PB-339
PREPRINT

A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning

Martin F. Møller

November 13, 1990

Computer Science Department
University of Aarhus
Denmark

This is a preprint of a paper intended for publication in a journal. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning

Martin F. Møller
Computer Science Department
University of Aarhus, Denmark
email: fodslett@daimi.aau.dk

Abstract— A supervised learning algorithm (Scaled Conjugate Gradient, SCG) with superlinear convergence rate is introduced. The algorithm is based upon a class of optimization techniques well known in numerical analysis as the Conjugate Gradient Methods. SCG uses second order information from the neural network but requires only O(N) memory usage, where N is the number of weights in the network. The performance of SCG is benchmarked against the performance of the standard backpropagation algorithm (BP) [13], the conjugate gradient backpropagation (CGB) [6] and the one-step Broyden-Fletcher-Goldfarb-Shanno memoryless quasi-Newton algorithm (BFGS) [1]. SCG yields a speed-up of at least an order of magnitude relative to BP. The speed-up depends on the convergence criterion, i.e., the bigger demand for reduction in error the bigger the speed-up. SCG is fully automated including no user dependent parameters and avoids a time consuming line-search, which CGB and BFGS uses in each iteration in order to determine an appropriate step size.

Incorporating problem dependent structural information in the architecture of a neural network often lowers the overall complexity. The smaller the complexity of the neural network relative to the problem domain, the bigger the possibility that the weight space contains long ravines characterized by sharp curvature. While BP is inefficient on these ravine phenomena, it is shown that SCG handles them effectively.

1 Introduction

1.1 Motivation

Several adaptive learning algorithms for feed-forward neural networks has recently been discovered [5]. 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, because 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 backpropagation algorithm [13] which often behaves very badly on large-scale problems and which 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 [17].

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 only one class of optimization methods exists that are able to handle large-scale problems in an effective way. These methods are often referred to as the *Conjugate Gradient Methods* [4], [2], [3], [12]. Several conjugate gradient algorithms have recently been introduced as learning algorithms in neural networks [6], [1], [11]. Johansson, Dowla and Goodman describes in detail the theory of general conjugate gradient methods and how to apply the methods in feed-forward neural networks. They conclude that their algorithm (CGB)[1] is an order of magnitude faster than the standard backpropagation algorithm (BP) [13] when tested on the parity problem. Battiti and Masulli has used a variation of the standard conjugate gradient method, the one-step Broyden-Fletcher-Goldfarb-Shanno memoryless quasi-Newton algorithm (BFGS), as an alternative learning algorithm. Compared to BP, the algorithm yields a speed-up of 100-500 relative to the amount of learning iterations used [1]. Unfortunately Battiti and Masulli do not take the calculation complexity per learning iteration into account when calculating this impressive speed-up. Both CGB and BFGS raise the calculation complexity per learning iteration considerable, because they have to perform a line-search in order to determine an appropriate step size. A line-search involves several calculations[2] of either the global error function or the derivative to the global error function, both of which raise the complexity.

The paper introduces a variation of a conjugate gradient method (Scaled Conjugate Gradient, SCG), which avoids the line-search per learning iteration by using a Levenberg-Marquardt approach [2] in order to scale the step size.

---

[1] Johansson, Dowla and Goodman suggest several variations of a standard conjugate gradient algorithm. The variations are in this paper for convinience referred to as one algorithm named Conjugate Gradient Backpropagation (CGB) .

[2] In average about 3-15 calculations.

1.2. Notation

Let an arbitrary feed-forward 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 $\Re^N$, where N is the number of weights and biases in the network. A weight vector will often be referred to as a point in $\Re^N$ or just a point in weight space. Let w be the weight vector defined by

$$(1) \qquad w = (\ldots, w_{ij}^{(l)}, w_{i+1j}^{(l)}, \ldots, w_{N_l j}^{(l)}, \theta_j^{(l+1)}, w_{ij+1}^{(l)}, w_{i+1j+1}^{(l)}, \ldots)$$

where $w_{ij}^{(l)}$ is the weight from unit number i in layer number l to unit number j in layer number l+1, N_l is the number of units in layer l, 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(w) depending on all the weights and biases is attached to the neural network. E(w) could be the standard least square function or any other appropriate error function. E(w) can be calculated with one forward pass and the gradient $E'(w)$ with one backward pass [13]. If we assume that the number of patterns to be learned is proportional to the number of weights, which is reasonable according to Hinton [5], the complexity of calculating either E(w) or $E'(w)$ is $O(N^2)$, where N is the number of weights and biases. $E'(w)$ is given by

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

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 w is a column-vector and w^T, the transpose of 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 through out the implementation.Weight vectors and scalars are respectively indicated by roman and greek letters.

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

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

The *weight length* is defined as

$$(4) \qquad |w| = \Big[\sum_{j=1}^{N}(w^i)^2\Big]^{\frac{1}{2}}$$

It might also be useful to recall that the error function E(w) in a given point (w+y) in $\Re^N$ can be expressed by the well known Taylor expansion [3],[4]

$$(5) \qquad E(w+y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y + \ldots$$

A NxN matrix A is said to be *positive definite* if

$$(6) \qquad y^T A y > 0 \qquad \forall\, y \in \Re^N$$

Let $p_1,..,p_k$ be a set of non zero weight vectors in $\Re^N$. The set is said to be a *conjugate system* with respect to a non-singular symmetric NxN matrix A if the following holds [4]

$$(7) \qquad p_i^T A p_j = 0 \qquad (i \neq j,\ i = 1,..,k)$$

The set of points w in $\Re^N$ satisfying

$$(8) \qquad w = w_1 + \alpha_1 p_1 + \ldots + \alpha_k p_k,\ \ \alpha_i \in \Re,$$

where w_1 is a point in weight space and $p_1,..,p_k$ is a subset of a conjugate system, is called a *k-plane* or π_k [4].

2 Optimization strategy

Most of the optimization methods used to minimize functions are based on the same strategy. The minimization 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(w) [2].

```
1. Choose initial weight vector $w_1$ and set k = 1.
2. Determine a search direction $p_k$ and a step size $\alpha_k$
   so that $E(w_k + \alpha_k p_k) < E(w_k)$.
3. Update vector: $w_{k+1} = w_k + \alpha_k p_k$.
4. If $E'(w_k) \neq 0$ then set k = k+1 and go to 2
   else return $w_{k+1}$ 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.

If the search direction p_k is set to the negative gradient $-E'(w)$ and the step size α_k to a constant ε, then the algorithm becomes the gradient descent algorithm [3]. In the context of neural networks this is the BP algorithm without momentum term [13]. Minimization by gradient descent is based on the linear approximation $E(w+y) \approx E(w) + E'(w)^T y$, which is the main reason why the algorithm often show poor convergence. Another reason is that the algorithm uses constant step size, which in many cases are inefficient and makes the algorithm less robust. Including a momentum term in the BP algorithm is an attempt in an ad hoc fashion to force the algorithm to use second order information from the network. Unfortunately the momentum term is not able to speed up the algorithm considerable, but causes the algorithm to be even less robust, because of the inclusion of another user dependent parameter, *the momentum constant.*

3 The SCG algorithm

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

Quadratic functions have some nice properties that general functions not necessarily have. Denote the quadratic approximation to E in a neighbourhood of a point w by $E_{qw}(y)$, so that $E_{qw}(y)$ is given by

$$(9) \qquad E_{qw}(y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y$$

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

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

The critical points are the solution to the linear system defined by (10). If a conjugate system is available the solution can be simplified considerable [4], [6]. Johansson, Dowla and Goodman shows in a very understandable way how. Let $p_1, \ldots, p_N$ be a conjugate system. Because $p_1, \ldots, p_N$ form a basis for $\Re^N$, the step from a starting point y_1 to a critical point y_* can be expressed as a linear combination of $p_1, \ldots, p_N$

$$(11)\qquad y_* - y_1 = \sum_{i=1}^{N} \alpha_i p_i\ ,\ \ \alpha_i \in \Re$$

Multiplying (11) with $p_j^T E''(w)$ and sudstituting $E'(w)$ for $-E''(w)y_*$ gives

$$(12)\qquad p_j^T(-E'(w) - E''(w)y_1) = \alpha_j p_j^T E''(w)p_j \qquad \Rightarrow$$

$$\alpha_j = \frac{p_j^T(-E'(w) - E''(w)y_1)}{p_j^T E''(w)p_j} = \frac{-p_j^T E_{qw}'(y_1)}{p_j^T E''(w)p_j}$$

The critical point y_* can be determined in N iterative steps using (11) and (12). Unfortunately y_* is not necessarily a minimum, but can be a saddle point or a maximum. Only if the Hessian matrix $E''(w)$ is positive definite then $E_{qw}(y)$ has a unique global minimum [4]. This can be realized by

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

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

The intermediate points $y_{k+1} = y_k + \alpha_k p_k$ given by the iterative determination of y_* are in fact minima for $E_{qw}(y)$ restricted to every k-plane π_k: $y = y_1 + \alpha_1 p_1 + \ldots + \alpha_k p_k$ [4]. How to determine these points recursively is shown in theorem 1. The proof is omitted.[5]

Theorem 1 *Let $p_1,\ldots,p_N$ be a conjugate system and y_1 a point in weight space. Let the points $y_2,\ldots,y_{N+1}$ be recursively defined by*

$$y_{k+1} = y_k + \alpha_k p_k\ ,$$

where $\alpha_k = \dfrac{\mu_k}{\delta_k}$, $\mu_k = -p_k^T E_{qw}'(y_k)$, $\delta_k = p_k^T E''(w)p_k$. Then y_{k+1} minimizes E_{qw} restricted to the k-plane π_k given by y_1 and $p_1,\ldots,p_k$ [4].

---

[3] $E''(w)$ is symmetric.

[4] $E''(w)y_* + E'(w) = 0$ by (10).

[5] A proof for theorem 1 can be found in [4].

We are now able to formulate a conjugate gradient algorithm as proposed by [4]. Select an initial weight vector y_1 and a conjugate system $p_1,\ldots,p_N$. 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 $y = y_1 + \alpha_1 p_1 + \ldots + \alpha_k p_k$, $\alpha_i \in \Re$. The algorithm assures that the global minimum for a quadratic function is detected in at most N iterations. If all the eigen values to the Hessian $E''(w)$ falls into multiple groups with values of the same size, then there is great possibility that the algorithm terminates in much less than N iterations. Praxis shows that this is often the case [2].

It is not necessary to know the conjugate weight vectors $p_1,\ldots,p_N$ in advance, they can be determined recursively. Initially p_1 is set to the steepest descent vector $-E'_{qw}(y_1)$. p_{k+1} is then determined recursively as a linear combination of the current steepest descent vector $-E'_{qw}(y_{k+1})$ and the previous direction p_k [6]. More precisely is p_{k+1} chosen as the orthogonal projection of $-E'_{qw}(y_{k+1})$ on the (N–k)-plane π_{N-k} conjugate to π_k. Theorem 2 shows without proof how this can be done.[6]

Theorem 2 *Let y_1 be a point in weight space and p_1 and r_1 equal to the steepest descent vector $-E'_{qw}(y_1)$. Define p_{k+1} recursively by*

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

where $r_{k+1} = -E'_{qw}(y_{k+1})$, $\beta_k = \frac{|r_{k+1}|^2 - r_{k+1} r_k}{p_k^T r_k}$ and y_{k+1} is the point generated in theorem 1. Then 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 y_1 and $p_1,\ldots,p_k$ [4].

For each iteration the above described algorithm is applied to the quadratic approximation E_{qw} of the global error function E in the current point w in weight space. Because the error function E(w) is non-quadratic 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 p_{k+1} to the current steepest descent direction r_{k+1} [2], [3], [4], [12]. The first version of the SCG algorithm is

1. Choose initial weight vector w_1.
   Set $p_1 = r_1 = -E'(w_1)$, $k = 1$.
2. Calculate second order information:
   $$s_k = E''(w_k) p_k,$$
   $$\delta_k = p_k^T s_k.$$
3. Calculate step size:
   $$\mu_k = p_k^T r_k,$$

---

[6] A proof for theorem 2 can be found in [4] and [6].

$$\alpha_k = \frac{\mu_k}{\delta_k}.$$

4. Update weight vector:

$$w_{k+1} = w_k + \alpha_k p_k,$$
$$r_{k+1} = -E'(w_{k+1}).$$

5. If k mod N = 0 then restart algorithm: $p_{k+1} = r_{k+1}$

else create new conjugate direction:

$$\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}r_k}{\mu_k},$$
$$p_{k+1} = r_{k+1} + \beta_k p_k.$$

6. If the steepest descent direction $r_k \neq 0$ then set k = k+1 and go to 2

else terminate and return w_{k+1} as the desired minimum.

Several other formulas for β_k can be derived [4], [2], [3], [6], but when the conjugate gradient methods are applied to non-quadratic functions the above formula, called the Hestenes-Stiefel formula, for β_k is considered superior [6]. When the algorithm shows poor development the formula forces the algortihm to restart because of the following relation

$$(14) \quad r_{k+1} \approx r_k \quad \Rightarrow \quad \beta_k \approx 0 \quad \Rightarrow \quad p_{k+1} \approx r_{k+1}$$

For each iteration the Hessian matrix $E''(w_k)$ has to be calculated and stored. Unfortunately is it 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. The solution to the problem is to estimate the $E''(w_k)p_k$ with

$$(15) \quad s_k = E''(w_k)p_k \approx \frac{E'(w_k+\sigma_k p_k) - E'(w_k)}{\sigma_k} \quad , 0 < \sigma_k << 1$$

The calculation complexity and memory usage of s_k is respectively $O(N^2)$ and $O(N)$, and the algorithm is now direct applicable to a feed-forward neural network [4].

The algorithm was tested on an appropriate test problem. It failed in almost any case and converged to a non stationary point. The reason 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 [4], [3]. The Hessian matrix for the global error function E has shown to be indefinite in different areas of the weight space, which explaines why the algorithm fails in the attempt to minimize E. The solution to the problem is to modify the algorithm so that it prevents the difficulties with the indefinite Hessian matrices. Both the CGB and the BFGS algorithm solves the problem by using a line-search per iteration in order to determine a better step size. A line-search could also be used in the above algorithm, but this would like in CGB and BFGS raise the calculation complexity per iteration considerable. Instead a Levenberg-Marquardt approach was used in modifying the algorithm [2].[7] The idea is to introduce a scalar λ_k, which is supposed to regulate the indefiniteness of $E''(w_k)$.[8] This is done by setting

$$(16) \quad s_k = \frac{E'(w_k+\sigma_k p_k)-E'(w_k)}{\sigma_k} + \lambda_k p_k$$

and for each iteration adjusting λ_k looking at the sign of δ_k, which directly reveal if $E''(w_k)$ is not positive definite. If $\delta_k \leq 0$ then λ_k is raised and s_k is estimated again. Call the new s_k for $\bar{s}_k$ and the raised λ_k for $\bar{\lambda}_k$. Then $\bar{s}_k$ is

$$(17) \quad \bar{s}_k = s_k + (\bar{\lambda}_k - \lambda_k)p_k$$

The Levenberg-Marquardt algorithm [2] has to raise λ_k with a constant factor, whenever the Hessian is not positive definite, because the algorithm contains no information about how much λ_k must be raised. This is however possible when using this approach in the above algorithm, as the values of δ_k indirectly reveal how much λ_k should be raised. Assume in a given iteration that $\delta_k \leq 0$. It is possible to determine how much λ_k should be raised in order to get $\delta_k > 0$.

$$(18) \quad \bar{\delta}_k = p_k^T \bar{s}_k = p_k^T (s_k + (\bar{\lambda}_k - \lambda_k)p_k) = \delta_k + (\bar{\lambda}_k - \lambda_k)|p_k|^2 > 0 \quad \Rightarrow$$
$$\bar{\lambda}_k > \lambda_k - \frac{\delta_k}{|p_k|^2}$$

(18) implies that if λ_k is raised with more than $-\frac{\delta_k}{|p_k|^2}$ then $\bar{\delta}_k > 0$. The question is with how much $\bar{\lambda}_k$ should be raised to get an optimal solution. This question can not yet be answered, but it is clear that $\bar{\lambda}_k$ in some way should depend on λ_k, δ_k and $|p_k|^2$. A choice that has shown to be reasonable is

$$(19) \quad \bar{\lambda}_k = 2(\lambda_k - \frac{\delta_k}{|p_k|^2})$$

---

[7] The Levenberg-Marquardt algorithm is a variation of the standard Newton algorithm.

[8] λ_k is also known as a Lagrange Multiplier [2].

This leads to

$$(20)\quad \bar{\delta}_k = \delta_k + (\bar{\lambda}_k - \lambda_k)|p_k|^2 = \delta_k + \left(2\lambda_k - 2\frac{\delta_k}{|p_k|^2} - \lambda_k\right)|p_k|^2 = -\delta_k + \lambda_k|p_k|^2 > 0$$

The step size is given by

$$(21)\quad \alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{p_k^T s_k + \lambda_k|p_k|^2}$$

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

Because λ_k scales the Hessian matrix in an artificial way, the quadratic approximation E_{qw} on which the algorithm works may in some points not be a very good approximation to E(w). In order to get a good approximation, a mechanism to raise an lower λ_k even when the Hessian is positive definite is needed. Define

$$(22)\quad \Delta_k = \frac{E(w_k) - E(w_k + \alpha_k p_k)}{E(w_k) - E_{qw}(\alpha_k p_k)} = \frac{2\delta_k[E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2}$$

Δ_k is a measure of how well $E_{qw}(\alpha_k p_k)$ approximates $E(w_k + \alpha_k p_k)$ 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 [2]

$$(23)\quad \begin{array}{l} \text{if } \Delta_k > 0.75 \text{ then } \lambda_k = \frac{1}{2}\lambda_k \\ \text{if } \Delta_k < 0.25 \text{ then } \lambda_k = 4\lambda_k \end{array}$$

The final SCG algorithm is as shown below.

1. Choose weight vector w_1 and scalars $\sigma > 0$, $\lambda_1 > 0$ and $\bar{\lambda}_1 = 0$.
   Set $p_1 = r_1 = -E'(w_1)$, $k = 1$ and success = true.
2. If success = true then calculate second order information:

   $$\sigma_k = \frac{\sigma}{|p_k|},$$

   $$s_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k},$$

   $$\delta_k = p_k^T s_k.$$
3. Scale s_k:

   $$s_k = s_k + (\lambda_k - \bar{\lambda}_k)p_k,$$

   $$\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|p_k|^2.$$

```
4.  If $\delta_k \le 0$ then make the Hessian matrix positive definite:
        $s_k = s_k + (\lambda_k - 2\frac{\delta_k}{|p_k|^2})p_k$,
        $\bar{\lambda}_k = 2(\lambda_k - \frac{\delta_k}{|p_k|^2})$,
        $\delta_k = -\delta_k + \lambda_k|p_k|^2, \quad \lambda_k = \bar{\lambda}_k$.
5.  Calculate step size:
        $\mu_k = p_k^T r_k, \quad \alpha_k = \frac{\mu_k}{\delta_k}$.
6.  Calculate the comparison parameter: $\Delta_k = \frac{2\delta_k[E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2}$.
7.  if $\Delta_k \ge 0$ then a successful reduction in error can be made:
        $w_{k+1} = w_k + \alpha_k p_k$,
        $r_{k+1} = -E'(w_{k+1})$,
        $\bar{\lambda}_k = 0$, success = true.
        7a. If k mod N = 0 then restart algorrithm: $p_{k+1} = r_{k+1}$
            else create new conjugate direction:
                $\beta_k = \frac{|r_{k+1}|^2 - r_{k+1} r_k}{\mu_k}$,
                $p_{k+1} = r_{k+1} + \beta_k p_k$.
        7b. If $\Delta_k \ge 0.75$ then reduce the scale parameter: $\lambda_k = \frac{1}{2}\lambda_k$.
    else a reduction in error is not possible: $\bar{\lambda}_k = \lambda_k$, success = false.
8.  If $\Delta_k < 0.25$ then increase the scale parameter: $\lambda_k = 4\lambda_k$
9.  If the steepest descent direction $r_k \ne 0$ then set k = k+1 and go to 2
    else terminate and return $w_{k+1}$ as the desired minimum.
```

For each iteration there is one call of $E(w)$ and two calls of $E'(w)$, which gives a calculation complexity per iteration of $O(3N^2)$. When the algorithm is implemented this complexity can be reduced to $O(2N^2)$, because the calculation of $O(N^2)$ can be built into one of the calculations of $E'(w)$. In comparison with BP, SCG involves twice as much calculation work per iteration, since BP has a calculation complexity of $O(N^2)$ per iteration. The calculation complexity of CGB and BFGS is about $O(2\text{-}15N^2)$ since the line-search in average involves 2-15 calls of $E(w)$ or $E'(w)$ per iteration [3]. When λ_k is zero, SCG is equal to the unmodified conjugate gradient algorithm shown before. Figure 1 illustrates SCG functioning on the logistic map problem, a test problem described later in detail. Graph A) shows the error development versus learning iteration. The error decreases monotonic 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

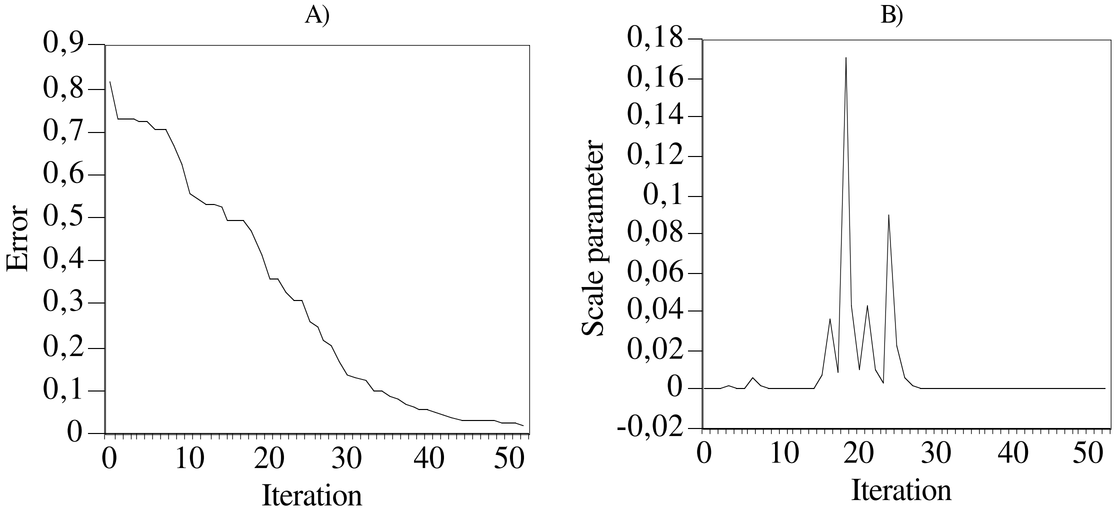

Figure 1. SCG functioning on the logistic map problem.

iterations.[9] The Hessian matrix has here been not positive definite and λ_k has been increased using equation (19). The development of λ_k is shown in graph B). λ_k is only varying between 0 and 25 iterations and is 0 in the rest of the minimization. This reveals, that E''(w) has been not positive definite only in the beginning of the minimization. This is not surprising because the closer the current point is to the desired minimum the bigger is the possibility that E''(w) is positive definite [4]. We observe that whenever equation (19) is used to increase λ_k a big reduction in error is archieved.[10]

4. Test results

4.1 The parity problem

The aim with this test is to compare the performance of SCG with BP and CGB. The comparison with CGB will be based on the results reported by Johansson, Dowla and Goodman. SCG and BP were tested on 3, 4, 5, 6 and 7 bit parity problems using 10 different initial weight vectors. Three and four layer neural network architectures was used for each problem.[11] A training set containing all possible input patterns was used, i.e., 2^n patterns. An average of the total calls of E(w) and E'(w) was used in comparing the performance of the algorithms. The convergence criterion was the same as used by Johansson, Dowla and Goodman, i.e, the algorithm was terminated when the average error was less than 10^{-6}. The results are illustrated in table 1. Johansson, Dowla and Goodman obtained speed-up ranging from 15 to 34 when testing CGB on 3, 4 and 5 bit parity problems. However, the results are not directly comparable with the results reported here, because their results are not based on average tests.

---

[9] See iteration 6, 13, 20 and 23.

[10] See iteration 7, 16, 21 and 24.

[11] 3-3-1, 3-3-3-1, 4-4-1, 4-4-4-1, 5-5-1, 5-5-5-1, 6-6-1, 6-6-6-1, 7-7-1 and 7-7-7-1 architectures.

Architecture	SCG	BP	Speed-up
3-3-1	154	6394	41.5
3-3-3-1	167	3587	21.5
4-4-1	494	15373	31.1
4-4-4-1	467	10899	23.3
5-5-1	642	18451	28.7
5-5-5-1	608	13679	22.5
6-6-1	1217	56341	46.3
6-6-6-1	1009	18110	17.9
7-7-1	1752	77370	44.2
7-7-7-1	1337	41846	31.3

Table 1. Results from the parity problem. Average of 10 tests.

It would also be interesting to see how the learning time is scaled by SCG and BP. According to Hinton the learning time for BP should be approximately $O(N^3)$, i.e., the total number of function calls, each costing $O(N^2)$ time, should be approximately $O(N)$. This depends, however, of the nature of task [5], [15]. Judd shows that in the worst case it is exponential [7]. Figure 2 illustrates the number of function calls versus the number of input units for each of the two network architectures using a logarithmic plot. We observe that the learning time for BP is bounded by $O(N^2 \log N)$ and $O(N^3)$ function calls, while the learning time for SCG is bounded by $O(N \log N)$ and $O(N^2)$ function calls. Four layer networks seems to yield slightly better performance for both algorithms. However, tests on other problems than the parity problem is needed to confirm these results.

4.2 The logistic map problem

In order to compare SCG with BFGS, SCG was tested on a problem original introduced by Lapedes and Farber [8], but used by Battiti and Masulli [1] in testing BFGS.

The logistic map is a discrete-time, non-linear dynamical system [1]. The recurrence relation is

$$(7) \qquad x_{n+1} = r\, x_n(1 - x_n) \qquad 0 < x_n < 1$$

According to Battiti and Masulli relation (7) is an ergodic, chaotic system, when r = 4. They created a 3-layer neural network consisting of one input, five hidden and one output unit. A sequence of 10 example pairs (x_n, x_{n+1}) was generated by the logistic map and used as a training set. The convergence criterion was set to 0.01, i.e., the network was said to have converged, when all outputs was within a margin of 0.01 from the corresponding target outputs. Battiti and Masulli obtained a speed-up of 500 compared to a variation of BP, called the Bold Driver Method (BD), where the learning rate is raised or lowered depending on whether the global error is increasing or decreasing [8], [16], [1]. This speed-up is, however, obtained without taking the calculation complexity per learning iteration into account.

SCG and BD were tested on a similar example. The convergence criterion was set to 0.1, 0.09, 0.08, …, 0.01. SCG and BD was tested on 10 different initial weight vectors for each of the 10 criteria.

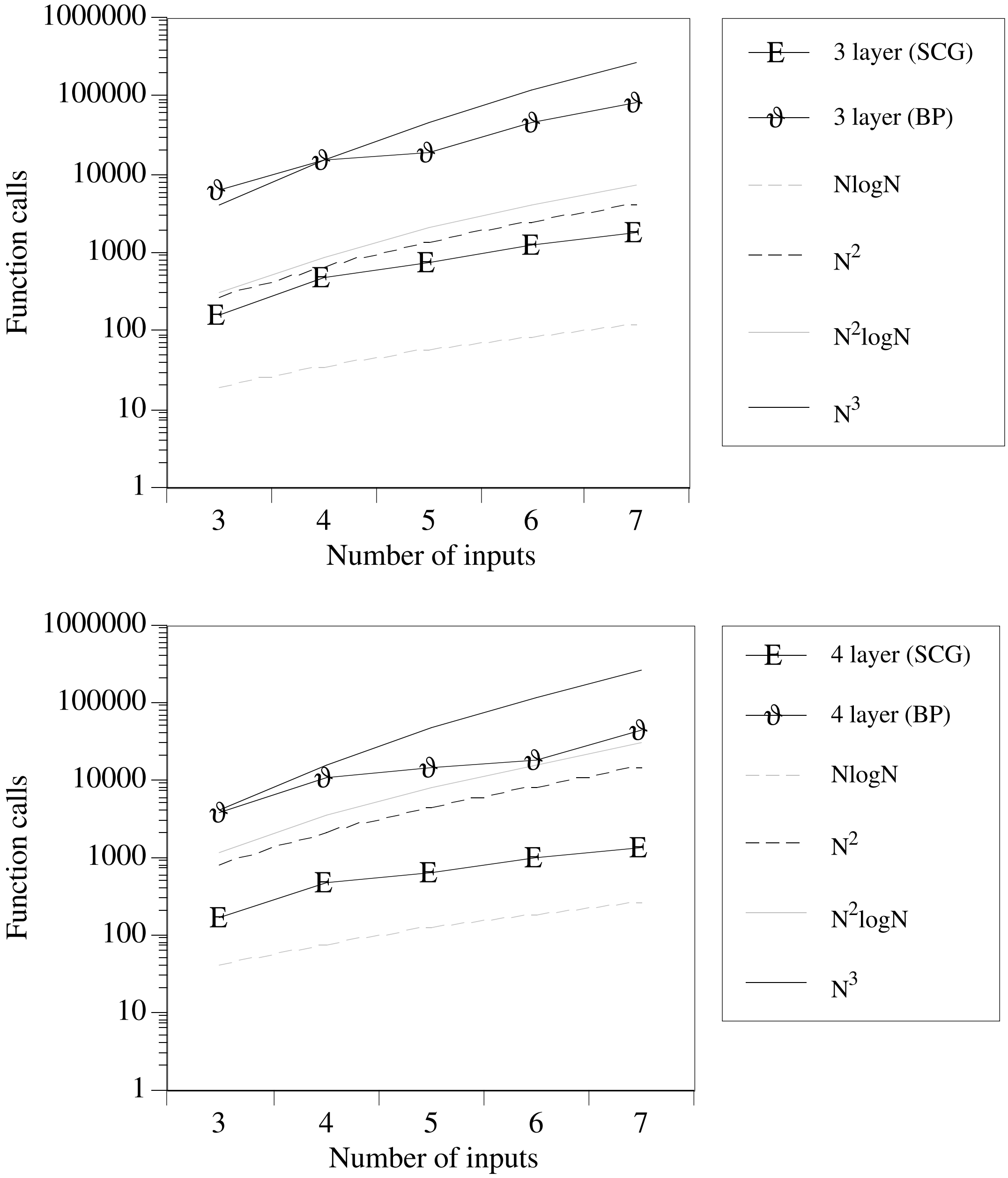

Figure 2. Function calls per number of input units for 3 and 4 layer networks on 3, 4, 5, 6 and 7 bit parity problems. Notice the logarithmic scale.

Criterion	SCG	BD	Speed-up
0.10	107	1044	9.8
0.09	126	2293	18.2
0.08	198	9459	47.8
0.07	319	19008	59.6
0.06	438	23518	53.7
0.05	497	30408	61.2
0.04	580	42491	73.3
0.03	683	57093	83.6
0.02	1011	241557	238.9
0.01	4718	2548570	540.2

Table 2. Results from the logistic map problem. Average of 10 tests.

An average of the total calls of E(w) and $E'(w)$ was used in comparing the performance of the algorithms. The results are illustrated in table 2. At a convergence criterion of 0.01 SCG obtains a speed-up of about 540 which is better than reported by Battiti and Masulli even though their speed-up not takes the calculation complexity per learning iteration into account.

Another interesting thing appears from table 1, when we look at the speed-up relative to the convergence criterion. Figure 3 shows the relation. The speed-up grows linearly from convergence criteria 0.1 to 0.03, but when the convergence criterion gets small the speed-up grows almost exponential . This is not surprising, when the main differences between SCG and BD are considered. BD is an optimization techniques based on first order information from the neural network. First order algorithms converge slower the closer they get to the minimum [17].

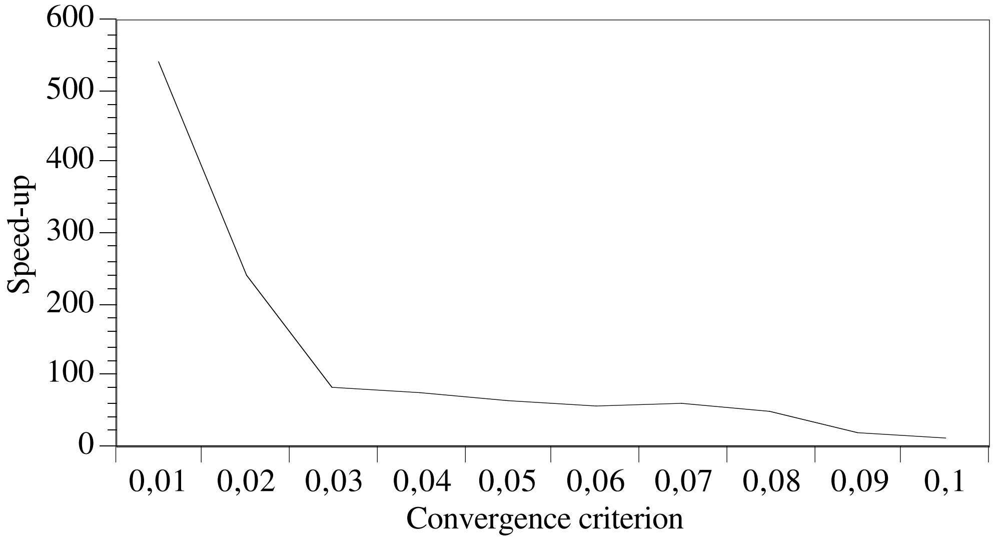

Figure 3. Speed-up versus convergence criterion.

It does in fact hold that although they converge to the minimum, they will never reach the minimum. SCG is based on second order information from the network. Second order algorithms converge faster the closer they get to the minimum. The smaller the convergence criterion one demands in a specific neural network application the bigger the speed-up one gets using SCG.

4.3 The =1 problem

If a neural network has to function in a dynamical environment the structure of the neural network must reflect the natural geometry or at any rate some informationally significant dimensions of the problem domain [9], [14]. Mühlenbein concludes in his discussion of the limitations of multi-layer perceptrons [10], that the backpropagation algorithm is not able to use such stuctural information. Mühlenbein uses the *"=1" boolean function* as a test example. The function determines whether a binary string is a 1-bit string or not. The =1 problem can be solved using a modular network approach. Smaller, independent networks are trained to exploit partial knowledge of the problem. These network modules are then incorporated into larger structures. The composed modular network solves the =1 problem, given that the smaller networks has learned the partial problems. The composed network for n = 8 is shown in figure 4. Mühlenbein shows that the network solves the =1 problem using the weights obtained from training the smaller network modules. Now knowing that a solution to the =1 problem exist, BP was tested on the network. It was not able to find a solution, and Mühlenbein conlude, that BP is not able to use structural information.

In order to confirm Mühlenbeins results and to determine whether SCG is able to find a solution, BP and SCG were tested using 5 different initial weight vectors and convergence criterion 0.1. Table 3 shows the result. Surprisingly BP is able to solve the problem in 3 out of 5 tests, although having major difficulties.

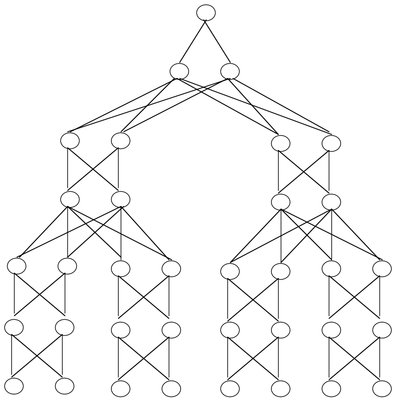

Figure 4. Composed modular network for =1 problem (n = 8) [10].

SCG	BP
8027	885270
966	357167
4014	2000000*
656	240754
1282	2000000*

Table 3. Results from SCG and BP on the =1 problem. * = failed to converge.

It is well known from the optimization literature that gradient descent methods, like BP, are very inefficient, when *the weight space contains long ravines that are characterized by sharp curvature across the ravine and a gently sloping floor* [1], [13].[12] The gradient descent algorithm does not use any information about the curvature in the minimization process. The conjugate gradient methods, like SCG, can handle the ravine phenomena more effectively, because they use second order information which characterizes the curvature. This major difference could explain, why SCG is superior on the =1 problem and other problems. The composed modular network approach described by Mühlenbein produces more layers and reduces the overall complexity of the neural network considerable compared to a general feed-forward neural network. It is our experience that the smaller the dimension of the weight space relative to the problem domain or the more layers, the bigger the probability for running into ravine phenomena. Figure 5 which shows a run with SCG and BP on the =1 problem support this thesis.

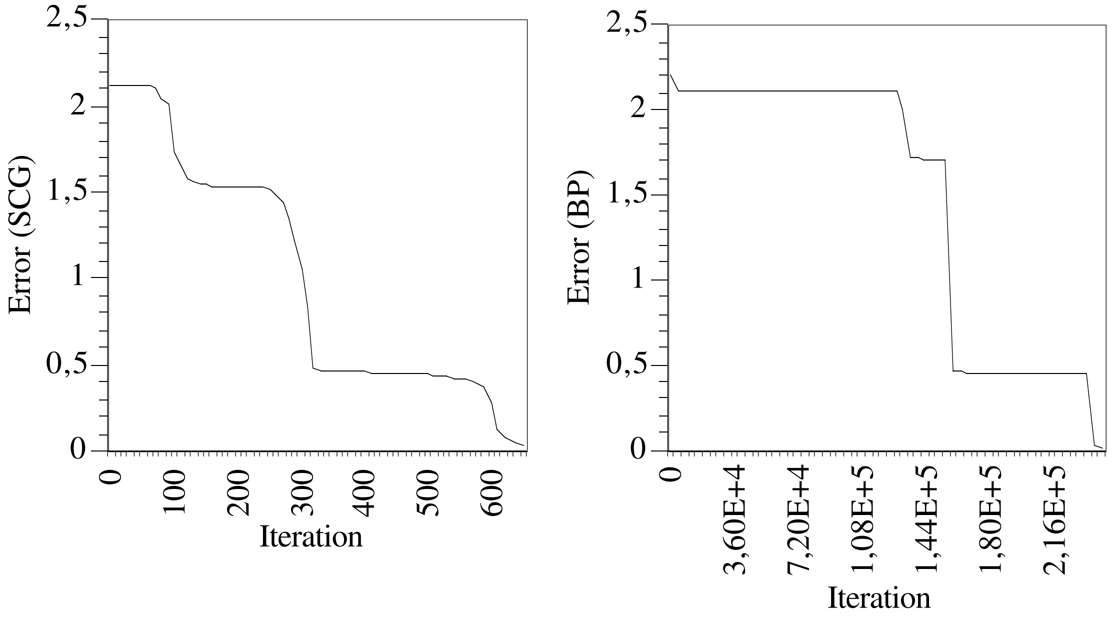

Figure 5. SCG and BP on the =1 problem.

---

[12] We will refer to this situation as a *ravine phenomenon*.

At three stages in both runs very little reduction in error is obtained[13] which indicates the presence of ravine phenomena. SCG does not avoid the ravines, but handles them effectively. BP uses many more iterations to get through the ravines.

5 Conclusion

Using an optimization approach an alternative and more effective learning algorithm (SCG) than the standard backpropagation (BP) has been introduced. SCG belongs to the class of Conjugate Gradient Methods, which shows superlinear convergence on most problems. Through several experiments we find that SCG is at least an order of magnitude faster than BP. The speed-up depends on the convergence criteria, i.e, the bigger demand for reduction in error, the bigger the speed-up. By using a step size scaling mechanism SCG avoids a time consuming line-search per learning iteration, which makes the algorithm faster than other second order algorithms recently proposed.

Tests on the 3, 4, 5, 6 and 7 bit parity problem suggest that SCG scale an order of magnitude better than BP; however, tests with smaller size training sets and bigger problems have to be made to conclude anything definite about the scaling of SCG.

SCG and BP were tested on the =1 problem using a network whose architecture contained problem dependent structural information. Mühlenbein has earlier reported that BP fails on this test and conludes that BP is not able to use structural information available in the network. Surprisingly BP, although having major difficulties, does solve the problem when enough iterations are used. Incorporating problem dependent structural information in the architecture of a neural network often lowers the overall complexity. The smaller the complexity of the neural network relative to the problem domain, the bigger the possibility that the weight space contains long ravines characterized by sharp curvature. It is well known that BP is very inefficient when these ravine phenomena occure, and for that reason BP has major difficulties solving the =1 problem. SCG can handle the ravine phenomena more effectively using second order information, and solves the =1 problem without these difficulties.

Acknowledgement

I would like to thank Brian Mayoh (AAU), Kim Plunkett (AAU) and Ole Østerby (AAU) for many good discussions and advice. I am also grateful to Eric Johansson (LLNC) who has been very helpful answering questions concerning his recent work on neural networks and conjugate gradient algorithms.

---

[13] At error 2.1, 1.6 and 0.4.

References

1. Battiti, R., F. Masulli (1990), *BFGS Optimization for Faster and Automated Supervised Learning*, INCC 90 Paris, International Neural Network Conference, pp 757–760.

2. Fletcher, R. (1975). *Practical Methods of Optimization*, John Wiley & Sons.

3. Gill, P.E., W. Murray, M.H. Wright (1980). *Practical Optimization*, Academic Press. inc.

4. Hestenes, M. (1980). *Conjugate Direction Methods in Optimization*, Springer Verlag, New York.

5. Hinton, G. (1989). *Connectionist Learning Procedures*, Artificial Intelligence **40**, pp 185–234.

6. Johansson, E.M., F.U. Dowla, D.M. Goodman (1990), *Backpropagation Learning for Multi-Layer Feed-Forward Neural Networks Using the Conjugate Gradient Method*, Lawrence Livermore National Laboratory, Preprint UCRL-JC-104850.

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

8. Lapedes, A., R. Farber (1986). *A self-optimizing neural net for content addressable memory and pattern recognition*, Physia **22 D**, pp 247–259.

9. Marr, D. (1982), *Vision*, W.H. Freeman and Company.

10. Mühlenbein, H. (1990). *Limitations of multi-layer perceptron networks - steps towards genetic neural networks*, Parallel Computing **14**, pp 249–260.

11. Møller, M.F. (1990), *Learning by Conjugate Gradients*, The 6th International Meeting of Young Computer Scientists, Czechoslovakia, in press.

12. Powell, M. (1977). *Restart Procedures for the Conjugate Gradient Method*, in: Mathematical Programming, pp 241–254.

13. Rumelhart, D.E., G.E. Hinton, R.J. Williams (1986). *Learning Internal Representations by Error Propagation*, in: Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Eds. D.E. Rumelhart, J.L. McClelland, MIT Press, Cambridge, MA., pp 318–362.

14. Schwartz, J.T. (1988). *The New Connectionism: Developing Relationships Between Neuroscience and Artificial Intelligence*, in: The Artificial Intelligence Debate. False Starts, Real Foundations, Eds. S.R. Graubard, MIT Press, Cambridge, MA., pp 123–143.

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

16. Vogl, T.P., J.K. Mangis, W.T. Zink, D.L. Alkon (1988). *Accelerating the Convergence of the Back-Propagation Method*, Biological Cybernetics **59**, pp. 257–263.

17. Watrous, R.L. (1987), *Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization*, Proc. IEEE 1st International Conference on Neural Networks **2**, pp 619–628.