximation $E(\tilde{w} + \tilde{y}) \approx E(\tilde{w}) + E'(\tilde{w})^T\tilde{y} + \frac{1}{2}y^T E''(\tilde{w})\tilde{y}$.

Quadratic functions have some advantages over general functions. Denote the quadratic approximation to E in a neighbourhood of a point $\tilde{w}$ by $E_{qw}(\tilde{y})$ so that $E_{qw}(\tilde{y})$ is given by

$$E_{qw}(\tilde{y}) = E(\tilde{w}) + E'(\tilde{w})^T\tilde{y} + \tfrac{1}{2}\tilde{y}^T E''(\tilde{w})\tilde{y}. \quad (9)$$

In order to determine minima to $E_{qw}(\tilde{y})$ the critical points for $E_{qw}(\tilde{y})$ must be found, i.e., the points where

$$E'_{qw}(\tilde{y}) = E''(\tilde{w})\tilde{y} + E'(\tilde{w}) = 0. \quad (10)$$

The critical points are the solution to the linear system defined by eqn (10). If a conjugate system is available, the solution can be simplified considerably (Hestenes, 1980). Johansson, Dowla, and Goodman (1990) show this in a very understandable manner. Let $\tilde{p}_1, \ldots, \tilde{p}_N$ be a conjugate system. Because $\tilde{p}_1, \ldots, \tilde{p}_N$ form a basis for $\mathcal{R}^N$, the step from a starting point $\tilde{y}_1$ to a critical point $\tilde{y}_*$ can be expressed as a linear combination of $\tilde{p}_1, \ldots, \tilde{p}_N$

$$\tilde{y}_* - \tilde{y}_1 = \sum_{i=1}^{N} \alpha_i\tilde{p}_i,\ \alpha_i \in \mathcal{R}. \quad (11)$$

Multiplying eqn (11) with $\tilde{p}_j^T E''(\tilde{w})$ and substituting $E'(\tilde{w})$ for $-E''(\tilde{w})\tilde{y}_*$ gives

$$\tilde{p}_j^T(-E'(\tilde{w}) - E''(\tilde{w})\tilde{y}_1) = \alpha_j\tilde{p}_j^T E''(\tilde{w})\tilde{p}_j \Rightarrow$$
$$\alpha_j = \tilde{p}_j^T(-E'(\tilde{w}) - E''(\tilde{w})\tilde{y}_1)/\tilde{p}_j^T E''(\tilde{w})\tilde{p}_j$$
$$= -\tilde{p}_j^T E'_{qw}(\tilde{y}_1)/\tilde{p}_j^T E''(\tilde{w})\tilde{p}_j. \quad (12)$$

The critical point $\tilde{y}_*$ can be determined in N iterative steps using eqns (11) and (12). Unfortunately, $\tilde{y}_*$ is not necessarily a minimum, but can be a saddle point or a maximum. Only if the Hessian matrix $E''(\tilde{w})$ is positive definite then $E_{qw}(y)$ has a unique global minimum (Hestenes, 1980). This can be realized by

[2] Such as the parity problem which is used in this paper as a benchmark problem.

$$\begin{aligned}
E_{qw}(y) &= E_{qw}(\tilde{y}_* + (\tilde{y} - \tilde{y}_*)) \\
&= E(\tilde{w}) + E'(\tilde{w})^T(\tilde{y}_* + (\tilde{y} - \tilde{y}_*)) \\
&\quad + \tfrac{1}{2}(\tilde{y}_* + (\tilde{y} - \tilde{y}_*))^T E''(\tilde{w})(\tilde{y}_* + (\tilde{y} - \tilde{y}_*)) \\
&= E(\tilde{w}) + E'(\tilde{w})^T\tilde{y}_* + E'(\tilde{w})^T(\tilde{y} - \tilde{y}_*) \\
&\quad + \tfrac{1}{2}y_*^T E''(\tilde{w})\tilde{y}_* + \tfrac{1}{2}y_*^T E''(\tilde{w})(\tilde{y} - \tilde{y}_*) \\
&\quad + \tfrac{1}{2}(\tilde{y} - \tilde{y}_*)^T E''(\tilde{w})\tilde{y}_* \\
&\quad + \tfrac{1}{2}(\tilde{y} - \tilde{y}_*)^T E''(\tilde{w})(\tilde{y} - \tilde{y}_*) \\
&=^3 E_{qw}(\tilde{y}_*) + (\tilde{y} - \tilde{y}_*)^T(E''(\tilde{w})\tilde{y}_* + E'(\tilde{w})) \\
&\quad + \tfrac{1}{2}(\tilde{y} - \tilde{y}_*)^T E''(\tilde{w})(\tilde{y} - \tilde{y}_*) \\
&=^4 E_{qw}(\tilde{y}_*) + \tfrac{1}{2}(\tilde{y} - \tilde{y}_*)^T E''(\tilde{w})(\tilde{y} - \tilde{y}_*). \qquad (13)
\end{aligned}$$

It follows from eqn (13) that if $\tilde{y}_*$ is a minimum, then $\frac{1}{2}(\tilde{y} - \tilde{y}_*)^T E''(\tilde{w})(\tilde{y} - \tilde{y}_*) > 0$ for every $\tilde{y}$, hence $E''(\tilde{w})$ has to be positive definite. The Hessian $E''(\tilde{w})$ in the following will be assumed to be positive definite, if not otherwise stated.

The intermediate points $\tilde{y}_{k+1} = \tilde{y}_k + \alpha_k\tilde{p}_k$ given by the iterative determination of $\tilde{y}_*$ are in fact minima for $E_{qw}(y)$ restricted to every k-plane π_k: $\tilde{y} = \tilde{y}_1 + \alpha_1\tilde{p}_1 + \ldots + \alpha_k\tilde{p}_k$. How to determine these points recursively is shown in the following theorem. Its proof can be found in Hestenes (1980).

THEOREM 1. *Let $\tilde{p}_1, \ldots, \tilde{p}_N$ be a conjugate system and $\tilde{y}_1$ a point in weight space. Let the points $\tilde{y}_2, \ldots, \tilde{y}_{N+1}$ be recursively defined by*

$$\tilde{y}_{k+1} = \tilde{y}_k + \alpha_k\tilde{p}_k,$$

where $\alpha_k = \mu_k/\delta_k$, $\mu_k = -\tilde{p}_k^T E'_{qw}(\tilde{y}_k)$, $\delta_k = \tilde{p}_k^T E''(\tilde{w})\tilde{p}_k$. Then $\tilde{y}_{k+1}$ minimizes E_{qw} restricted to the k-plane π_k given by $\tilde{y}_1$ and $\tilde{p}_1, \ldots, \tilde{p}_k$.

The conjugate direction algorithm as proposed by Hestenes (1980) can be formulated as follows. Select an initial weight vector $\tilde{y}_1$ and a conjugate system $\tilde{p}_1, \ldots, \tilde{p}_N$. Find successive minima for E_{qw} on the planes $\pi_1, \ldots, \pi_N$ using Theorem 1, where π_k, $1 \le k \le N$, is given by $\tilde{y} = \tilde{y}_1 + \alpha_1\tilde{p}_1 + \ldots + \alpha_k\tilde{p}_k$, $\alpha_i \in \mathcal{R}$. The algorithm assures that the global minimum for a quadratic function is detected in, at most, N iterations. If all the eigenvalues of the Hessian $E''(\tilde{w})$ fall into multiple groups with values of the same size, then there is a great probability that the algorithm terminates in much less than N iterations. Practice shows that this is often the case (Fletcher, 1975).

4.1. Conjugate Gradients

The conjugate direction algorithm above assumes that a conjugate system is given. But how does one determine such a system? It is not necessary to know the conjugate weight vectors $\tilde{p}_1, \ldots, \tilde{p}_N$ in advance as they can be determined recursively. Initially, $\tilde{p}_1$ is set to the steepest descent vector $-E'_{qw}(\tilde{y}_1)$. Then $\tilde{p}_{k+1}$ is determined recursively as a linear combination of the current steepest descent vector $-E'_{qw}(\tilde{y}_{k+1})$ and the previous direction $\tilde{p}_k$. More precisely, $\tilde{p}_{k+1}$ is chosen as the orthogonal projection of $-E'_{qw}(\tilde{y}_{k+1})$ on the $(N-k)$-plane π_{N-k} conjugate to π_k. Theorem 2, given in Hestenes (1980), shows how this can be done.

THEOREM 2. *Let $\tilde{y}_1$ be a point in weight space and $\tilde{p}_1$ and $\tilde{r}_1$ equal to the steepest descent vector $-E'_{qw}(\tilde{y}_1)$. Define $\tilde{p}_{k+1}$ recursively by*

$$\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k\tilde{p}_k$$

where $\tilde{r}_{k+1} = E'_{qw}(\tilde{y}_{k+1})$, $\beta_k = (|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T\tilde{r}_k)/\tilde{p}_k^T\tilde{r}_k$ and $\tilde{y}_{k+1}$ is the point generated in Theorem 1. Then $\tilde{p}_{k+1}$ is the steepest descent vector to E_{qw} restricted to the $(N-k)$-plane π_{N-k} conjugate to π_k given by $\tilde{y}_1$ and $\tilde{p}_1, \ldots, \tilde{p}_k$.

The conjugate vectors obtained using Theorem 2 are often referred to as *conjugate gradient directions.* Combining Theorem 1 and Theorem 2 we get a *conjugate gradient algorithm.* In each iteration this algorithm can be applied to the quadratic approximation E_{qw} of the global error function E in the current point $\tilde{w}$ in weight space. Because the error function $E(\tilde{w})$ is nonquadratic, the algorithm will not necessarily converge in N steps. If the algorithm has not converged after N steps, the algorithm is restarted, i.e., initializing $\tilde{p}_{k+1}$ to the current steepest descent direction $\tilde{r}_{k+1}$ (Fletcher, 1975; Powell, 1977). This also means that Theorems 1 and 2 are only valid in the ideal case when the error E is equal to the quadratic approximation E_{qw}. This is, of course, not often the case but it does hold that the nearer the current point is to the minimum the better is the quadratic approximation E_{qw} of the error E. This property is, in practice, adequate to give a fast convergence. A standard CG can now be described as follows.

```
1. Choose initial weight vector $\tilde{w}_1$.
   Set $\tilde{p}_1 = \tilde{r}_1 = -E'(\tilde{w}_1)$, $k = 1$.
2. Calculate second order information:
      $\tilde{s}_k = E''(\tilde{w}_k)\tilde{p}_k$,
      $\delta_k = \tilde{p}_k^T\tilde{s}_k$.
3. Calculate step size:
      $\mu_k = \tilde{p}_k^T\tilde{r}_k$,
      $\alpha_k = \mu_k/\delta_k$.
4. Update weight vector:
      $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k\tilde{p}_k$,
      $\tilde{r}_{k+1} = -E'(\tilde{w}_{k+1})$.
5. If $k$ mod $N = 0$ then restart algorithm: $\tilde{p}_{k+1} = \tilde{r}_{k+1}$
   else create new conjugate direction:
      $\beta_k = (|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T\tilde{r}_k)/\mu_k$,
      $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k\tilde{p}_k$.
```

[3] $E''(\tilde{w})$ is symmetric.

[4] $E''(\tilde{w})\tilde{y}_* + E'(\tilde{w}) = 0$ by eqn (10).

6. If the steepest descent direction $\tilde{r}_k \neq \tilde{0}$ then set $k = k + 1$ and go to 2
   else terminate and return $\tilde{w}_{k+1}$ as the desired minimum.

Several other formulas for β_k can be derived (Fletcher, 1975; Gill, Murray, & Wright, 1980; Hestenes, 1980), but when the conjugate gradient methods are applied to nonquadratic functions, the above formula, called the Hestenes-Stiefel formula, for β_k is considered superior. When the algorithm shows poor development, the formula forces the algorithm to restart because of the following relation

$$\tilde{r}_{k+1} \approx \tilde{r}_k \Rightarrow \beta_k \approx 0 \Rightarrow \tilde{p}_{k+1} \approx \tilde{r}_{k+1}. \qquad (14)$$

For each iteration in CG the Hessian matrix $E''(\tilde{w}_k)$ has to be calculated and stored. It is not desirable to calculate the Hessian matrix explicitly because of the calculation complexity and memory usage involved; actually, calculating the Hessian would demand $O(N^2)$ memory usage and $O(N^3)$ in calculation complexity. Usually this problem is solved by approximating the step size with a line search. Using the fact that $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k\tilde{p}_k$ is a minimum for the k-plane $\tilde{p}_1, \ldots, \tilde{p}_k$ (by Theorem 1) it is possible to show that

$$E'(\tilde{w}_{k+1})\tilde{p}_k = 0. \qquad (15)$$

Equation (15) shows that α_k is the solution to

$$\min_\alpha E(\tilde{w}_k + \alpha\tilde{p}_k). \qquad (16)$$

So α_k is the minimum for E along the line $\tilde{w}_k + \alpha\tilde{p}_k$. α_k is in fact only an approximated solution to eqn (16) since E is nonquadratic. The techniques for solving eqn (16) are known as line-search techniques (Gill, Murray, & Wright, 1980). The Appendix gives a description of the line-search algorithm used in this paper. All line-search techniques include at least one user dependent parameter which determine when the line search should terminate. The value of this parameter is often crucial for the success of the line search.

4.2. The CGL Algorithm

The CG shown above is often used with line search. That means the step size is approximated with a line-search technique, avoiding the calculation of the Hessian matrix. Johansson, Dowla, and Goodman (1990) used this scheme using a cubic interpolation algorithm. We use the CG combined with the safeguarded quadratic univariate minimization described in the Appendix. This algorithm will be referred to as CGL.

4.3. The BFGS Algorithm

Battiti and Massuli have proposed another method from the optimization literature known as the BFGS (Battiti, 1990). The algorithm is also based on conjugate directions combined with line search. The direction is updated by the following rule:

$$\tilde{p}_k = S_k\tilde{r}_k + A_k\tilde{y}_k + S_kB_k\tilde{q}_k, \qquad (17)$$

where $\tilde{r}_k = -E'(\tilde{w}_k)$, $\tilde{y}_k = \tilde{w}_k - \tilde{w}_{k-1}$ and $\tilde{q}_k = E'(\tilde{w}_k) - E'(\tilde{w}_{k-1})$. The coefficients S_k, A_k, and B_k are defined as:

$$A_k = -\left(1 + S_k\frac{\tilde{q}_k^T\tilde{q}_k}{\tilde{y}_k^T\tilde{q}_k}\right)B_k - S_k\frac{\tilde{q}_k^T\tilde{r}_k}{\tilde{y}_k^T\tilde{q}_k},$$

$$B_k = -\frac{\tilde{y}_k^T\tilde{r}_k}{\tilde{y}_k^T\tilde{q}_k}, \quad S_k = \frac{\tilde{y}_k^T\tilde{q}_k}{\tilde{q}_k^T\tilde{q}_k}. \qquad (18)$$

S_k, which is referred to as the *scaling factor*, is not strictly necessary (Luenberger, 1984). Battiti and Masulli (1990) have used $S_k = 1$ with positive results. Again, the safeguarded quadratic univariate minimization algorithm has been used in our experiments to estimate an appropriate step size.

5. THE SCG ALGORITHM

It is possible to use another approach in estimating the step size than the line-search technique. The idea is to estimate the term $\tilde{s}_k = E''(\tilde{w}_k)\tilde{p}_k$ in CG with a non-symmetric approximation of the form (Hestenes, 1980):

$$\tilde{s}_k = E''(\tilde{w}_k)\tilde{p}_k$$
$$\approx \frac{E'(\tilde{w}_k + \sigma_k\tilde{p}_k) - E'(\tilde{w}_k)}{\sigma_k}, \quad 0 < \sigma_k \ll 1. \qquad (19)$$

The approximation tends in the limit to the true value of $E''(\tilde{w}_k)\tilde{p}_k$. The calculation complexity and memory usage of $\tilde{s}_k$ are, respectively, $O(3N^2)$ and $O(N)$.[5] If this strategy is combined with the CG approach, we get an algorithm directly applicable to a feedforward neural network. This slightly modified version of the original CG algorithm will also be referred to as CG.

The CG algorithm was tested on an appropriate test problem. It failed in almost every case and converged to a nonstationary point. Cause of this failure is that the algorithm only works for functions with positive definite Hessian matrices, and that the quadratic approximations on which the algorithm works can be very poor when the current point is far from the desired minimum. The Hessian matrix for the global error function E has shown to be indefinite in different areas of the weight space, which explains why CG fails in the attempt to minimize E.

We propose a new solution to this problem. The approach is new not only in the context of learning in feedforward neural networks but also in the context of

[5] It is in fact possible to calculate $\tilde{s}_k$ exactly using automatic differentiation (Yoshida, 1991) without explicitly calculating the Hessian. This would involve $O(9N^2)$ calculations.

the underlying optimization theory which we have discussed so far. The idea is to combine the model-trust region approach, known from the Levenberg-Marquardt algorithm,[6] with the conjugate gradient approach. Let us introduce a scalar λ_k in CG, which is supposed to regulate the indefiniteness of $E''(\tilde{w}_k)$.[7] This is done by setting

$$\tilde{s}_k = \frac{E'(\tilde{w}_k + \sigma_k \tilde{p}_k) - E'(\tilde{w}_k)}{\sigma_k} + \lambda_k \tilde{p}_k, \tag{20}$$

and adjusting λ_k in each iteration looking at the sign of δ_k, which directly reveals if $E''(\tilde{w}_k)$ is not positive definite. If $\delta_k \le 0$, then the Hessian is not positive definite and λ_k is raised and $\tilde{s}_k$ is estimated again. If the new $\tilde{s}_k$ is renamed as $\bar{\tilde{s}}_k$ and the raised λ_k as $\bar{\lambda}_k$, then $\bar{\tilde{s}}_k$ is

$$\tilde{w}_k = \tilde{s}_k + (\bar{\lambda}_k - \lambda_k)\tilde{p}_k. \tag{21}$$

Assume in a given iteration that $\delta_k \le 0$. It is possible to determine how much λ_k should be raised in order to get $\delta_k > 0$. If the new δ_k is renamed as $\bar{\delta}_k$, then

$$\begin{aligned}\bar{\delta}_k &= \tilde{p}_k^T \bar{\tilde{s}}_k \\ &= \tilde{p}_k^T(\tilde{s}_k + (\bar{\lambda}_k - \lambda_k)\tilde{p}_k) = \delta_k + (\bar{\lambda}_k - \lambda_k)|\tilde{p}_k|^2 > 0 \Rightarrow \\ &\qquad \bar{\lambda}_k > \lambda_k - \frac{\delta_k}{|\tilde{p}_k|^2}. \end{aligned} \tag{22}$$

Equation (22) implies that if λ_k is raised with more than $-(\delta_k/|\tilde{p}_k|^2)$, then $\bar{\delta}_k > 0$. The question is: How much should $\bar{\lambda}_k$ be raised to get an optimal solution? This question cannot yet be answered, but it is clear that $\bar{\lambda}_k$ in some way should depend on λ_k, δ_k, and $|\tilde{p}_k|^2$. A reasonable choice is

$$\bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{|\tilde{p}_k|^2}\right). \tag{23}$$

This leads to

$$\begin{aligned}\bar{\delta}_k &= \delta_k + (\bar{\lambda}_k - \lambda_k)|\tilde{p}_k|^2 \\ &= \delta_k + \left(2\lambda_k - 2\frac{\delta_k}{|\tilde{p}_k|^2} - \lambda_k\right)|\tilde{p}_k|^2 \\ &= -\delta_k + \lambda_k|\tilde{p}_k|^2 > 0. \end{aligned} \tag{24}$$

The step size is given by

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{\tilde{p}_k^T \tilde{s}_k + \lambda_k|\tilde{p}_k|^2}. \tag{25}$$

The values of λ_k directly scale the step size in such a way that the bigger λ_k is the smaller the step size, which agrees well with our intuition of the function of λ_k.

The quadratic approximation E_{qw}, on which the algorithm works, may not always be a good approximation to $E(\tilde{w})$ since λ_k scales the Hessian matrix in an artificial way. A mechanism to raise and lower λ_k is needed which gives a good approximation, even when the Hessian is positive definite. Define

$$\begin{aligned}\Delta_k &= \frac{E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)}{E(\tilde{w}_k) - E_{qw}(\alpha_k \tilde{p}_k)} \\ &= \frac{2\delta_k[E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)]}{\mu_k^2}. \end{aligned} \tag{26}$$

Here Δ_k is a measure of how well $E_{qw}(\alpha_k \tilde{p}_k)$ approximates $E(\tilde{w}_k + \alpha_k \tilde{p}_k)$ in the sense that the closer Δ_k is to 1, the better is the approximation. λ_k is raised and lowered following the formula

$$\begin{aligned} &\text{if} \quad \Delta_k > 0.75, \quad \text{then} \quad \lambda_k = \tfrac{1}{4}\lambda_k \\ &\text{if} \quad \Delta_k < 0.25, \quad \text{then} \quad \lambda_k = \lambda_k + \frac{\delta_k(1 - \Delta_k)}{|\tilde{p}_k|^2}. \end{aligned} \tag{27}$$

The formula for $\Delta_k < 0.25$ increases lambda such that the new step size is equal to the minimum to a quadratic polynomial fitted to $E'(\tilde{w}_k)^T\tilde{p}_k$, $E(\tilde{w}_k)$, and $E(\tilde{w}_k + \alpha_k \tilde{p}_k)$ (Williams, 1990, personal communication). The SCG algorithm is as shown below.

1. Choose weight vector $\tilde{w}_1$ and scalars $0 < \sigma \le 10^{-4}$, $0 < \lambda_1 \le 10^{-6}$, $\bar{\lambda}_1 = 0$.
   Set $\tilde{p}_1 = \tilde{r}_1 = -E'(\tilde{w}_1)$, $k = 1$ and success = true.
2. If success = true, then calculate second order information:
   $\sigma_k = \sigma/|\tilde{p}_k|$,
   $\tilde{s}_k = (E'(\tilde{w}_k + \sigma_k \tilde{p}_k) - E'(\tilde{w}_k))/\sigma_k$,
   $\delta_k = \tilde{p}_k^T \tilde{s}_k$.
3. Scale δ_k: $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|\tilde{p}_k|^2$.
4. If $\delta_k \le 0$ then make the Hessian matrix positive definite:
   $\bar{\lambda}_k = 2(\lambda_k - \delta_k/|\tilde{p}_k|^2)$,
   $\delta_k = -\delta_k + \lambda_k|\tilde{p}_k|^2$,
   $\lambda_k = \bar{\lambda}_k$.
5. Calculate step size:
   $\mu_k = \tilde{p}_k^T \tilde{r}_k$,
   $\alpha_k = \mu_k/\delta_k$.
6. Calculate the comparison parameter:
   $\Delta_k = 2\delta_k[E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)]/\mu_k^2$.
7. If $\Delta_k \ge 0$ then a successful reduction in error can be made:

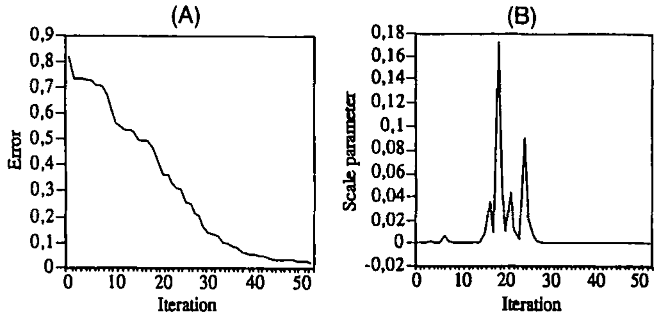

FIGURE 1. SCG functioning on the logistic map problem.

[6] The Levenberg-Marquardt algorithm is a variation of the standard Newton algorithm (Fletcher, 1975).

[7] λ_k is also known as a Lagrange Multiplier (Fletcher, 1975).

TABLE 1
Results From the Parity Problem

	BP	SCG	CGL	BFGS
Par.	av./st. dev./fai.	av./st. dev./fai./sp.	av./st. dev./fai./sp.	av./st. dev./fai./sp.
3	3475/1020/0	413/306/1/8.4	1232/1383/1/2.8	736/473/0/4.7
4	16427/10185/1	1727/725/2/9.5	3320/3147/1/4.9	3004/3458/0/5.5
5	9864/5651/2	2131/1494/1/4.6	3682/2029/0/2.7	3246/2387/3/3.0
6	28671/20727/6	2811/1548/2/10.2	5435/6036/1/5.3	5601/3021/2/5.1
7	48878/38293/4	3801/3593/1/12.9	9903/12545/1/4.9	9343/10902/2/5.2
8	134130/64572/2	6206/3077/1/21.6	12518/14012/2/10.7	11426/8575/4/11.7
9	189453/53535/4	8105/5879/0/23.4	25855/22094/3/7.3	25748/24165/0/7.4

av. = Average Number of cu's. *st. dev.* = Standard Deviation. *fai.* = Number of Failures. *sp.* = Speed-Up Relative to BP.

```
    $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$,
    $\tilde{r}_{k+1} = -E'(\tilde{w}_{k+1})$,
    $\bar{\lambda}_k = 0$, success = true.
    If $k \bmod N = 0$ then restart algorithm:
        $\tilde{p}_{k+1} = \tilde{r}_{k+1}$
    else:
        $\beta_k = (|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T \tilde{r}_k)/\mu_k$,
        $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$.
    If $\Delta_k \geq 0.75$, then reduce the scale parameter:
        $\lambda_k = \frac{1}{4}\lambda_k$.
  else:
    $\bar{\lambda}_k = \lambda_k$,
    success = false.
8. If $\Delta_k < 0.25$, then increase the scale parameter:
    $\lambda_k = \lambda_k + (\delta_k(1 - \Delta_k)/|\tilde{p}_k|^2)$.
9. If the steepest descent direction $\tilde{r}_k \neq \tilde{0}$, then set $k = k + 1$ and go to 2 else terminate and return $\tilde{w}_{k+1}$ as the desired minimum.
```

The value of σ should be as small as possible, taking the machine precision into account. When σ is kept small ($\leq 10^{-4}$), experiments indicate that the value of σ is not critical for the performance of SCG (see Section 6.3). Because of that, SCG seems not to include any user dependent parameters which values are crucial for the success of the algorithm. This is a major advantage compared to the line search based algorithms which include those kinds of parameters.

For each iteration there is one call of $E(\tilde{w})$ and two calls of $E'(\tilde{w})$, which gives a calculation complexity per iteration of $O(7N^2)$. When the algorithm is implemented, this complexity can be reduced to $O(6N^2)$ because the calculation of $E(\tilde{w})$ can be built into one of the calculations of $E'(\tilde{w})$. In comparison with BP, SCG involves twice as much calculation work per iteration since BP has a calculation complexity of $O(3N^2)$ per iteration. The calculation complexity of CGL and BFGS is about $O(6\text{–}20N^2)$ since the line search, on average, involves 6–20 calls of $E(\tilde{w})$ or $E'(\tilde{w})$ per iteration (Gill, Murray, & Wright, 1980).

When λ_k is zero, SCG is equal to the CG shown before. Figure 1 illustrates SCG functioning on an appropriate test problem.[8] Graph (A) shows the error development versus learning iteration. The error decreases monotonically towards zero, which is characteristic for SCG because an error increase is not allowed. At several iterations, the error is constant for one or two iterations.[9] In these instances the Hessian matrix has not been positive definite and λ_k has been increased using eqn (23). The development of λ_k is shown in graph (B). λ_k varies between 0 and 25 iterations and is 0 in the rest of the minimization. This reveals that $E''(\tilde{w})$ has not been positive definite in the beginning of the minimization. This is not surprising since the closer the current point is to the desired minimum, the greater the probability that $E''(\tilde{w})$ is positive definite (Hestenes, 1980). We observe that whenever eqn (23) is used to increase λ_k, a large reduction in error is achieved.[10]

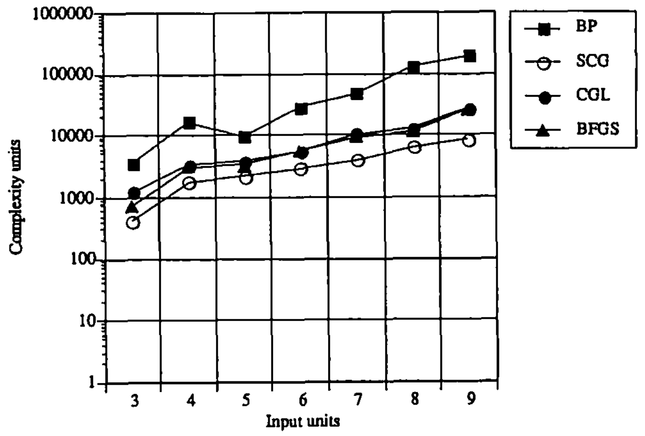

FIGURE 2. Number of complexity units vs. number of input units for three layer networks on 3, 4, 5, 6, 7, 8, and 9 bit parity problems when tested on the SCG, BP, CGL, and BFGS algorithms. Notice the logarithmic scale.

[8] The test problem was the logistic map problem described in Battiti and Masulli (1990).
[9] See iteration 6, 13, 20, and 23.
[10] See iteration 7, 16, 21, and 24.

6. TEST RESULTS

6.1. Comparison Metric

In order to compare the performance of the different algorithms, some kind of comparison metric is needed. Obviously the number of iterations is not a valid metric considering that the calculation complexity per iteration is not the same for any of the algorithms. Forward and backward passing of all the patterns through the network costs in the order of $O(N^2)$ and $O(3N^2)$, respectively, which is an order of magnitude greater than any other calculation in a given iteration for any of the algorithms. For that reason it seems reasonable to define the comparison metric proportional to the amount of forward and backward passings of patterns. Define a *complexity unit* (cu) to be equivalent to the complexity of one forward passing of all patterns in the training set. Then calculating the error costs 1 cu while calculating the derivative cost 3 cu. The complexity unit will be used to compare the performance of the different algorithms.

6.2. The Parity Problem

The aim of this test was to compare the performance of SCG with BP, CGL, and BFGS. The algorithms were tested on 3, 4, 5, 6, 7, 8, and 9 bit parity problems using 20 different initial weight vectors.[11] Three layer neural network architectures were used for each problem.[12] A training set containing all possible input patterns was used, i.e., 2^n patterns. The comparison metric described above was used in comparing the performance of the algorithms. The algorithms were terminated when the average error was less than 10^{-4} or the number of iterations had exceeded an appropriate large number of iterations. BP was run with learning rate 0.2 on parity 3–6, 0.05 on parity 7, and 0.01 on parity 8–9. This lowering of the learning rate was done in order to get BP to converge. The momentum parameter was set to 0.9 for all problems. The line search parameter η in CGL and BFGS was set to 0.25. The results are illustrated in Table 1. We observe that SCG is two to three times faster than CGL and BFGS on all problems.

It would also be interesting to visualize how the learning time is scaled by SCG, CGL, BFGS, and BP. According to Hinton, the learning time for BP should be approximately $O(N^3)$, i.e., the total number of complexity units, each costing $O(N^2)$ time, should be approximately $O(N)$. This depends, however, on the

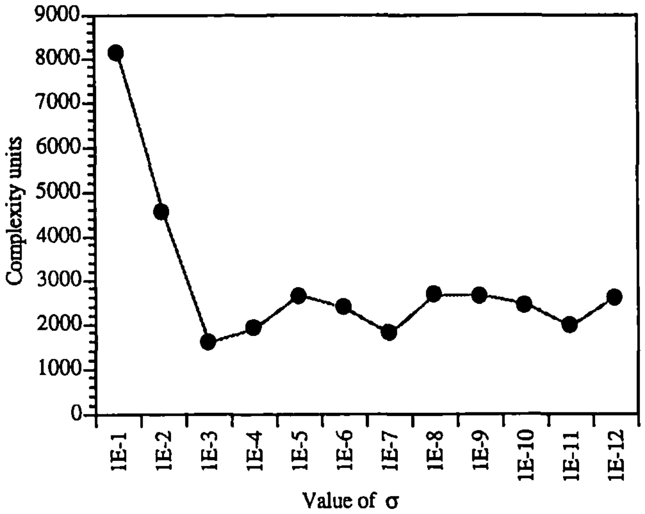

FIGURE 3. SCG on parity 5 with different σ-values.

nature of the task (Hinton, 1989; Tesauro, 1987). Judd (1987) shows that in the worst case it is exponential. Figure 2 uses a logarithmic plot to illustrate the number of complexity units versus the number of input units. The curves are clearly all sublinear indicating that all four algorithms scale polynomial on this particular problem. The BP curve and the increasing speed-ups in Table 1 indicate that the scaling of BP is worse than for the other algorithms. There seems to be no significant difference in the scaling of SCG, CGL, and BFGS.

6.3. SCG Performance vs. Different Values of σ.

The aim of this test was to determine how crucial the value of the σ-parameter is to the performance of SCG. Twelve different values for σ were used on the parity 5 problem using 20 different initial weight vectors. The result is shown in Figure 3. We observe that the average performance of SCG is not significantly affected when the value of σ is small ($\leq 10^{-4}$). For $\sigma \leq 10^{-4}$, the number of failures (local minima) was in the range 0–2 and the standard deviation was 330. When σ was less than 10^{-12}, roundoff errors began to have an effect.[13]

7. CONCLUSION

An optimization approach was used to introduce a learning algorithm (SCG), which is more effective than the standard BP, CGL, and the BFGS. SCG does not contain any user-dependent parameters whose values are crucial for the success of SCG. By using a step size scaling mechanism, SCG avoids a time consuming line search per learning iteration, which makes the algo-

[11] Only 10 different initial weight vectors were used for BP on parity 8 and 9 because of the large amount of cpu-time involved in these experiments.

[12] $n - n - 1$ architectures where n is the number of bits.

[13] All the experiments was run on a SUN-4 machine.

rithm faster than other second-order algorithms recently proposed (CGL, BFGS).

REFERENCES

Battiti, R. (1989). Accelerated backpropagation learning: Two optimization methods. *Complex Systems*, **3**, 331–342.

Battiti, R. (1990). Optimization methods for back-propagation: Automatic parameter tuning and faster convergence. *IJNNC-90-WASH-DC*, **1**, 593–596.

Battiti, R., & Masulli, F. (1990). BFGS Optimization for faster and automated supervised learning. *INCC 90 Paris, International Neural Network Conference*, **2**, 757–760.

Fletcher, R. (1975). *Practical methods of optimization.* New York: John Wiley & Sons. Gill, P. E., Murray, W., & Wright, M. H. (1980). Practical optimization. New York: Academic Press Inc.

Gill, P. E., & Murray, W. (1974). *Safeguarded steplength algorithms for optimization using descent methods.* NPL Report NAC 37, National Physica Laboratory, Division of Numerical Analysis and Computing, Middlesex, England.

Hestenes, M. (1990). *Conjugate direction methods in optimization.* New York: Springer-Verlag.

Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence*, **40**, 185–234.

Johansson, E. M., Dowla, F. U., & Goodman, D. M. (1991). *Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. International Journal of Neural Systems*, 2(4), 291–302.

Judd, J. S. (1987). *Complexity of connectionist learning with various node functions.* COINS Technical Report 87-60, University of Amherst, Amherst, MA.

Le Cun, Y. (1989). Generalization and network design strategies. In M. Pfeifer (Ed.), *Connectionism in perspective* (pp. 143–155). Amsterdam: North Holland.

Luenberger, D. (1984). *Linear and nonlinear programming.* Reading, MA: Addison-Wesley Publishing Company, Inc.

Møller, M. F. (1990). Learning by conjugate gradients. *Proceedings of the 6th International Meeting of Young Computer Scientists*, Czechoslovakia, LNCS, **464**, 184–195.

Møller, M. F. (1991). Supervised learning on large redundant training sets (in preparation).

Powell, M. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, **12**(2), 241–254.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing: Exploration in the microstructure of cognition* (pp. 318–362). Cambridge, MA: MIT Press.

Tesauro, G. (1987). Scaling relationships in back-propagation learning: Dependence on training set size. *Complex Systems*, **2**, 367–372.

Yoshida, T. (1991). A learning algorithm for multilayered neural networks: A Newton method using automatic differentiation. *IJCNN-91 Seattle*, Poster.

Watrous, R. L. (1987). Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. *Proceedings IEEE 1st International Conference on Neural Networks*, **2**, 619–628.

Williams, P. School of Cognitive and Computing Sciences, University of Sussex, UK. (1990). Personal communication.

APPENDIX. LINE SEARCH

A well known line search technique is that of Successive polynomial approximation (Gill & Murray, 1974), where the function $E(\tilde{w})$ is approximated by a simple function $e(\tilde{w})$ which agrees exactly with $E(\tilde{w})$ in either function value or function value and derivatives at a certain number of points. $e(\tilde{w})$ is normally chosen to be a quadratic or cubic polynomial depending on whether or not the derivatives of $E(\tilde{w})$ are available or easily calculated. We will use a quadratic polynomial and thereby avoiding calculating the derivatives of the error function, which involves approximately three times as many calculations than calculating the error. Define the function $f(x)$ as

$$f(x) = E(\tilde{w}_k + x\tilde{p}_k). \tag{28}$$

Assume that the minimum for $f(x)$ is bracketed by $(u, f(u))$, $(v, f(v))$ and that a third point $(x, f(x))$ in between is known. The minimum α for the quadratic polynomial passing through the three points is given by $x + s/q$ where s, q is:

$$s = [(v - x)^2(f(u) - f(x)) - (u - x)^2(f(v) - f(x))]$$
$$q = -2[(u - x)(f(v) - f(x)) - (v - x)(f(u) - f(x))]. \tag{29}$$

Successive applications of eqn (29) can be shown to be superlinearly convergent when some mild conditions of $E(\tilde{w})$ are satisfied. The disadvantage of line search techniques is, obviously, that each successive step involves several calculations of the error which is of the order of $O(N^2)$ calculations, where N is the number of weights and biases in the network. Even initializing the line search algorithm, i.e., bracketing the minimum, can cost several calculations of the error. Because of the calculation complexity involved in each step, the line search should terminate after a small amount of steps. The termination criteria used in this paper is (Gill, Murray, & Wright, 1980):

$$E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_j\tilde{p}_k) \Leftarrow \eta\alpha_j E'(\tilde{w}_k)\tilde{p}_k \quad 0 < \eta \le 0.5, \tag{30}$$

where α_j is the quadratic minimum for the j-th iteration in the successive line search. Other termination criteria exists. Common for all is that they include at least one user dependent parameter, such as η in eqn (30). The values of these parameters are often crucial for the amount of computations involved in the line search and can also be crucial for the success of the line search dependent algorithm. Terminating the line search before the actual minimum is found is called *inexact line search*. When the function to be minimized is nonquadratic, like the error function, making an exact line search is not worthwhile because the direction of search is also only an approximation to the exact direction. A slightly extended version of the quadratic line search technique is used in the experiments called *safeguarded quadratic univariate minimization* (Gill & Murray, 1974).

NOMENCLATURE

N	number of weights and biases.
P	number of patterns in the training set.
$\mathcal{R}^N$	N-dimensional euclidean space
$\tilde{v}$	vector
$\tilde{v}^T$	transpose of $\tilde{v}$
$\tilde{v}^i$	the i-th coordinate of $\tilde{v}$
$\lvert\tilde{v}\rvert$	length of vector
$E(\tilde{w})$	global error function
$E'(\tilde{w})$	gradient to global error function
$E''(\tilde{w})$	Hessian matrix to global error function