IDIAP

Technical report

Neural Networks with Adaptive Learning Rate and Momentum Terms

M. Moreira and E. Fiesler

October, 1995

INSTITUT DALLE MOLLE D'INTELLIGENCE ARTIFICIELLE PERCEPTIVE
CASE POSTALE 609 - 1920 MARTIGNY - VALAIS - SUISSE
TELEPHONE : ++41 26 22.76.64 - FAX : ++41 26 22.78.18
E-MAIL : IDIAP@IDIAP.CH

Number: 95-04

1 Introduction

The backpropagation algorithm is the most popular method for neural networks training and it has been used to solve numerous real life problems. Nevertheless, it still presents some difficulties in dealing with sufficiently large problems.

The fact that it consists of a relatively simple procedure has played an important role in the success of the algorithm. In addition, its local computations avoid the use of large storage resources. These features, however, lead to small performance achievements. The standard backpropagation algorithm shows a very slow rate of convergence and a high dependency on the value of the learning rate parameter. Furthermore, the shape of the multi-dimensional error function for the majority of the applications usually presents irregularities difficult to handle by a gradient descent technique with a fixed step size. Another serious problem is the existence of regions with local minima. If the search procedure enters such a region, the minimization will be directed to that minimum and stop when it reaches it, while it would be desired that it continues towards a global minimum, or at least a sufficiently low one. It was discovered, however, that the appropriate manipulation of the learning rate parameter during the training process can lead to very good results and, hence, a large number of different methods for its adaptation have been proposed. The main purpose of this paper consists in an analysis of some of those proposals. They are classified based on the underlying techniques used for the parameter adaptation and a special concern is applied to those methods that are parameter-independent, that is, that do not imply the need for the user to tune parameters whose values exert influence on the performance of the algorithm depending on the particular problem handled and network architecture used. The results of simulations performed with those methods are then presented. Among the methods reviewed here, some perform the adaptation of the momentum term, either by itself or in parallel with learning rate adaptation.

1.1 The Backpropagation Algorithm

Backpropagation (henceforth called BP) is a training procedure for feedforward neural networks that consists in an iterative optimization of a so called *error function* representing a measure of the performance of the network[1]. This error function E is defined as the mean square sum of differences between the values of the output units of the network and the desired target values, calculated for the whole pattern set:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{N_L}(t_j - a_j)^2 \ .$$

t_j and a_j are the target and actual response values of output neuron j and N_L is the number of output neurons; L being the number of layers. In order to allow consistent comparisons, some authors use the mean squared error in such a way that the obtained value of the error will not depend on the size of the pattern set and number of output neurons of the specific network used and simulation performed. In this case, the summation of the equation above is divided by $N_L \cdot P$.

During the training process a set of pattern examples is used, each example consisting of a pair with the input and corresponding target output. The patterns are presented to the network sequentially, in an iterative manner; the appropriate weight corrections being performed during the process to adapt the network to the desired behaviour. This iterating continues until the connection weight values allow the network to perform the required mapping. Each presentation of the whole pattern set is named an *epoch*. Hereafter, the term "iteration" will refer either to a pattern presentation or a to a complete epoch, depending on the context.

The minimization of the error function is carried out using a gradient-descent technique. The necessary corrections to the weights of the network for each iteration n are obtained by calculating the partial derivative of the error function in relation to each weight w_{ij}, which gives a direction of steepest descent. A gradient vector representing the steepest increasing direction in the weight space is thus obtained. Due to the fact that a minimization is required, the weight update value Δw_{ij} uses the negative of the

[1] A brief description of the algorithm is given here. For a more detailed explanation of its derivation and theoretical basis, see for example [Rumelhart-86.2] or [Haykin-94].

corresponding gradient vector component for that weight. The *delta rule* determines the amount of weight update based on this gradient direction along with a step size:

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} \ .$$

The η parameter represents the step size and is called the *learning rate*. w_{ij} represents the connection weight from unit i in layer l to unit j in layer $l+1$.

Each iteration of the algorithm is composed of a sequence of three steps. i) The forward pass. It consists in presenting a pattern example to the inputs of the network and make it propagate sequentially through all the neuron layers until a result is obtained at the output units. The activation value a_j of unit j in layer l ($2 \leq l \leq L$) is calculated using a sigmoid activation function f:

$$a_j = f(\text{net}_j) \equiv \frac{1}{1+e^{-\text{net}_j}} \qquad \text{net}_j = \sum_{i=1}^{N_{l-1}} w_{ij} a_i + \theta_j \ .$$

Each i represents one of the units of layer $l-1$ connected to unit j and θ_j represents the bias or w_{0j}. ii) The *generalized delta rule* is used to calculate the values of the local gradients. Each weight update is defined as

$$\Delta w_{ij}(n) = \eta \ \delta_j a_i$$

and the equations of the generalized delta rule used to calculate the δ values are:

$$\delta_j = a_j(1-a_j)(t_j - a_j) \qquad \text{for output neurons,}$$

$$\delta_j = a_j(1-a_j)\textstyle\sum_{k=1}^{N_{l-1}} \delta_k w_{kj} \qquad \text{for hidden neurons.}$$

The δ_j for the output units can be calculated using directly available values, since the error measure is based on the difference between the desired (t_j) and actual (a_j) values. However, that measure is not available for the hidden neurons. The solution is to backpropagate the δ_j values layer by layer through the network. iii) Finally, the weights are updated.

Two variants exist in presenting the pattern examples. With *on-line training*[2], all of the above three steps are performed sequentially in each iteration. If, on the contrary, *batch training* is used, only the first two steps are performed at each iteration, the third being performed once per epoch, using the sum of the collected local gradient values for all the iterations in that epoch.

A *momentum* term was introduced in the BP algorithm by [Rumelhart-86.2]. The idea consists in incorporating in the present weight update some influence of the past iterations. The delta rule becomes

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} + \alpha \Delta w_{ij}(n-1) \ .$$

α is the momentum parameter and determines the amount of influence from the previous iteration on the present one. The momentum introduces a "damping" effect on the search procedure, thus avoiding oscillations in irregular areas of the error surface by averaging gradient components with opposite sign and accelerating the convergence in long flat areas. In some situations it possibly avoids the search procedure from being stoped in a local minimum, helping it to skip over those regions without performing any minimization there. Momentum may be considered as an approximation to a second-order method, as it uses information from the previous iterations[3]. In summary, it has been shown to improve the convergence of the BP algorithm, in general. Furthermore, it is possible that it allows a range of different learning rate values to produce approximately analogous convergence times.

[2] Also called *stochastic* training

[3] See Section 2.2.1 for more information about this issue.

2 Adaptive Learning Rate and Momentum Methods

There are two fundamental reasons that justify a study of adaptive learning rate schedules. One is that the amount of weight update can be allowed to adapt to the shape of the error surface at each particular situation. The value of the learning rate should be sufficiently large to allow a fast learning process, but small enough to guarantee its effectiveness. If at some moment the search for the minimum is being carried out in a ravine, it is desirable to have a small learning rate, since otherwise the algorithm will oscillate between both sides of the ravine, moving slowly towards the real descent direction. This is justified by the fact that the gradient will point towards the current descent direction, which will be far different from the direction of the minimum, except for the case where the current search point is on the bottom of the valley. If, on the contrary, the search is in the middle of a plateau and the minimum is still a long distance away, then an increase of the step size will allow the search to move faster, since in this case there exists a sufficiently accurate descent direction. The other reason is that with automatic adaptation of the learning rate, the trial-and-error search for the best initial values for the parameter can be avoided. Normally, the adaption procedures are able to quickly adapt from the initial given values to the appropriate ones.

In addition to learning rate adaptation, certain methods use also an individual learning rate for each connection weight, the adaptation being processed independently to each one of them, thus allowing each weight dimension to be autonomously adapted based on its own local information. In this case, however, the descent direction obtained at each step no longer corresponds to steepest descent, since an independent direction is found for each weight dimension. This technique is named as *local learning rate adaptation* as opposed to *global learning rate adaptation*, where a single global value is used for all weights.

Besides learning rate adaptation, momentum adaptation is also used in some methods. The main reason for this is that the influence of the momentum in the weight update can become too large, in cases where the gradient value is small.

In the course of this study, a large number of methods for learning rate and/or momentum adaptation are reviewed and their main characteristics summarized.

2.1 Taxonomy of the optimization techniques

The studied adaptive methods are presented here following a classification based on the different techniques used to adapt the learning rate. The list of techniques presented is not exhaustive.

- Based on numerical optimization procedures using second-order information:
    - Conjugate gradient
    - Quasi-Newton
    - Using a second-order calculation of the step size
- Based on Stochastic Optimization
- Heuristic-based:
    - Adaptation based on the angle between gradient direction in consecutive iterations
    - Adaptation based on the sign of the local gradient in consecutive iterations
    - Adaptation based on the evolution of the error
    - Prediction of a set of new values for the learning rate
    - Searching for zero-points of the error function instead of zero-points of its derivative
    - Adaptation using the derivative of the error function in relation to the learning rate
    - Using peak values for the learning rate
- Other:
    - Calculation of the optimal fixed values for the parameters before the training

2.2 Description of the methods

2.2.1 Methods based on numerical optimization procedures using second-order information.

Numerical optimization techniques are, in general, capable of achieving high rates of convergence. Nevertheless, when applied to the training of feed-forward neural networks, their fast convergence features do not allways imply fast training, either because of incompatibilities between the nature of the particular problem handled and the properties of the technique or because the reduction in the number of iterations will not compensate, in terms of processing time, for the computational complexity of those techniques, specially in large-scale problems. Nevertheless, they are still worth considering since some approximation techniques were derived that allow a successful use of second order information in the neural networks field. These techniques still capture the main principles of the original procedures but reduce considerably their processing demands.

The use of second-order information provides a better knowledge of the characteristics of the error function surface and thus optimizes the search procedure. Although the methods described in this section are all based on second-order techniques they never use explicit information from the second-order derivatives since they are all based on approximations. Two main categories of methods are distinguished: the Quasi-Newton and the conjugate gradient ones. These have been reported as the most successfully applied to the training of feed-forward neural networks, amongst all those using second-order information. Additionally, a reference is given to a technique that uses second-order information in a different manner.

When numerical optimization techniques are applied to the training of multilayer perceptrons, the weights are considered to be the independent variables of the objective function to be optimized, which is the above defined error function. Although the weights are distributed over the network, they are considered here as a single one-dimensional vector of W independent variables.

Conjugate gradient Conjugate gradient (CG) methods find the search direction at each iteration based on a linear combination between the current gradient direction and the previous one, where steepest descent is used as the direction for the first iteration. The original CG algorithm, based on conjugate directions theory applied to quadratic functions, makes use of the Hessian matrix H of second-order derivatives, and its inverse, in two basic operations: the calculation of the step size and the linear combination of the previous and present gradient directions to find the new search direction. The calculation and inversion of an $W \times W$ matrix, W being the total number of weights in the network, are two computationally expensive procedures to be performed at each iteration during the training of feedforward neural networks. Besides this, it involves the explicit calculation of the second-order derivatives, which are not directly obtainable by backpropagation. Considering the inconvenience of manipulating the Hessian, an approximation to it was derived[4] to replace the true matrix in the calculation of the conjugate direction, while the step size is obtained by a line search. With this, a simplified version of the algorithm that depends only on the function and gradient values is obtained which is considered to be the standard conjugate gradient as applied to the training of feed-forward neural networks. As it is only briefly outlined here, the derivation and formal description of the conjugate direction principles can be found in [Johansson-92] or [Moller-93]. In this section, the vector containing the summation of the negative gradients vectors for all the pattern presentations in epoch k $(-\nabla E(w_k))$ will be denoted as g_k.

1. Initialization. The weight vector w_0 is set. The initial direction d_0 is obtained by gradient descent (g_0). $k = 0$.
2. A line search is performed to find the η_k that minimizes $E(w_k + \eta_k d_k)$.
3. The weight vector is updated: $w_{k+1} = w_k + \eta_k d_k$.
4. A new direction d_{k+1} is computed. If $(k+1 \mod W)$=0 then the algorithm is restarted with $d_{k+1} = g_{k+1}$. Otherwise $d_{k+1} = g_{k+1} + \beta_k d_k$.
5. If the minimum was reached then the search is terminated. Otherwise, a new iteration is performed: $k = k + 1$ and jump to step 2.

[4] See [Johansson-92] for details on its derivation.

The β_k parameter is always calculated in order to force the consecutive directions to be conjugate. Different formulas exist to calculate it, such as the following:

Fletcher-Reeves:

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$$

Polak-Ribière:

$$\beta_k = \frac{g_{k+1}^T [g_{k+1} - g_k]}{g_k^T g_k}$$

Hestenes-Stiefel:

$$\beta_k = \frac{g_{k+1}^T [g_{k+1} - g_k]}{d_k^T [g_{k+1} - g_k]}$$

When applied to quadratic functions, conjugate gradient methods converge in at most W iterations. As the function used here is not quadratic, the algorithm is restarted in the gradient descent direction at the current point after each W iterations, otherwise a deterioration in the conjugacy of the directions occurs.

The line search referred in step 2 is an iterative procedure that executes a one-dimensional search for the minimum along the search direction determined by conjugate gradient. It is a computationally expensive procedure caused by the fact that the expression $E(w_k + \eta_k d_k)$ must be calculated at each search iteration for different values of η, which involves the presentation of all the training set for each calculation. The search tolerance parameter, that must be selected by the user, plays a very important role in the global convergence time of the algorithm, since it influences the number of line search iterations performed until a near-optimal step size is found. Furthermore, the accuracy of the line search is crucial for the success of the conjugate gradient algorithm.

The fact that gradient descent methods use the steepest descent direction in all cases leads to the fact that in subsequent search iterations those directions can be quite different, caused by the locality of the technique. One important consequence of this is that the direction taken in one iteration can influence negatively the optimization already achieved with previous iterations. Conjugate gradient directions help avoiding this zig-zag movement detected on steepest-descent searches.

In Section 1.1 it was mentioned that the momentum technique is in principle similar to conjugate gradient. In both cases, information from previous iterations is used. In the previously described standard BP algorithm the momentum parameter α determines the influence from the past iteration direction in the current one. A similar purpose is assigned to the β parameter of CG, that is, to combine the direction used in the previous iteration with the current descent direction to find a new one. However, the main difference between the two techniques lies in the fact that, while there is always a well-defined prescription to find β in CG, even if it varies between different algorithm versions, there is no established procedure to set the value of momentum, and either a user-defined constant value is used or a particular procedure is applied to adapt it. Based on this, momentum is never used in a CG method, except in the case where it is considered that that CG method itself consists of a momentum adaptation procedure - see the description for [Leonard-90].

In practice, conjugate gradient methods are only used in batch mode. One reason for this is their large computational demands. Another reason relates to the fact that their optimization procedure should consider the whole training set, otherwise, optimizations done on single or small groups of pattern presentations could be in conflict to the optimization of the rest of the set. M. Moller has studied this problem. The results can be found in [Moller-93].

[Leonard-90] Leonard and Kramer emphasize the parallelism between momentum and CG, as discussed above, and even describe the principle of the latter based on the former. In this method, the β parameter existent in the above description of the CG algorithm is treated as being the momentum term. It is calculated using the Fletcher-Reeves formula. The direction is also reset to gradient descent every W iterations. It is stated that a line search is used to calculate the step size, although no details are given about the exact procedure. The authors say that a *coarse* line search is used, instead of an *exact* one and, because of that, a negative (ascending) step may be taken sometimes. In such a case, the algorithm is restarted using a momentum value of zero. As mentioned the tolerance parameter for the line search must be set by the user.

[Moller-93] With the *Scaled Conjugate Gradient* method, Møller proposes a technique to regulate the indefiniteness of the Hessian matrix, based on a model-trust region approach obtained from the Levenberg-Marquardt algorithm. This technique uses a parameter λ to adjust H and assures that it is positive-definite at each iteration, avoiding the need to perform a line search. Instead, an approximation of the Hessian is used to calculate the step size, as is done with the standard CG algorithm. The matrix is approximated by:

$$s_k = \nabla^2 E(w_k)\ d_k \approx \frac{\nabla E(w_k + \sigma_k d_k) - E(w_k)}{\sigma_k}, \qquad \sigma_k = \frac{\sigma}{\mid d_k \mid}\ .$$

The value of σ is said not to influence the behaviour of the algorithm as long as it is smaller than 10^{-4}, but it should be as small as possible, considering the machine precision. The quality of the approximation obtained for the Hessian is measured in every iteration, the effective calculation of the step size depending on that. Thus, if the measured quality is not satisfying, the λ parameter is adjusted and the step size calculation and weight updating procedures are skipped. All the parameters involved in this method are precisely given the values (or ranges of values) that they should assume.

Quasi-Newton Modelling the incremental change of the objective function between iterations with a Taylor series expansion, the Newton method can be derived as follows:

$$\begin{aligned} \Delta E(w) &= E(w + \Delta w) - E(w) \\ &\approx g^T \Delta w + \tfrac{1}{2} \Delta w^T H \Delta w \end{aligned} .$$

After setting the differential with respect to Δw to zero, to minimize $\Delta E(w)$, the equation for the weight update becomes:

$$\Delta w = -H^{-1} g\ .$$

The use of the Hessian in this equation provides a direction as well as a step size for the optimization step. As can be observed, the Newton method is fully based on the use of the Hessian matrix. [Battiti-92] describes some techniques that deal with analytical approximations of the Hessian. As already discussed, it is not practical to approximate the matrix analytically. Because of this, only Quasi-Newton methods will be considered here, these being the ones that are commonly applied to train multilayer perceptrons. The main feature of this class of methods lies in the fact the Hessian is obtained iteratively, based on information from the function and its derivative accumulated during the iterations. Hence, the second-order derivative is obtained by any update procedure that verifies the *Quasi-Newton condition*:

$$H_k\ (w_k - w_{k-1}) = \nabla E(w_k) - \nabla E(w_{k-1})\ .$$

There are several methods to derive H_k respecting this condition. The *BFGS update* developed by Broyden, Fletcher, Goldfarb and Shanno in 1970 is one example. A description of it is given below, since the method proposed in [Battiti-92] is based on it. Another example is the *DFD update method*, proposed by Davidon, Fletcher and Powell in 1963 (see [Watrous-87]).

For the Newton method to be applied, the Hessian has to be both positive definite and symmetric, otherwise numerical instability problems may arise. Furthermore, convergence is only guaranteed if a good initial estimate of the minimum in the convex region where it is located already exists. In this case, considering the non-quadratic error function, the method converges quadratically. Taking into account that a considerable effort must be done by the search procedure before it arrives at the region where the minimum is located, and that the Newton method is accurate only locally in that region, it must be combined with some global search procedure. Similar to the methods derived from conjugate gradient, the derivations of the Newton method, which do not use direct second-order derivatives calculation, find the step size by means of a line search. With this feature the Quasi-Newton methods become global.

[Battiti-89] [Battiti-92] Battiti first presented his method in 1989 under the name of *conjugate gradient with inexact linear searches* based on a procedure proposed by D. F. Shanno (1978) with the same name. By that time, the method was described based on the principles of conjugate gradient. More recently, in [Battiti-92], the same method is derived from the Quasi-Newton BFGS update used to approximate the Hessian matrix and it is given the definitive name of *OSS method* (One-Step Secant). This shows that although conjugate gradient and the Newton method originally used second-order information differently, the modifications operated to adapt each of them to the training of multilayer perceptrons allows methods derived from either of them to have similar features. In this case, the same method was presented derived from both techniques. In any of the cases, this is a second-order method and it is described here following the BFGS derivation.

The BFGS update is considered to be one of the must successful procedures to iteratively approximate H. At each iteration, the already existent matrix approximation is updated using the new information obtained from the objective function and its derivative. The identity matrix is normally used as H_0, which means that gradient descent direction is used in the first iteration. Considering $y_k = g_k - g_{k-1}$ and $p_k = w_k - w_{k-1}$ the equation of the BFGS update is:

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k} \ .$$

This formula guarantees the positive definiteness of the Hessian.

Although it consists of a simplification, the above formula is still considerably complex, demanding much computational effort and storage space if a large number of weights are considered. The solution proposed for this problem was to invert the equation and obtain an equation for the inverse of H with the identity matrix I being used instead of the approximated H obtained in the previous iteration. With this, no information is transferred between iterations and the method is called *one-step memory-less Broyden-Fletcher-Goldfarb-Shanno.* Using the described transformations, the new search direction is obtained by:

$$d_k = -g_k + A_k p_k + B_k y_k \ , \ \text{where}$$

$$A_k = -\left(1 + \left(\frac{y_k^T y_k}{p_k^T y_k}\right)\right)\left(\frac{p_k^T g_k}{p_k^T y_k}\right) + \left(\frac{y_k^T g_k}{p_k^T y_k}\right) \ , \ \text{and} \ B_k = \left(\frac{p_k^T g_k}{p_k^T y_k}\right) .$$

The search direction is reset to gradient descent every W iterations. The line search procedure proposed in [Battiti-89] to calculate the step size is based on quadratic interpolation and is said to be optimized.

[Becker-89] Becker and le Cun propose a method based on previous work by le Cun. It consists of approximating the Hessian by its diagonal. The use of a diagonal matrix allows them to perform matrix manipulations that would otherwise be too computationally expensive. They use a procedure that calculates the second-order derivatives of the error function:

$$H = \frac{\partial^2 E}{\partial w_{ij}^2} = \frac{\partial^2 E}{\partial a_j^2} f(\mathrm{net}_i)^2 .$$

The second-order derivatives of the error in relation to the activations $\left(\frac{\partial^2 E}{\partial a_j^2}\right)$ is obtained explicitly using a recursive procedure similar to backpropagation. The obtained Hessian, being diagonal, is easy to invert and can thus be used in a weight update rule similar to the original Newton step:

$$\Delta w = -\eta \ \frac{\frac{\partial E}{\partial w_{ij}}}{\left|\frac{\partial^2 E}{\partial w_{ij}^2}\right| + \mu}$$

The aim of using the absolute value of the Hessian term is to reverse the sign of the step for directions with negative diagonal terms. The addition of μ to the Hessian guarantees its positive-definiteness. No line search is performed, instead, a fixed learning rate is used with a different value for each layer, calculated relatively to the fan-in of each unit of the corresponding layer. Momentum is used. The authors conclude that the behaviour of their method is considerably sensitive to the values of μ and η. They performed a study involving a set of simulations to analyze the quality of the

Hessian's diagonal approximation. Considering the error surface regions where the behaviour of a training method is more critical, such as ravines (valleys), the diagonal becomes more similar to the real matrix as those ravines become parallel to one of the weight axes. This is due to the fact that the diagonal does not produce any rotation effect on the descent direction, as occurs with the full matrix of the original Newton method. As the effect of the diagonal consists simply of scaling the step size, if operating in a ravine nearly parallel to one of the weight axes, the behaviour will be based in approximately the same curvature information obtained with a full Hessian.

[Fahlman-89] The Quickprop method of Fahlman is included in this section because it uses a second-order rationale close to that of the Newton method, although it is mainly heuristic. Quickprop is composed of a set of heuristics. The main principle is to perform independent optimization for each weight, minimizing an approximation of its curve by a parabola whose arms turn upwards. At each step, the parabola is minimized using the inclination of the corresponding weight dimension of the error surface in the current and previous step. With this, the update rule is

$$\Delta w_{ij}(n) = \frac{\nabla E(w_n)}{\nabla E(w_{n-1}) - \nabla E(w_n)} \Delta w_{ij}(n-1) \ .$$

There is no learning rate in this equation. However, the step size should still be controled when the parabola determines a too large or infinite step, which is done with a "maximum growth factor" μ that determines $\Delta w_{ij}(n) \ \leq \ \mu \ \Delta w_{ij}(n-1)$. A value of $\mu = 1.75$ is suggested. In the first step, and for the cases where the previous weight update is 0, the method uses the standard delta rule, which means using simple gradient descent along with learning rate. Later, the author thought of combining both update rules which results in always adding the delta rule update to the quadratic rule mentioned above, except when the sign of the steepness in the current step is opposite to the previous one. In this case, the quadratic rule is used alone. A further improvement consists of adding a weight decay term to each weight to prevent it to grow exceedingly and cause overflow. Finally, the *flat-spot* elimination technique is proposed. The flat-spot problem occurs when the output a_j of the sigmoid activation function of some neuron j approaches 0.0 or 1.0. In this case, the function derivative $a_j(1-a_j)$ becomes too close to zero leading to a small value in the δ_j term of the generalized delta rule mentioned in Section 1.1 and producing a too small weight update. The technique consists in always adding a constant of 0.1 to the derivative of a_j, yielding $a_j(1-a_j)+0.1$. Quickprop is still under development.

Using a second-order calculation of the step size As already mentioned, line search is a procedure used to find the optimal η^* that minimizes the function h defined as:

$$h(\eta_k) \equiv E(w_k + \eta_k d_k).$$

Given this, the error at the present iteration is expressed as an univariate function of the learning rate value. The learning rate adaptation performed by this technique is based on the optimization of $h(\eta_k)$, which can be accomplished using traditional one-dimensional numerical optimization techniques that make use of the first and second-order derivatives of the function.

[Yu-95] Based on some experiments, Yu, Chen, and Cheng assume that the function $h(\eta_k)$ can be approximated by a parabola in the region where η_k^* is located. Then, a set of recursive formulas is suggested to calculate the first and second-order derivatives of $h(\eta_k)$ as well as a set of different techniques to find η_k^* using these derivatives, including *Linear Expansion*, *Acceptable Line Search*, *Polynomial Approximation*, and the *Newton Method*. Computer simulations were done to evaluate the performance of each of them, the Newton method being suggested as the most appropriate one. The step size is then obtained in one single iteration if the Newton step is applied:

$$\eta_k^* = -\frac{h'(0)}{h''(0)}.$$

It is mentioned that when $h''(0) \geq 0$, the Newton method is not applicable. In this case, either the Acceptable Line Search technique or the Polynomial Approximation are used, without details given on how to make the selection. A momentum adaptation technique was developed to be used in parallel with the learning rate adaptation. This technique is inspired by the principles of the conjugate gradient method but includes information from the learning rate value. The formula is

$$\alpha_k = \frac{\eta_k^* \beta_k}{\eta_{k-1}^*} ,$$

where η_k^* corresponds to the value for the learning rate obtained in that same iteration. The value of β_k is calculated by the following expression, where $E(w_k)$ is represented by E_k for simplicity:

$$\beta_k = \begin{cases} \frac{[\nabla E_k - \nabla E_{k-1}]^T \nabla E_k}{\|\nabla E_{k-1}\|^2}, & \text{if } k \neq rW^r \text{ or } \mid \nabla E_k^T \nabla E_{k-1} \mid \leq 0.2 \, \|\nabla E_k\| \, \|\nabla E_{k-1}\| \\ 0 & \text{otherwise.} \end{cases}$$

W^r is the largest possible period for restarting the direction search, but no explicit value is given for r, which is a positive integer. A rescue procedure is applied when, during the training, the value of the error between iterations increases beyond a small positive threshold. This is aimed at avoiding the search procedure to jump into undesirable regions of the error surface. In this case the learning rate is recalculated using the Acceptable Line Search technique or a fixed value is used.

2.2.2 Methods based on Stochastic Optimization

While the deterministic training procedures perform local minimizations of the objective function based on the currently available information about that function, the training of feedforward neural networks based on stochastic approximation is considered as a statistical process. It regards the training patterns as random samples from an unknown distribution, the main task being the search for a solution w^* that minimizes the objective function by accumulating information about the distribution given by the successive presentation of the exemplars, which are chosen randomly from the training set in an on-line supervised manner. Thus, each pattern presentation represents an observation of the probabilistic relationship between the set of input patterns and the set of all their corresponding target outputs. The stochastic procedures have a similar behaviour as the deterministic ones on average. Their random nature, however, introduces some noise in the training that can entail positive effects, like the ability to escape from local minima.

The following are the conditions imposed on the learning rate for convergence, in the mean square sense, to a minimum:

$$\lim_{m \to \infty} \sum_{n=1}^{m} \eta_n = \infty \qquad \lim_{m \to \infty} \sum_{n=1}^{m} \eta_n^2 < \infty$$

To ensure these, the learning rate is generally taken as a function of time, following a decreasing sequence. The most commonly used function in stochastic approximation in accordance with this has been $\eta_n = 1/n$, where n is the iteration number. However, it has been discussed that this schedule is far from achieving the optimal rate of convergence, and alternative schedules are proposed in the methods presented below. Namely, one solution to improve the rate of convergence, using an annealing schedule for the learning rate, is to keep its value high until the region of the minimum is found and only then apply the decreasing schedule to allow convergence, leading to two different stages during the search where distinct regimes are adopted for the learning rate.

[Darken-92] Darken and Moody propose the "Search Then Converge" (STC) schedule:

$$\eta_n = \eta_0 \frac{1 + \frac{c}{\eta_0} \frac{n}{\tau}}{1 + \frac{c}{\eta_0} \frac{n}{\tau} + \tau \frac{t^2}{\tau^2}}$$

The parameter τ corresponds to the point in time (iteration) when to switch from the *search phase* to the *convergence phase*. The learning rate remains fairly constant and equal to η_0 until time τ is reached, thereafter being decreased as c/n. The c parameter is of major importance to the convergence behaviour of this schedule. Defining $c^* \equiv 1/2\lambda$, where λ is the smallest eigenvalue of the Hessian matrix of the objective function obtained in the location of the optimum, two very different convergence behaviours are detected when $c < c^*$ and when $c > c^*$, the later condition being crucial for the success of the convergence. As λ is unknown, the authors propose an on-line procedure to evaluate if c is either less or greater than c^*, based on what they call "drift", defined as an excessive correlation in the change vectors of that parameter. Two different ways to quantify the drift are proposed. The authors expect to obtain an entirely automatic method using the principles explained here. However, the work they developed in this field is not yet concluded. In fact, no explicit quantified relation between the drift and the precise value that c should assume is given.

[Leen-94] Leen and Orr study the case where momentum is used in parallel with the learning rate schedule $\eta_n = \eta_0/n$. They refer the fact that when momentum is used, the learning at late times is determined by a factor

$$\eta_{eff} \equiv \frac{\eta}{1-\alpha},$$

called the *effective learning rate*. After studying the asymptotic behaviour of the error function in the presence of the stochastic methods with momentum, they found that similar to the method described above, two very different convergence behaviours were observed depending on whether η_{eff} is less or greater than a critical value equivalent to the c^* parameter described in the method above. As mentioned, η_{eff} must be greater than the critical value to allow optimal asymptotic convergence. The solution found to overcome the fact that no knowledge exists about the eigenvalue λ was to develop a momentum adaptation technique that assures optimal convergence independently of the value of η_0. The formula for the adaptive momentum matrix is:

$$\alpha = I - c\,x\,x^T, \qquad c = \min(\eta_0\,,\,1/(x^T x)).$$

I is the $W \times W$ identity matrix and x the vector containing the input values of the network. The optimization effort developed here concerns mainly the convergence phase of the learning. The author's research work in this field is not yet finished, since the method includes no prescriptions to decide when to switch from the constant parameter phase to the adaptive one.

2.2.3 Heuristic methods

Adaptation based on the angle between gradient directions in consecutive iterations. Considering the previous weight update $\Delta\vec{w}_{n-1}$ and the present gradient descent $\nabla\vec{E}(w_n)$ vector directions, the value of the angle θ between them in successive iterations can give information about the properties of the error surface. If those two vectors have similar directions, indicating stability of the search procedure, the value of the learning rate can be increased. On the contrary, a noticeable difference between those directions, detected by a change in the angle value between iterations, suggests the presence of an irregular region of the error surface, a situation where the value of the learning rate should be reduced. The angle information is obtained through the calculation of its cosine, using the formula:

$$\cos\theta_n = \frac{-\nabla\vec{E}(w_n).\Delta\vec{w}_{n-1}}{||\nabla\vec{E}(w_n)||\;||\Delta\vec{w}_{n-1}||}.$$

It should be noted that the angle between $-\nabla\vec{E}(w_n)$ and $\Delta\vec{w}_{n-1}$ is equal to the angle between $\Delta\vec{w}_n$ and $\Delta\vec{w}_{n-1}$ only when momentum is not used, otherwise, the effect of α in the learning rule, combined with the direction of the previous gradient vector $-\nabla\vec{E}(w_{n-1})$ originate a weight update $\Delta\vec{w}_{n-1}$ that follows a direction different from the gradient one.

[Chan-87] Chan and Fallside use the angle-driven approach. The value of the learning rate for the present iteration is obtained by combining its value in the previous iteration with the present value of $\cos\theta$, giving

$$\eta_n = \eta_{n-1}\left(1 + \frac{1}{2}\cos\theta_n\right).$$

In parallel, they propose the adaptation of the momentum coefficient, based on the fact that it should be proportional to the value of the learning rate $\alpha_n = \lambda_n \eta_n$. The adaptation is, in fact, performed through the proportionality factor with $\lambda_n = \lambda_0(\|\nabla E(w_n)\|/\|\Delta w_{n-1}\|)$, having $\lambda_0 = \alpha_0/\eta_0$ and $0 \leq \lambda_0 < 1$. These initial values of learning rate and momentum should be selected by the user. The authors state that the algorithm is relatively insensible to these initial values, and support it by some simulation results. They use batch update - but with a large number of patterns, the training set is carefully divided into subsets (respecting an uniform distribution) and the weights are updated after the presentation of each subset. Global values are used for learning rate and momentum.

[Haffner-89] Haffner did some experiments while modifying in different aspects of the BP algorithm. The proposed improvements concerning the step size include an adaptation technique based on $\cos\theta$, where

$$\eta_n = \eta_{n-1}\, \mathrm{e}^{u \cos\theta_n}.$$

An empirically obtained value of 0.1 is suggested for u. Further proposals include the use of local learning rate and the limitation of the norm of the vector $\eta\nabla\vec{E}(w_n)$ to avoid weight modifications that are too large. This is done by resizing the learning rate with

$$\eta' = \frac{\eta}{1 + \frac{\eta}{\omega}\sum_{w \in W}\left(\frac{\partial E}{\partial w}\right)^2}$$

and using η' in the weight update rule. $\omega = 1.0$ was empirically found to work well for different problems. Additionally, a momentum adaptation schedule determines an independent momentum term α_{ij} for each weight, setting $\alpha_{ij} = 1/(1 + d \mid \sum_{w \in W} w\Delta w \mid)$ with a proposed value of $d = 1.0$.

[Hsin-95] This is an on-line learning rate adaptation method where each iteration makes use of information from the previous ones. The learning rate at each iteration is obtained by a weighted average over the values of $\cos\theta$ from the current and last L iterations. A formula is given to calculate the relative weights for the average. L is an arbitrary parameter that is changed for each problem. A momentum term is not considered and the learning rate parameter is global.

Adaptation based on the sign of the local gradient in consecutive iterations. After performing a detailed analysis of the reasons for the slow rate of convergence of the BP algorithm, R. Jacobs [Jacobs-88] proposed a set of four heuristics aimed at improving its convergence. They influenced the work of several other researchers that incorporated them in their proposed techniques for BP optimization. In a concise description, the heuristics state that each weight should have its own learning rate that should be allowed to vary over time. Furthermore, the learning rate should be increased when the corresponding local gradient holds the same sign for several iterations, and decreased when consecutive changes in the sign are detected. The principles used for varying the learning rate values are based on the fact that the behaviour of the sign of the local gradient throughout consecutive iterations can give information about the curvature of the error surface in the corresponding weight dimension.

[Jacobs-88] In order to implement the proposed heuristics, Jacobs developed, with the cooperation of R. Sutton, the *delta-bar-delta (D-B-D)* learning rule, that consists in varying the learning rate based on a comparison between the sign of the current local gradient and that of an exponential average of the same gradient in the present and several past iterations. In addition, the learning rate is incremented by a constant but decremented by a proportion, which results in a faster decrease than increase rate. As proposed in the heuristics, independent learning rates and batch updating are used in the simulations described. A momentum factor is not used. Instead, it is presented in the paper as one of the possibilities to implement the heuristics. This technique has four tunable parameters without precisely defined values: initial learning rate value, the constants for learning rate increasing and decreasing, and the base for the exponential average. [Haykin-94], in Sec. 6.15, proposes the introduction of the *gradient reuse technique* [Hush-88] into the Delta-Bar-Delta learning rule in order to reduce its computational effort. In this case too, batch updating is used.

[Minai-90] Minai and Williams based their work on the Delta-Bar-Delta rule, developing the Extended-Delta-Bar-Delta (E-D-B-D) which consists mainly of four modifications of the D-B-D learning rule. The features added or modified by E-D-B-D are:

1. instead of increasing the learning rate by adding a constant to it, an exponentially decreasing function of $|\bar{\delta}(t)|$ is used to increase it, $\bar{\delta}(t)$ being the exponential average of the local gradient values δ in present and past iterations;
2. a momentum term is used, being adapted exactly the same way as the learning rate;
3. upper limits are imposed for the value of these two parameters;
4. a special kind of backtracking procedure is used that memorizes the point where the lower error was found during the search and returns to that point whenever the actual error value exceeds it by a certain amount, called the tolerance. In such cases, the learning rate and momentum values are decreased.

Batch updating is used, as well as independent learning rate and momentum coefficients for each weight. The E-D-B-D technique adds seven more user-defined parameters to D-B-D, resulting in a total of eleven.

[Almeida-90] [Silva-90] Silva and Almeida propose a simplified implementation of the principles described in this section. The learning rate is increased multiplying it by a constant u slightly greater than 1 (= 1.2) and decreased multiplying it by the reciprocal of u (1/1.2). Independent learning rates values are used. Momentum also takes part in the method, but it is held constant. A backtracking procedure is performed in the cases where an increase of the error function is detected, making the search return to the point where it was in the previous iteration. In this case, the normal adaptation of the learning rate values is still performed; however, if the error increases in three consecutive iterations, all the independent parameters are reduced by half. Upper and lower bounds are defined for the step size, in order to prevent overflow and underflow. Batch updating of the weights and learning rate is used.

[Riedmiller-93] This is not an adaptive learning rate method. Referring to the learning rate adaptation techniques in general, Riedmiller and Braun state that, as the weight update value is composed both of the adapted learning rate and the value of the derivative, the effect of the former can in some cases be disturbed by the unforeseeable values of the latter, making the recently performed learning rate adaptation useless. They therefore propose a method, called RPROP using a technique that, although based on the sign of the local gradient in consecutive iterations, differs considerably from all the other adaptation techniques and even from the standard BP algorithm itself, since the weight update is done directly, without using the derivative nor the learning rate. That means that at each iteration, a certain quantity Δ_{ij} called *update value* is added to each weight. The value of Δ_{ij} is adapted by the gradient sign technique, as it would be done with a learning rate parameter:

$$\Delta_{ij}(n) = \begin{cases} 1.2 * \Delta_{ij}(n-1) & \text{if } \frac{\partial E(n-1)}{\partial w_{ij}(n-1)} \frac{\partial E(n)}{\partial w_{ij}(n)} > 0 \ , \\ 0.5 * \Delta_{ij}(n-1) & \text{if } \frac{\partial E(n-1)}{\partial w_{ij}(n-1)} \frac{\partial E(n)}{\partial w_{ij}(n)} < 0 \ , \\ \Delta_{ij}(n-1) & \text{otherwise.} \end{cases}$$

Furthermore, if the current local gradient is positive, the update-value is subtracted to the weight, being added to it if the gradient is negative. With a gradient of zero, the weight remains unchanged. $\Delta_{ij}(0) = 0.1$ is the proposed initialization, although it is stated that this value is not influent in the general behaviour of the algorithm. Further details of the method include the reversal of the previous weight update in the case where the derivative changes sign and the existence of upper and lower bounds for the update-value, precisely defined by the authors. No parameters are defined by the user, each weight has its own independent update-value, and batch updating is used.

Adaptation based on the evolution of the error. Considering the simple principle that the search for a minimum should be accelerated when following a descending movement in the error surface and slowed down when ascending, the learning rate adaptation can be done simply by observing the evolution of the value of the error at each iteration. In particular, the learning rate can be increased if the current error is smaller than in the previous iteration and decreased otherwise.

[Vogl-88] The team composed of Vogl, Mangis, Rigler, Zink, and Alkon proposed the principle described above plus some additional features. In the case of an error decrease, the learning rate is increased proportionally to a constant ϕ which is greater than 1. An error increase is considered only when the current error exceeds the previous one by 1%–5%, in which case the weight update is rejected, the momentum coefficient (which is part of the technique) is set to zero, the learning rate is multiplied by a constant β smaller than 1, and the step is repeated. The act of setting the momentum to zero when an error increase is detected is justified by the fact that otherwise its memory effect would possibly cause the search procedure to still go up despite the rejection of the weight update. Batch learning and a global learning rate are used. There is no specific reference to the values that ϕ and β should assume, except the ones shown to be used in one of the tests performed to measure the performance of the method, as compared to standard BP. No reference is given to the initial values of learning rate and momentum, nor about the sensitiveness of the method to those.

[Hush-88] Although the essence of this method is not centered on learning rate adaptation, it is still performed. Instead of calculating the gradient values at each iteration, the same values are used in several iterations, being recalculated only when they produce no reduction in the error function anymore. This is expected to reduce the number of gradient calculations that will become smaller than the number of iterations, thus saving computational effort and time. The learning rate is then adapted based on the number of times that the gradient is consecutively reused. If the reuse rate is high, the learning rate is increased, being assumed that the search is stable. If the reuse rate is low the learning rate is decreased. Hush and Salas concentrate mainly in the gradient reuse part, the learning rate adaptation procedure being poorly defined. The high and low values of the reuse rate must be stated in advance, so that when any of them is reached, the corresponding learning rate adaptation is performed. However, these values are not clearly defined, nor how the learning rate should be adapted in each case. The learning rate parameter is global, there is no reference to momentum, and the weights are updated in batch mode.

Prediction of new values for the learning rate. Some adaptation techniques increase or decrease the learning rate at each iteration based on information obtained from the behaviour of the search procedure and the current state of the network. A different approach consists of anticipating the adaptation by suggesting different new values for the learning rate, measuring the error on the new point found by the weight update produced by each, and choosing the one that gives the best results.

[Salomon-90] [Salomon-92] The technique proposed by Salomon, later with the cooperation of van Hemmen, consists in finding two new values for the learning rate. Each of these two values is found by multiplying the current value by 1.3 and by 0.77 ($\approx 1/1.3$). A momentum term is used and the adaptation of the learning rate is alternated with the adaptation of momentum in consecutive iterations. The technique used is exactly the same for both parameters. An upper limit is defined for each to avoid numerical problems. Global parameters and batch updating are used. Furthermore, the authors propose the use of a gradient normalization procedure. At the end of each epoch, the gradient vector that consists of the summation of the individual gradients from the presentation of all patterns is divided by its own norm before being used to update the weights. The upper limits for learning rate and momentum are defined by the authors, leaving no tunable parameters.

Searching for zero-points of the error function instead of zero-points of its derivative. The principle of this technique is to find the tangential hyper-plane in the current point of the error surface at each iteration and consider it as a local approximation to the error surface itself. Next, the step size is chosen in a way that a step is taken towards the point where that hyper-plane tangent intersects the error function's zero-plane, being expected that it corresponds to a point relatively close to the zero of the actual error surface. The same procedure is repeated with the point found at each iteration being used as starting point for the next one until a point sufficiently close to zero is found.

[Schmidhuber-89] The geometric principles supporting the technique proposed by Schmidhuber will then set the learning rate at each iteration as

$$\eta = \frac{E_p}{||\nabla E_p||^2}$$

in case the gradient is non-zero. E_p is the error caused by the presentation of pattern p in the current epoch. If the gradient is zero, the learning rate is also set to zero. Considering the nature of the technique, there are situations where an undesirably too big step may be taken. To avoid this, an upper bound is set to the learning rate, defined by the author as having the value of 20. Another feature of the technique is the addition of a very small negative constant to the error function, in order to guarantee that the minimum is achieved in problems where the error surface does not have a zero point. There is no suggested value for this constant nor any clear principle to decide whether to use it or not, depending on the particular problem and network used. The learning rate is global, on-line training is used, and momentum is not considered.

Using peak values for the learning rate The use of small learning rate values is needed to allow true convergence but they lead to slow convergence rates. On the other hand, using high values often cause oscillations and some times even unpredictable behaviour of the search. This technique is aimed at trying to use peak values for the learning rate without causing oscillations by selecting the appropriate occasions where those values should be applied.

[Cater-87] The method proposed by Cater works with on-line training and involves two techniques:

1. Consider, at epoch k, the pattern p that originates the highest aberration $\max \mid t_{j_k} - a_{j_k} \mid$ amongst all the pattern set and all the output units j of the network; during epoch $k+1$, the learning rate used for the presentation of that same pattern will be:

$$\eta_{p_{k+1}} = 2\,\eta_{k+1} + \mid t_{j_k} - a_{j_k} \mid,$$

   η_k being the global learning rate at epoch k. This is used to compensate for the cases where the target error is high and yet the delta rule corrections for the corresponding pattern are small.

2. To avoid training oscillations and local minima, each time that the error increases by at least 1% in comparison to the previous iteration, the global learning rate is decreased by 50%.

The method uses a fixed momentum term. Both the initial learning rate and momentum values must be selected before training. The author proposes $\eta_0 = 10$ and $\mu \leq 0.001$. The proposed initial learning rate value seems too high, but it is stated that, with the second technique described above, when the search reaches a state of some oscillation, the increases of the error lead to a decrease in the value of η. In particular, it was observed that, after several thousand epochs, that value suffered a decay of some orders of magnitude, going below 10^{-6}.

2.2.4 Other methods

Calculation of the optimal fixed values for the parameters before the training Here, a somewhat different technique to select the learning rate is presented, which consists in using an heuristic to find a good fixed value for it before the training. Although this avoids the trial-and-error selection for the initial optimal value for η, it does not allow it to change and adapt to the different regions of the error surface. This suggests that a sufficiently reliable such technique could possibly be combined with adaptive learning rate techniques that still demand the user choice of the parameter's initial value.

[Eaton-92] Eaton and Olivier state that the range of η's that will produce rapid training depends on the number and types of input patterns. In addition, they observed that the length of the error gradient increases with the size of the training set. These conclusions were taken after a study of the behaviour of the batch BP algorithm using three-layered networks and lead them to propose a formula that calculates the fixed learning rate value based on the characteristics of the training set, assuming that we can divide it into subsets with similar patterns:

$$\eta = \frac{1.5}{\sqrt{N_1^2 + N_2^2 + \cdots + N_m^2}}.$$

N_m is the number of patterns of the subset m, with a total of m subsets. The value of 1.5 in the numerator was chosen empirically and a momentum term of 0.9 should be used in parallel. Although this is a straightforward procedure in some cases, situations exist where the partitioning of the set becomes difficult. When the similarity of patterns and the existence of distinct types are difficult to detect, a technique may be applied, which consists in observing the outputs of the network to decide on the selection of the different types, although the value of η obtained will then be suboptimal. Details are not given about this technique. The method may not be considered sufficiently general, since nothing is said about the procedure that should be followed with a real valued function mapping problem, for example. A global learning rate value is used.

3 Chosen Methods

A considerable number of methods for learning rate and/or momentum adaptation have been presented. From those, some of them were selected to be implemented and tested, based on a well defined criterion. As an attempt to create a general definition of a *good training method*, some features that should be verified are here enumerated. Namely, from [Chan-87]:

Robustness - the ability to adapt to different kinds of problems.

Efficacy - the presentation of good performance results.

Insensitiveness - independence of user set initial parameter values.

Computational efficiency - avoidance of large computational and storage demands.

To these four, the following can be added:

Generalization ability - the ability to obtain a set of network weights that allows it to perform the desired mapping, being capable of correctly classifying examples that were not used during the training.

3.1 The need for parameterless methods

Due to the diversity of possible applications, it is still difficult to create a method that is able to perform well in all situations. One possible solution is to introduce appropriate parameters in the method that can be tuned according to the particular task to be performed. While this is a resource adopted by some methods, its usage creates a clear contradiction, since one of the advantages of an adaptation method is to avoid initial parameter tuning. This paper is therefore focused on finding a method that is simple to operate by the user - meaning that it involves no initial parameter specification and is well defined. The methods selection emphasis is thus on the *insensitiveness* criteria; all the other points being then considered in their final evaluation. Neither of the tested methods presents tunable parameters or, in the case that it does, their suitable values are clearly specified as being fixed and sufficiently general to work for all applications.

3.2 The selection of parameterless methods

A total of five methods were implemented, tested, and compared: SCG (Møller), Chan-Fallside, Silva-Almeida, RPROP (Riedmiller), and Salomon-van Hemmen. They were all implemented as suggested without any modifications, with the exception of the method from Chan-Fallside. Implementation details including additional information not provided in the descriptions above are:

SCG It was implemented exactly as described.

Chan-Fallside Some modifications were made to this method. Upper and lower limits were imposed for learning rate (between 10^{-8} and 10) and momentum (between 10^{-8} and 0.95). The backtracking procedure had to be removed because of the chaotic results it produced – the training virtualy always entered a state of no evolution at the point where the backtracking occurred, the learning rate always going down. With these two modifications it performed well. Gradient normalization was used. Furthermore, the subdivision of the training set into subsets was not tried.

Silva-Almeida The authors recommend the use of higher and lower values for the learning rate, respectively a few orders of magnitude below and above the largest and the smallest representable number of the particular machine used. The learning rate values were allowed to vary between 10^{-8} and 100. As suggested for the backtracking procedure, the error is allowed to increase in three consecutive iterations, after which the learning rate values are all reduced by 50%. The weights are still rejected in every error increase. The only additional implementation change consisted in normalizing the gradient before updating the weights, as suggested in the method of Salomon-van Hemmen.

RPROP No modifications were made in this method. The upper limit for the update value is 50 and the lower is 10^{-6}.

Salomon-van Hemmen No modifications were made. The upper limit for the learning rate is 2.0 and for momentum is 0.95.

These five methods were compared using both versions of the BP algorithm: on-line and batch.

4 Simulations

This section contains all the information about the simulations performed to test the chosen methods, including the description of the benchmark problems used to test them, the relevant details relating to their implementation, and the presentation of the obtained results, followed by their discussion.

4.1 Benchmarks used

For testing the chosen methods, a set of benchmarks including six real-world problems plus the Exclusive-OR (XOR) problem was used. The numbers inside the brackets indicate the input and output layer sizes.

Auto-MPG (7,1) This concerns city-style fuel consumption of cars in miles per gallon, to be predicted based in 3 multi-valued discrete and 4 continuous attributes relating to their technical properties. All input and output values are scaled to the interval [0,1]. Incomplete patterns have been removed.

Digit (64,10) The data consists of handwritten digits of the *NIST Special Database 3*. Each pixel is represented by an eight bit value. Each digit was scaled to fit into an image of 8×8 points. The patterns are equally distributed over the ten digits. The input values are scaled to the interval [0,1] and the boolean output values are encoded as 0 and 1.

Exclusive-OR (2,1) This is the well known non-linear problem that has been widely used in the literature to evaluate the performance of the training. The inputs and the output are encoded as 0 and 1.

Promoters (228,1) This problem relates to E. Coli promoter gene sequences (DNA) with associated imperfect domain theory. Given a set of 57 sequential DNA nucleotide positions of four types, the task is to predict if it is member/non-member of a class of sequences with biological promoter activity. The inputs are coded as a 1-of-4 binary representation, resulting in 228 binary inputs. The binary inputs and output are encoded as 0 and 1. This problem is linearly separable, which justifies the use of no hidden layer.

Sonar (60,2) The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Each pattern is a set of 60 numbers in the range [0,1]. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The boolean output values are encoded as 0 and 1

Vowels (20,5) This is a subset of 300 patterns of the vowel data set, obtainable via ftp from `cochlea.hut.fu (130.233.168.48)` with the LVQ-package (lvq_pak). An input pattern consists of 20 unscaled cepstral coefficients obtained from continuous Finnish speech. The task is to determine whether the pronounced phoneme is a vowel and, in the case it is, which of the five possible ones. The input values are scaled to the interval [0,1] and the boolean output values are encoded as 0 and 1.

Wine (13,3) Is the result of a chemical analysis of wines grown in a region of Italy derived from three different cultivars. The analysis determined the quantity of each of 13 constituents found in each of the three types of wines. A wine has to be classified using these values, which are scaled to the interval [0,1]. The output patterns use boolean values, encoded as 0 and 1.

4.2 Implementation details

Networks with zero or one hidden layer only are used, zero for Promoters and one for all others (see table 1). They are fully interlayer connected, meaning that each neuron of a layer is connected with all the neurons of the adjacent layers, including biases.

The activation function used in all cases was the sigmoid presented in section 1.1, with exception of the Auto-MPG benchmark. In this case, a linear activation function ($y=x$) was used in the output layer of the network. This is justified by the fact that the output values are continuous in this case.

In the simulations performed here it was decided to repeat each set of experiments with a set of three different initial learning rate values with a maximum distance between them of one order of magnitude. The values are 0.5, 0.1, and 0.05. On one hand, this set allows the evaluation of differences in the behaviour of the methods with somewhat different initial conditions (the method's robustness). One the other hand, considering these as middle-range values, there is no interest in experimenting with others, since these are the ones that will probably be used with methods that do not require parameter tuning. Following the same principles, two different initial momentum values are used: 0.9 and 0.5. The only exception to this is the method of Silva-Almeida. As the authors advise the use of small initial learning rate values, they are here an order of magnitude below those used for the other methods, that is 0.05, 0.01, and 0.005. The two versions of the standard BP algorithm, on-line and batch, were tried with the same initial values as the other methods. For RPROP and SCG, as they are both exempt of initial LR and momentum values, no more than one set of runs is necessary for each. For the Chan-Fallside method, the fact that the λ_0 parameter is limited between 0 and 1 causes it to be exactly the same for both momentum values used in these experiments (0.5 and 0.9), the results being the same in both cases.

For each combination of initial values for LR and momentum 10 simulations were performed with different initial weights, except for the Exclusive-OR benchmark, where 20 were performed. The weights were always initialized with random numbers between -0.40 and 0.40.

It was observed that the training methods in general had difficulties in learning a network with 2 hidden units for the Exclusive-OR problem. This benchmark was then run in two different versions: one with a 2-4-1 topology and the other with a (more difficult) 2-2-1 one. Results for both are presented in the corresponding section.

The criterion for convergence used was based on the *maximal aberration*, which gives the maximal difference between the target and the actual output for all patterns in the traing set and all network outputs. Convergence was accepted when that value was below a threshold ϵ. A different convergence criterion was used for the benchmark problems using generalization, as described in section 4.3.

The maximum number of epochs allowed varied for each benchmark. That number was always considerably bigger than the largest number of epochs spent by any of the methods to successfully reach convergence. This means that the reported non-convergences always refer to cases where the training clearly stagnated without satisfying the convergence criterion.

The squared error percentage, as mentioned in section 1.1, was used in all cases. With this, the error values presented in the results are normalized and hence independent of the benchmark.

4.3 Generalization

Following the generalization ability criterion mentioned in section 3, a procedure was used in these simulations to check the quality of the network. This was applied to three benchmarks only, whose number of available patterns per number of output classes allowed a proper evaluation of the generalization properties: Auto-MPG, Digit, and Wine.

To verify the generalization, the complete available pattern set was sub-divided, respecting an equal distribution of the different classes, in three sets: a *training* set with 50% of the total patterns and a *validation* set and a *test* set, each of these with 25% of the patterns.

The network was trained using only the training set and after every five epochs the validation set was presented to check the generalization error. In this case, the convergence criterion was different and based on the validation set error. Consider the *training progress* at epoch k as

$$Prog(k) = 1000 \cdot \left(\frac{\sum_{t=k-4}^{k} E(t)}{5 \ . \ \min_{t \ \in \ k-4,\ldots,k} \ E(t)} - 1 \right) ,$$

where $E(k)$ is the training error at epoch k. $Prog(k)$ gives a measure of the evolution of the training over the last five epochs. Furthemore, consider the *generalization loss* at epoch k as

$$GL(k) = 100 \cdot \left(\frac{E_{va}(k)}{\min_{t<k} \ E_{va}(t)} - 1 \right) ,$$

$E_{va}(k)$ being the validation error at epoch k. $GL(k)$ corresponds to the percentage of validation error growth comparing to its best value so far. These two parameters were measured every five epochs, after presenting the validation set. Then, training was stoped when one of two different situations occured: $Prog(k)$ fell below 0.1 or $GL(k)$ went beyond a threshold of 5.0 in five consecutive measurements. This means that training was considered concluded either when the training error stagnated or when a reduction on that error lead to a deterioration in the generalization properties of the network. After the training stoped, the test set was presented using the network configuration that provided the lowest validation error. The results presented for the benchmarks using generalization are all based on the values measured with that configuration.

The presented simulation results relating to the percentage of misclassified patterns in each of the pattern sets were obtained with two different procedures: for the Auto-MPG benchmark, a misclassified pattern corresponds to one whose maximal aberration does not exceed ϵ, and for Digit and Wine, a pattern is considered correctly classified whenever the network output corresponding to the correct one is higher in value than all the other outputs. This procedure is called *winner-takes-all* and can only be used in problems whose output is implemented as a 1-of-N_L representation.

4.4 Benchmark problems overview summary

In table 1, the input and output representation obeys the following notation: "b" stands for *boolean*, "c" stands for *continuous*, and "mvd" stands for *multi-valued discrete*. The "#classes" column represents the number of different output types or classes of patterns. "w.t.a" in the column of the ϵ values means that the winner-takes-all method was used to evaluate pattern misclassification, no ϵ being used.

benchmark	network topology	gener. used	pattern set sizes train.	valid.	test	representation inputs	outputs	#classes	ϵ
Auto-MPG	7-3-1	yes	196	98	98	4mvd+3c	1c	–	0.1
Digit	64-30-10	yes	500	250	250	64mvd	10b	10	w.t.a.
Exclusive-OR	2-4-1	no	4	–	–	2b	1b	2	0.1
Exclusive-OR(2)	2-2-1	no	4	–	–	2b	1b	2	0.1
Promoters	228-1	no	106	–	–	228b	1b	2	0.1
Sonar	60-8-2	no	104	–	–	60c	2b	2	0.1
Vowels	20-20-5	no	300	–	–	20c	5b	25	0.1
Wine	13-6-3	yes	89	44	45	13c	3b	3	w.t.a.

Table 1: Summary of the benchmark problems characteristics.

4.5 Results

The tables with the results from all the experiments done are presented in appendix A, starting with the ones that do not use generalization. Each row in the tables corresponds to a set of simulations with different initial weights. For the benchmarks not using generalization, the "%Conv" column indicates the percentage of simulations in the set where convergence was not reached. The numbers in italics are aimed at emphasizing results for sets of simulations with less than 100% convergence, since the statistics for the number of epochs is based on the successful simulations only.

4.6 The number of epochs

In this subsection, the performance results concerning the number of epochs and the convergence abilities of each of the methods is discussed, based on the tables presented in appendix A.

Starting with some general comments, it can be stated that the results for different methods vary depending on the benchmark problem. In particular, one very important remark must be made: The Exclusive-OR benchmark, thoroughly used in the literature to compare different methods, should not be taken as a serious and definitive instrument of comparison. In fact, the results presented for this benchmark differ considerably from those of the other benchmarks, with the bigger discrepancies between the fixed and the adaptive methods, specially between on-line BP and all the other methods.

Referring to some of the different benchmarks, unsatisfying results were obtained for Auto-MPG, since neither of the tested methods was able to reach true convergence. Nevertheless, the attained generalization performance was nearly the same for all the methods. A similar generalization performance for all methods was also obtained with Wine. This makes the comparisons between the methods easy for these two problems, the only difference between them being the number of epochs spent to reach those values. The analysis of the results for Digit, however, are more difficult, since different methods achieved different generalization performances. The fact that the Promoters problem is linearly separable leads to very good results for all the methods. Hence, this benchmark problem does not allow useful comparisons between different methods; the only remark concerning the fact that on-line BP had convergence problems for initial values of 0.5 and 0.9 for learning rate and momentum, respectively.

About each of the tested methods, the first comment relates to the performance of both versions of standard BP, which is visibly dependent on the values of learning rate and momentum. In particular, and as expected, the number of epochs spent increases with decreasing initial values of both parameters. Batch BP is in almost all cases faster than on-line BP, the latter having failed to converge more times. In some benchmark problems, on-line BP shows serious convergence problems when values of 0.5 and 0.9 for learning rate and momentum, respectively, are used.

The SCG method showed very good results for XOR, was the best by far on Sonar, one of the best on Wine, but it was the worst adaptive method on Auto-MPG. Furthermore, it was never capable of reaching convergence for Vowels. The percentage of convergence for XOR(2) was 85%, which is good.

Considering the number of epochs, the Chan-Fallside method was almost always worse than batch BP and the worst among the adaptive methods. The percentage of convergence for Vowels never went beyond 40%, was 60%–80% for Sonar, and 90%–95% for XOR(2), this latter being a good result.

The method of Silva-Almeida was the best on Auto-MPG, among the best on Wine and both XOR problems, was able to obtain good results for Sonar and very good for Vowels, in terms of number of iterations, but showed serious convergence problems. It was the only method that failed once for XOR, and the one that failed more often on XOR(2). It converged only a few times on Sonar and had a considerable number of non-convergences on Vowels.

RPROP showed average performance on both XOR and Sonar. The results for Vowels and Auto-MPG were good, and it was among the best on Wine. As far as convergence is concerned, the results were 55% on XOR(2), 40% on Sonar and 70% on Vowels, which may not be considered satisfactory.

Finally, the Salomon-van Hemmen method gave good results for XOR and Auto-MPG, the best on Wine, but without good results for Sonar and Vowels. This method was, however, the one that showed the best percentage of convergence overall.

The Digit benchmark problem was the one with the largest size. Its analysis is done independently from the other problems because the results obtained present some particularities. Both on-line and batch BP failed to converge with a learning rate of 0.5 and momentum of 0.9, while the SCG method was never able to converge. Chan-Fallside also had serious non-convergence problems. RPROP was the method with best performance, but without very good generalization results. Silva-Almeida had good performance on the number of epochs, but some convergence problems. The generalization is also not optimal. The number of epochs spent by Salomon-van Hemmen is higher than the best methods, but the generalization results are good.

Finally, the insensitiveness of the behaviour of the methods to initial parameter values is discussed. The results show that the three adaptive methods that demand the choice of initial learning rate and momentum values (Chan-Fallside, Silva-Almeida, and Salomon-van Hemmen) can be considered insensitive to those, being able to show similar performances with different initial values. A few exceptions can be mentioned, including the method of Silva-Almeida with Auto-MPG, showing some differences on the performance for the two different initial momentum values, and the Salomon-van Hemmen method on Exclusive-OR, its performance decaying with increasing values of initial learning rate and momentum.

4.7 The complexity

There is no established standard for comparisons between the performance of different training methods. Various ways of showing the performance of a method have been used. While the number of epochs is a safe criterion, it is insufficient, since it hides differences in the processing demands per epoch of each method. If this would be available, along with the number of epochs spent, a considerably accurate estimation of the actual processing effort of the training could be obtained. The approach chosen here consists in counting the number of arithmetical operations per epoch spent by each method on each benchmark.

The equations of table 2 are based on certain assumptions, meaning that some of them should be modified when different conditions apply. Namely, it is considered that a sigmoid activation function is used, along with fully interlayer connected networks. The latter condition is very important and determines the equations of operations 1, 2, and 3. The number of arithmetical operations (complexity) is obtained considering the three different operations: addition, multiplication, and exponentiation each counting as one. Although it is true that their complexities are different and correspond to different amounts of basic processor operations, to take this into account would be a difficult task and would further increase the complexity of this presentation, but, if required, it can be done based on the results presented here. Backtracking procedures and the verification of the upper and lower bounds of the parameters were not considered in the calculation of the complexity.

In table 5, the number inside brackets for the SCG method mean that the corresponding operations are conditional, as determined by the method's algorithm and will not, presumably, be performed every epoch. It is considered, however, that the number of epochs in which they are not performed is small, as compared to the total number. Thus, the complexity obtained for this method considers the case where a normal (complete) epoch takes place.

The final complexity results are presented in table 6. The complexity of the on-line version of the standard BP with momentum is 30%–70% bigger than that of the batch version, depending on the benchmark problem. This is due to the weight update with momentum that is done at each epoch

#	Operation description	Equations	Complexity per epoch
Op1	propagate all patterns	$a_j = \frac{1}{1+e^{-\text{net}_j}}$, $\text{net}_j = \sum_{i=0}^{N_{l-1}} (w_{ij} \times a_i)$ $\quad j \in l_{2 \leq l \leq L}$	$(4 \times (N - N_1)) \times P$ $(W + W - (N - N_1)) \times P$
Op2	backpropagate and update of the weights (with momentum)	$w_{ij}(p-1) - \eta \times \delta_j \times a_i + \alpha \times \Delta w_{ij}(p-1) \quad \forall w \in W$ $\delta_j = \begin{cases} a_j \times (1-a_j) \times (t_j - a_j) & j \in L \\ a_j \times (1-a_j) \times \sum_{k=1}^{N_l+1} (\delta_k \times w_{jk}) & j \in l_{1<l<L} \end{cases}$	$(5 \times W) \times P$ $(4 \times N_L) \times P$ $(3 \times (N - N_1 - N_L)) \times P$ $((W - (N_1 \times N_2) - (N - N_1)) + ((W - (N_1 \times N_2) - (N - N_1)) - (N - N_1 - N_L))) \times P$
Op3	backpropagate without updating the weights	$\Delta w_{ij}(n) = \sum_P (\eta \times \delta_j \times a_i) \quad \forall w \in W$ $\delta_j = \begin{cases} a_j \times (1-a_j) \times (t_j - a_j) & j \in L \\ a_j \times (1-a_j) \times \sum_{k=1}^{N_l+1} (\delta_k \times w_{jk}) & j \in l_{1<l<L} \end{cases}$	$(2 \times W) \times (P-1)$ $(4 \times N_L) \times P$ $(3 \times (N - N_1 - N_L)) \times P$ $((W - (N_1 \times N_2) - (N - N_1)) + ((W - (N_1 \times N_2) - (N - N_1)) - (N - N_1 - N_L))) \times P$
Op4	end of epoch weight update with momentum	$w_{ij}(n-1) + \Delta w_{ij} + \alpha \times \Delta w_{ij}(n-1) \quad \forall w \in W$	$3 \times W$
Op5	end of epoch weight update - RPROP	$w_{ij}(n-1) \times \Delta_{ij}(n) \quad \forall w \in W$	W
Op6	end of epoch weight update - SCG	$w_{ij}(n) - \eta \times grad_{ij}(n) \quad \forall w \in W$	$2 \times W$
Op7	inner product of two vectors x and y	$\sum_{w=1}^{W} (x_w \times y_w) \quad \forall w \in W$	$W + (W-1)$
Op8	norm of vector x	$\sum_{w=1}^{W} (x_w \times x_w) \quad \forall w \in W$	$W + (W-1)$
Op9	sum (or subtraction) of two vectors x and y	$(x_w + y_w)_{w \in W} \quad \forall w \in W$	W
Op10	independent parameter update (η_{ij} or Δ_{ij}) based on previous and current local gradient sign	$grad_{ij}(n-1) \times grad_{ij}(n)$, $\eta_{ij} \times \text{Const} \quad \forall w \in W$	W W

Table 2: Operation equations and complexity per epoch

Method	Operation									
	Op1	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10
On-Line BP	1	1	–	–	–	–	–	–	–	–
Batch BP	1	–	1	1	–	–	–	1	–	–
SCG	3 (-1)	–	2 (-2)	2 (-1)	–	2 (-1)	3 (-2)	3 (-2)	2 (-2)	–
Chan-Fallside	1	–	1	1	–	–	1	3	–	–
Silva-Almeida	1	–	1	1	–	–	–	1	–	1
RPROP	1	–	1	–	1	–	–	–	–	1
Salomon-van Hemmen	3	–	1	3	–	–	–	1	–	–

Table 5: Operations used by each method per epoch

Benchmark	W	N	N_1	N_2	N_L	P
Auto-MPG	28	11	7	3	1	196
Digit	2260	104	64	30	10	500
Exclusive-OR	17	7	2	4	1	4
Exclusive-OR(2)	9	5	2	2	1	4
Promoters	229	229	228	1	1 (=N_2)	106
Sonar	506	70	60	8	2	104
Vowels	525	45	20	20	5	300
Wine	105	22	13	6	3	89

Table 3: Characteristics of each benchmark

Benchmark	Op1	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10
Auto-MPG	13,328	30,576	14,056	84	28	56	55	55	28	56
Digit	2,320,000	6,000,000	2,605,480	6,780	2,260	4,520	4,519	4,519	2,260	4,520
Exclusive-OR	196	420	182	51	17	34	33	33	17	34
Exclusive-OR(2)	108	228	102	27	9	18	17	17	9	18
Promoters	48,866	121,794	48,514	687	229	458	457	457	229	458
Sonar	108,368	268,944	110,060	1,518	506	1,012	1,011	1,011	506	1,012
Vowels	337,500	865,500	391,950	1,575	525	1,050	1,049	1,049	525	1,050
Wine	21,093	52,065	23,820	315	105	210	209	209	105	210

Table 4: Complexity of each operation when used with each benchmark per epoch

presentation with the on-line version. The SCG method is by far the most costly amongst all, including both versions of standard BP. This confirms the status of the conjugate gradient methods as being computationally too demanding. The Chan-Fallside, Silva-Almeida, and RPROP methods have all nearly the same complexity, comparable to that of batch BP. The method of Salomon-van Hemmen is also considerably complex, exceeding all the methods except SCG. However, the algorithm used by this method strongly encourages the use of a parallel implementation that would allow it to perform two simultaneous epochs. Its complexity would then be reduced to that of batch BP.

5 Conclusions

The objective of this work was to evaluate the present situation concerning the incorporation of learning rate and momentum adaptation on the backpropagation algorithm. A large number of methods were reviewed, from which a set of five was chosen to be implemented and tested, together with both the on-line and batch versions of the standard backpropagation algorithm. The criterion for the choice of the methods to be implemented took mainly into account the fact that they were well-defined and had no tunable parameters. Simulations were performed using a set of six real-world benchmark problems plus the boolean Exclusive-OR problem.

The first conclusion that can be drawn from the results is that there is no clear best method among those that perform automatic parameter adaptation. Nevertheless, comparing the fixed parameter methods with the adaptive ones, it can be seen that considerable improvements have been made. Although each of the adaptive methods behaved different for different problems, some general conclusions can be drawn for each of them.

The Scaled Conjugate Gradient method from Møller was able to achieve very good performances on some benchmarks, but clearly failed on the bigger ones, showing a poor scaling ability. This is accentuated by the fact that its computational complexity is too high, all resulting in a general inability to handle large scale problems.

The Chan-Fallside method was the most unsatisfying of all. It offers almost no advantages over the fixed parameter methods, in terms of number of iterations, although the raise in the complexity is insignificant. Additionally to being clearly the least performing of all the adaptive methods, it was also not able to converge in several situations.

The method of Silva-Almeida gave very good results for some benchmarks, concerning the number of

Method	Auto-MPG	Digit	X-or	X-or(2)	Promoters	Sonar	Vowels	Wine
On-Line BP	43×10^3	$8{,}320 \times 10^3$	616	336	170×10^3	377×10^3	$1{,}203 \times 10^3$	73×10^3
Batch BP	27×10^3	$4{,}936 \times 10^3$	462	254	98×10^3	220×10^3	732×10^3	45×10^3
SCG	68×10^3	$12{,}225 \times 10^3$	1,354	738	249×10^3	557×10^3	$1{,}808 \times 10^3$	113×10^3
Chan-Fallside	27×10^3	$4{,}950 \times 10^3$	561	305	99×10^3	223×10^3	735×10^3	46×10^3
Silva-Almeida	27×10^3	$4{,}941 \times 10^3$	496	272	98×10^3	221×10^3	733×10^3	45×10^3
RPROP	27×10^3	$4{,}932 \times 10^3$	429	237	98×10^3	219×10^3	731×10^3	45×10^3
Salomon-van Hemmen	54×10^3	$9{,}590 \times 10^3$	956	524	197×10^3	440×10^3	$1{,}410 \times 10^3$	88×10^3

Table 6: Computational complexity of each method when used with each benchmark per epoch

iterations, but, most important, it has serious convergence problems, being unable to successfully train the networks for several different problems. The complexity increase is low in comparison to the fixed parameter methods.

The method of Riedmiller and Braun (RPROP) showed average performance, in general. Of all the methods, it is the one with the lowest computational complexity, although not very different from the fixed parameter ones. In terms of number of iterations, it showed moderate results, although with some convergence problems.

Finally, for the method of Salomon-van Hemmen, the performance varied per benchmark, being very good for some, but disappointing for others. The two main features of this method are its computational complexity and convergence abilities. It is clearly more complex than all the others, except SCG, but it was able to achieve convergence in almost all situations. In fact, despite of its complexity, this can be considered as the most robust method of all. As already discussed in section 4.7, it can easily be implemented in a two-processor architecture, which would allow it to become a good all-purpose method with satisfying results.

Although a side issue of this paper, it can be concluded that, based on the experiments described here, the batch version of the standard BP is better than the on-line version, concerning the number of iterations spent to converge, the percentage of convergence, and the computational complexity. This is not intended at discussing the advantages and disadvantages of batch and on-line training. In fact, it cannot be concluded from here that the results obtained for both versions of the standard BP algorithm generalize to adaptive algorithms. One thing that can be taken as certain is the difference in the complexity of both versions, when momentum is used, batch training being computationally less demanding.

In general terms, the performance of the tested adaptive methods is better than that of the standard BP algorithm. However, the reduction is not remarkably high, specially considering the real-world benchmark problems used. It was even noted that in some cases the non-adaptive methods yielded better results for some particular values of the learning rate and momentum parameters. Nevertheless, an improvement in the performance is verified and, an important result, they avoid the usual parameter tuning by trial-and-error.

References

[Almeida-90] Fernando M. Silva and Luis B. Almeida. "Acceleration Techniques for the Backpropagation Algorithm." In Luis B. Almeida and C. J. Wellekens (editors), *Neural Networks; Proceedings of the EURASIP Workshop 1990* (Sesimbra, Portugal; February 15–17, 1990) (ISBN: 3-540-52255-7), volume 412 of *Lecture Notes in Computer Science*, pages 110–119. Springer-Verlag, Berlin, Germany, 1990. Series editors: G. Goos and J. Hartmanis.

[Battiti-89] Roberto Battiti. "Accelerated Backpropagation Learning: Two Optimization Methods." *Complex Systems* (ISSN: 0891-2513), volume 3, pages 331–342. Complex Systems Publications, Inc., Champaign, IL, 1989.

[Battiti-92] Roberto Battiti. "First and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method." *Neural Computation*, volume 4, number 2, pages 141–166. MIT Press, Cambridge, MA, 1992.

[Becker-89] Sue Becker and Yann le Cun. "Improving the Convergence of Back-Propagation Learning with Second Order Methods." In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski (editors), *Proceedings of the 1988 Connectionist Models Summer School* (Carnegie Mellon University; June 17–26, 1988) (ISBN: 0-55860-015-9; LCCCN: 88-031992), pages 29–37. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Cater-87] J. P. Cater. "Successfully Using Peak Learning Rates of 10 (and Greater) in Back-Propagation Networks with the Heuristic Learning Algorithm." In Maureen Caudill and Charles Butler (editors), *Proceedings of the IEEE First International Conference on Neural Networks (ICNN)* (IEEE; San Diego, California; June 21–24, 1987) (IEEE Catalog Number: 87TH0191-7), volume II, pages 645–651. SOS Printing, San Diego, California, 1987.

[Chan-87] Laiwan W. Chan and Frank Fallside. "An Adaptive Training Algorithm for Backpropagation Networks." *Computer Speech and Language*, volume 2, pages 205–218. Academic Press Limited, Cambridge, Masachussetts, September/December 1987.

[Darken-92] Christian Darken, Joseph Chang, and John Moody. "Learning Rate Schedules for Faster Stochastic Gradient Search." In S. Y. Kung, F. Fallside, J. Å. Sørenson, and C. A. Kamm (editors), *Neural Networks for Signal Processing II; Proceedings of the 1992 IEEE-SP Workshop* (IEEE-SP; Helsingøer, Denmark; August 31–September 2, 1992) (ISBN: 0-7803-0557-4; LCCCN: 91-59047), pages 1–12. Piscataway, New York, U.S.A., 1992.

[Eaton-92] Harry A. C. Eaton and Tracy L. Olivier. "Learning Coefficient Dependence on Training Set Size." *Neural Networks* (ISSN: 0893-6080), volume 5, number 2, pages 283–288. Pergamon Press, 1992.

[Fahlman-89] Scott E. Fahlman. "Faster Learning Variations on Back-Propagation: An Empirical Study." In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski (editors), *Proceedings of the 1988 Connectionist Models Summer School* (Carnegie Mellon University; June 17–26, 1988) (ISBN: 0-55860-015-9; LCCCN: 88-031992), pages 38–51. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Haffner-89] Patrick Haffner. "Fast Back-Propagation Learning Methods for Neural Networks in Speech." Rapport de stage; ingenieur elève, deuxieme année; promotion 1989, Telecom Paris; Ecole Superieure des Telecommunications, Paris, 1989.

[Haykin-94] Simon Haykin. "*Neural Networks; A Comprehensive Foundation*" (ISBN: 0-02-352761-7; LCCCN: QA76.87.H39). Macmillan College Publishing Company / IEEE Press, New York, New York, 1994.

[Hsin-95] Hsi-Chin Hsin, Ching-Chung Li, Mingui Sun, and Robert J. Sclabassi. "An Adaptive Training Algorithm for Back-Propagation Neural Networks." *IEEE Transactions on Systems, Man and Cybernetics*, volume 25, number 3, pages 512–514. IEEE, New York, March 1995.

[Hush-88] D. R. Hush and J. M. Salas. "Improving the learning rate of backpropagation with the gradient reuse algorithm." In *Proceedings of the IEEE International Conference on Neural Networks* (IEEE; San Diego, CA), volume I, pages 441–447. 1988.

[Jacobs-88] Robert A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaptation." *Neural Networks* (ISSN: 0893-6080), volume 1, number 4, pages 295–307. Pergamon Press, New York, 1988.

[Johansson-92] E.M. Johansson, F.U. Dowla, and D.M. Goodman. "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method." *International Journal of Neural Systems* (ISSN: 0129-0657), volume 2, number 4, pages 291–301. World Scientific Publishing Company, 1992.

[Leen-94] Todd K. Leen and Genevieve B. Orr. "Optimal Stochastic Search and Adaptive Momentum." In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector (editors), *Advances in Neural Information Processing Systems (NIPS) - Natural and Synthetic 6* (1993). Morgan Kaufmann Publishers, San Mateo, California, 1994.

[Leonard-90] J. Leonard and M. A. Kramer. "Improvement of the Backpropagation Algorithm for Training Neural Networks." *Computers & Chemical Engineering*, volume 14, number 3, pages 337–341. Pergamon Press plc, 1990.

[Minai-90] Ali A. Minai and Ronald D. Williams. "Acceleration of Back-Propagation through Learning Rate and Momentum Adaptation." In Maureen Caudill (editor), *Proceedings of the International Joint Conference on Neural Networks (IJCNN-90-WASH-DC)* (INNS and IEEE; Omni Shoreham Hotel, Washington, DC; January 15–19, 1990) (ISBN: 0-8058-0754-3), volume I: Theory Track, Neural and Cognitive Sciences Track, pages 676–679. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1990.

[Moller-93] Martin Møller. "*Efficient Training of Feed-Forward Neural Networks*" Ph.D. dissertation, Computer Science Department, Århus University, Århus, Denmark, December 1993.

[Prechelt-94.2] Lutz Prechelt. "PROBEN1 - A Set of Network Benchmark Problems and Benchmarking Rules." Technical Report 21/94, Universität Karlsruhe, September 1994.

[Riedmiller-93] Martin Riedmiller and Heinrich Braun. "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm." In *Proccedings of the 1993 IEEE International Conference on Neural Networks* (IEEE; San Francisco, California; March 28–April 1) (IEEE Catalog Number: 93CH3274-8), volume 1, pages 586–591. IEEE, 1993.

[Rumelhart-86.2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning internal representations by error propagation." In *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume I: Foundations, chapter 8, pages 318–362. MIT Press, Cambridge, Massachusetts, 1986.

[Salomon-90] Ralf Salomon. "Improved convergence rate of back-propagation with dynamic adaptation of the learning rate." In Hans-Paul Schwefel and Reinhard Männer (editors), *Parallel Problem Solving from Nature I* (Dortmund, FRG; October 1–3) (ISBN: 3-540-54148-9), volume 496 of *Lecture Notes in Computer Science*, pages 269–273. Springer-Verlag, 1990.

[Salomon-92] Ralf Salomon and Leo Van Hemmen. "A new learning scheme for dynamic self-adaptation of learning-relevant parameters." In Igor Aleksander and John Taylor (editors), *Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92)* (Brighton, UK; September 4–7), volume 2, pages 1047–1050. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 1992.

[Schmidhuber-89] Jürgen Schmidhuber. "Accelerated Learning in Back-Propagation Nets." In R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, and L. Steels (editors), *Connectionism in Perspective*, pages 439–445. Elsevier Science Publishers B. V., Amsterdam - North Holland, 1989.

[Silva-90] Fernando M. Silva and Luis B. Almeida. "Speeding Up Backpropagation." In Rolf Eckmiller (editor), *Advanced Neural Computers ((Proceedings of the) International Symposium on Neural Networks for Sensory and Motor Systems (NSMS))* (Neuss, Germany; March 22–24, 1990) (ISBN: 0-444-88400-9 (U.S.); LCCCN: QA76.87.A37), pages 151–158. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, The Netherlands, 1990.

[Vogl-88] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon. "Accelerating the Convergence of the Back-Propagation Method." *Biological Cybernetics*, volume 59, pages 257–263. Berlin, Germany, 1988.

[Watrous-87] Raymond L. Watrous. "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization." In Maureen Caudill and Charles Butler (editors), *Proceedings of the IEEE First International Conference on Neural Networks (ICNN)* (IEEE; San Diego, California; June 21–24, 1987) (IEEE Catalog Number: 87TH0191-7), volume II, pages 619–627. SOS Printing, San Diego, California, 1987.

[Yu-95] Xiao-Hu Yu, Guo-An Chen, and Shi-Xin Cheng. "Dynamic Learning Rate Optimization of the Back-propagation Algorithm." *IEEE Transactions on Neural Networks* (ISSN: 1045-9227), volume 6, number 3, pages 669–677. IEEE, May 1995.

A Tables with the results

Method	η_0	α_0	%Conv	Number of Epochs			
				Min	Max	Mean	σ
On-Line BP	0.5	0.9	100.0	161	434	262.2	56.2
	0.1	0.9	100.0	768	2284	1382.2	504.1
	0.05	0.9	100.0	1519	6844	2902.2	1351.7
	0.5	0.5	100.0	720	1488	1033.4	197.1
	0.1	0.5	100.0	3762	11838	6365.5	2101.6
	0.05	0.5	100.0	7574	26384	13286.2	5121.8
Batch BP	0.5	0.9	100.0	33	55	44.0	6.0
	0.1	0.9	100.0	43	139	74.8	26.0
	0.05	0.9	100.0	64	146	86.5	21.1
	0.5	0.5	100.0	48	130	94.8	20.1
	0.1	0.5	100.0	91	390	164.2	70.9
	0.05	0.5	100.0	176	528	243.1	81.4
SCG	–	–	100.0	20	42	28.4	5.6
Chan-Fallside	0.5	0.5, 0.9	100.0	162	856	338.1	165.9
	0.1	0.5, 0.9	100.0	166	816	339.5	155.8
	0.05	0.5, 0.9	100.0	170	813	340.8	154.8
Silva-Almeida	0.05	0.9	100.0	34	87	63.5	15.0
	0.01	0.9	100.0	42	80	59.3	11.5
	0.005	0.9	95.0	*47*	*87*	*58.0*	*10.1*
	0.05	0.5	100.0	47	100	66.3	13.6
	0.01	0.5	100.0	53	101	65.3	12.1
	0.005	0.5	100.0	48	85	64.7	10.9
RPROP	–	–	100.0	42	137	71.6	23.2
Salomon-van Hemmen	0.5	0.9	100.0	8	74	22.1	14.2
	0.1	0.9	100.0	14	46	30.2	8.1
	0.05	0.9	100.0	21	111	40.0	20.2
	0.5	0.5	100.0	21	90	49.0	18.2
	0.1	0.5	100.0	24	165	59.1	36.9
	0.05	0.5	100.0	22	135	57.7	27.5

Table 7: Results from the simulations with the **Exclusive-OR** benchmark.

Method	η_0	α_0	%Conv	Number of Epochs			
				Min	Max	Mean	σ
On-Line BP	0.5	0.9	65.0	*217*	*409*	*309.6*	*60.5*
	0.1	0.9	95.0	*1013*	*4784*	*2399.7*	*1113.9*
	0.05	0.9	90.0	*1947*	*12616*	*5502.6*	*3312.1*
	0.5	0.5	80.0	*903*	*2681*	*1301.2*	*424.9*
	0.1	0.5	95.0	*5004*	*18178*	*9011.5*	*3896.6*
	0.05	0.5	85.0	*9655*	*27288*	*16236.4*	*4666.4*
Batch BP	0.5	0.9	60.0	*32*	*6307*	*673.8*	*1720.3*
	0.1	0.9	90.0	*48*	*193*	*83.3*	*32.8*
	0.05	0.9	95.0	*69*	*236*	*120.6*	*48.5*
	0.5	0.5	75.0	*51*	*5607*	*455.5*	*1376.9*
	0.1	0.5	90.0	*91*	*429*	*186.2*	*90.8*
	0.05	0.5	95.0	*161*	*761*	*299.7*	*146.4*
SCG	–	–	85.0	*16*	*54*	*33.1*	*9.6*
Chan-Fallside	0.5	0.5, 0.9	90.0	*148*	*1511*	*468.2*	*347.5*
	0.1	0.5, 0.9	95.0	*154*	*1893*	*482.4*	*391.1*
	0.05	0.5, 0.9	95.0	*157*	*1902*	*484.2*	*392.1*
Silva-Almeida	0.05	0.9	65.0	*50*	*113*	*81.4*	*20.7*
	0.01	0.9	45.0	*51*	*101*	*64.7*	*16.4*
	0.005	0.9	70.0	*52*	*112*	*72.9*	*15.8*
	0.05	0.5	60.0	*48*	*106*	*70.2*	*18.9*
	0.01	0.5	65.0	*51*	*115*	*72.7*	*21.5*
	0.005	0.5	70.0	*50*	*119*	*67.9*	*16.5*
RPROP	–	–	55.0	*50*	*154*	*87.8*	*28.3*
Salomon-van Hemmen	0.5	0.9	95.0	*8*	*145*	*31.9*	*34.2*
	0.1	0.9	90.0	*13*	*60*	*28.9*	*12.8*
	0.05	0.9	100.0	14	63	32.8	13.1
	0.5	0.5	100.0	13	94	33.1	21.6
	0.1	0.5	90.0	*18*	*111*	*52.4*	*27.1*
	0.05	0.5	95.0	*20*	*275*	*61.7*	*54.8*

Table 8: Results from the simulations with the **Exclusive-OR(2)** benchmark.

Method	η_0	α_0	%Conv	Number of Epochs			
				Min	Max	Mean	σ
On-Line BP	0.5	0.9	50.0	*5*	*328*	*117.6*	*138.0*
	0.1	0.9	100.0	2	5	3.8	0.9
	0.05	0.9	100.0	4	7	5.4	0.8
	0.5	0.5	100.0	3	17	5.2	4.0
	0.1	0.5	100.0	13	21	15.7	2.8
	0.05	0.5	100.0	31	44	35.5	4.1
Batch BP	0.5	0.9	100.0	7	10	8.3	1.1
	0.1	0.9	100.0	14	20	16.7	1.9
	0.05	0.9	100.0	20	29	24.0	2.5
	0.5	0.5	100.0	8	12	9.4	1.0
	0.1	0.5	100.0	20	28	21.7	2.3
	0.05	0.5	100.0	35	50	39.1	4.1
SCG	–	–	100.0	11	17	13.2	1.8
Chan-	0.5	0.5, 0.9	100.0	7	9	7.8	0.7
Fallside	0.1	0.5, 0.9	100.0	9	12	10.3	0.9
	0.05	0.5, 0.9	100.0	10	13	11.9	0.9
Silva-	0.05	0.9	100.0	13	17	15.4	1.4
Almeida	0.01	0.9	100.0	20	24	22.6	1.4
	0.005	0.9	100.0	24	27	25.9	1.1
	0.05	0.5	100.0	15	19	15.8	1.2
	0.01	0.5	100.0	22	28	24.0	1.5
	0.005	0.5	100.0	26	36	28.1	2.7
RPROP	–	–	100.0	10	14	11.9	1.4
Salomon-	0.5	0.9	100.0	6	8	6.7	0.6
van Hemmen	0.1	0.9	100.0	10	13	11.4	1.0
	0.05	0.9	100.0	13	17	15.1	1.4
	0.5	0.5	100.0	6	11	8.5	1.4
	0.1	0.5	100.0	11	14	12.3	0.9
	0.05	0.5	100.0	12	16	14.8	1.0

Table 9: Results from the simulations with the **Promoters** benchmark.

Method	η_0	α_0	%Conv	Number of Epochs			
				Min	Max	Mean	σ
On-Line BP	0.5	0.9	0.0	–	–	–	–
	0.1	0.9	100.0	184	554	355.0	128.3
	0.05	0.9	100.0	300	645	434.5	110.0
	0.5	0.5	90.0	*219*	*829*	*386.8*	*209.7*
	0.1	0.5	90.0	*560*	*914*	*751.1*	*118.0*
	0.05	0.5	100.0	1108	1729	1390.9	216.7
Batch BP	0.5	0.9	90.0	*176*	*349*	*244.1*	*49.3*
	0.1	0.9	90.0	*215*	*1457*	*655.1*	*337.4*
	0.05	0.9	100.0	236	1978	648.8	489.6
	0.5	0.5	100.0	864	1149	996.5	81.0
	0.1	0.5	100.0	1443	4880	2021.0	970.1
	0.05	0.5	100.0	1455	5112	2068.2	1029.1
SCG	–	–	100.0	108	309	172.8	57.1
Chan-	0.5	0.5, 0.9	70.0	*845*	*1741*	*1205.7*	*272.2*
Fallside	0.1	0.5, 0.9	60.0	*879*	*1168*	*1080.3*	*101.6*
	0.05	0.5, 0.9	80.0	*952*	*1421*	*1203.4*	*151.0*
Silva-	0.05	0.9	60.0	*172*	*396*	*269.2*	*90.6*
Almeida	0.01	0.9	40.0	*131*	*188*	*162.8*	*21.4*
	0.005	0.9	50.0	*139*	*1232*	*421.2*	*410.9*
	0.05	0.5	10.0	*701*	*701*	*701.0*	*0.0*
	0.01	0.5	0.0	–	–	–	–
	0.005	0.5	0.0	–	–	–	–
RPROP	–	–	40.0	*294*	*833*	*544.5*	*230.9*
Salomon-	0.5	0.9	100.0	425	1343	779.5	262.9
van Hemmen	0.1	0.9	90.0	*529*	*1123*	*703.6*	*200.5*
	0.05	0.9	100.0	351	1763	903.6	419.7
	0.5	0.5	100.0	455	1109	723.1	196.0
	0.1	0.5	100.0	499	1105	741.3	178.8
	0.05	0.5	100.0	335	1653	728.3	391.5

Table 10: Results from the simulations with the **Sonar** benchmark.

Method	η_0	α_0	%Conv	Number of Epochs			
				Min	Max	Mean	σ
On-Line BP	0.5	0.9	0.0	–	–	–	–
	0.1	0.9	60.0	*918*	*2286*	*1718.0*	*490.6*
	0.05	0.9	100.0	1815	4086	2726.4	781.7
	0.5	0.5	50.0	*1086*	*3804*	*1981.2*	*1007.5*
	0.1	0.5	100.0	3285	3648	3483.1	125.3
	0.05	0.5	100.0	6418	8383	7048.3	592.9
Batch BP	0.5	0.9	100.0	428	838	602.1	147.6
	0.1	0.9	100.0	461	870	671.1	159.9
	0.05	0.9	100.0	465	585	524.2	33.1
	0.5	0.5	100.0	1769	2029	1895.9	74.7
	0.1	0.5	100.0	2479	2934	2680.4	154.4
	0.05	0.5	100.0	2658	3103	2831.9	152.7
SCG	–	–	0.0	–	–	–	–
Chan-Fallside	0.5	0.5, 0.9	40.0	*4788*	*5157*	*4948.0*	*140.4*
	0.1	0.5, 0.9	10.0	*4720*	*4720*	*4720.0*	*0.0*
	0.05	0.5, 0.9	30.0	*5723*	*6304*	*5997.3*	*238.3*
Silva-Almeida	0.05	0.9	60.0	*142*	*276*	*207.7*	*54.4*
	0.01	0.9	90.0	*154*	*399*	*224.3*	*73.3*
	0.005	0.9	90.0	*167*	*392*	*276.0*	*84.9*
	0.05	0.5	60.0	*432*	*881*	*642.0*	*130.2*
	0.01	0.5	50.0	*512*	*1165*	*701.6*	*235.4*
	0.005	0.5	60.0	*407*	*768*	*542.0*	*112.5*
RPROP	–	–	70.0	*491*	*1354*	*788.6*	*270.4*
Salomon-van Hemmen	0.5	0.9	100.0	995	1299	1127.4	86.9
	0.1	0.9	100.0	892	2098	1140.9	336.2
	0.05	0.9	100.0	954	1359	1163.2	118.6
	0.5	0.5	100.0	902	1419	1073.6	130.0
	0.1	0.5	100.0	850	1334	1069.4	131.8
	0.05	0.5	100.0	925	2427	1233.5	412.8

Table 11: Results from the simulations with the **Vowels** benchmark.

Method	η_0	α_0	Number of Epochs		Square Error Percentage (Mean)			Percentage of Misclassification					
								Training		Validation		Test	
			Mean	σ	Train.	Valid.	Test	Mean	σ	Mean	σ	Mean	σ
On-Line BP	0.5	0.9	224.0	18.0	0.610	0.433	0.470	17.2	0.5	10.9	1.4	11.6	0.5
	0.1	0.9	993.5	63.8	0.580	0.457	0.442	15.1	0.3	15.6	0.7	10.0	0.6
	0.05	0.9	1948.5	136.2	0.582	0.464	0.441	15.1	0.6	15.7	0.7	9.0	1.0
	0.5	0.5	998.0	61.0	0.579	0.456	0.444	15.3	0.3	15.8	0.8	11.2	0.5
	0.1	0.5	4876.5	363.9	0.581	0.466	0.442	15.2	0.6	15.9	0.5	9.1	0.8
	0.05	0.5	7428.0	469.5	0.641	0.500	0.475	15.7	0.7	17.9	1.0	9.1	1.5
Batch BP	0.5	0.9	64.5	30.6	0.626	0.607	0.453	17.1	1.5	17.2	1.2	13.8	0.9
	0.1	0.9	121.0	3.0	0.621	0.513	0.439	14.7	0.5	15.2	0.7	8.8	0.9
	0.05	0.9	210.0	11.6	0.592	0.501	0.430	14.3	0.8	15.8	1.0	9.3	1.7
	0.5	0.5	227.0	33.3	0.574	0.452	0.438	15.6	0.8	14.9	0.8	11.7	0.7
	0.1	0.5	464.0	20.3	0.579	0.463	0.439	14.9	0.8	15.4	0.5	9.6	1.0
	0.05	0.5	895.0	19.6	0.581	0.467	0.441	15.2	0.6	15.8	0.7	9.1	0.8
SCG	–	–	768.0	1290.7	0.587	0.484	0.438	15.6	1.2	15.2	1.1	9.2	1.0
Chan-Fallside	0.5	0.5, 0.9	320.5	103.0	0.613	0.579	0.440	14.7	0.6	15.6	0.7	11.7	1.0
	0.1	0.5, 0.9	312.0	16.5	0.616	0.577	0.443	14.4	0.5	15.7	0.7	11.8	1.0
	0.05	0.5, 0.9	379.0	187.0	0.612	0.565	0.454	14.4	0.4	16.0	0.7	11.6	1.0
Silva-Almeida	0.05	0.9	104.0	30.2	0.602	0.590	0.431	16.0	1.2	16.8	1.5	11.6	1.2
	0.01	0.9	102.5	28.9	0.616	0.598	0.418	16.4	2.0	16.4	1.3	11.2	1.3
	0.005	0.9	125.5	42.3	0.599	0.586	0.438	16.1	2.0	16.0	1.7	10.7	0.9
	0.05	0.5	66.0	8.3	0.624	0.569	0.419	16.1	0.7	14.7	0.9	10.9	0.5
	0.01	0.5	76.0	15.5	0.624	0.581	0.423	16.1	0.9	15.9	0.7	10.9	0.5
	0.005	0.5	76.0	13.9	0.633	0.577	0.426	17.1	1.2	15.7	1.7	10.9	0.7
RPROP	–	–	219.0	80.6	0.606	0.577	0.428	15.8	1.5	15.3	1.6	10.8	1.7
Salomon-van Hemmen	0.5	0.9	60.5	17.4	0.632	0.603	0.440	16.6	1.6	15.7	1.9	12.2	0.5
	0.1	0.9	87.0	26.3	0.667	0.609	0.455	19.0	2.0	15.3	1.4	12.8	1.0
	0.05	0.9	146.0	127.6	0.622	0.589	0.447	17.1	1.5	15.3	1.6	12.0	1.4
	0.5	0.5	79.5	30.0	0.623	0.594	0.484	17.1	2.2	16.2	2.2	13.4	1.9
	0.1	0.5	97.5	29.1	0.620	0.610	0.442	17.2	2.0	16.2	1.5	12.7	1.2
	0.05	0.5	98.5	32.5	0.637	0.603	0.445	17.1	2.7	16.0	1.8	12.8	0.7

Table 12: Results from the simulations with the **Auto-MPG** benchmark.

Method	η_0	α_0	Number of Epochs		Square Error Percentage (Mean)			Percentage of Misclassification					
								Training		Validation		Test	
			Mean	σ	Train.	Valid.	Test	Mean	σ	Mean	σ	Mean	σ
On-Line BP	0.5	0.9	55.0	28.9	4.459	5.054	4.957	35.0	22.8	38.4	21.1	38.0	21.6
	0.1	0.9	126.0	115.9	0.068	1.171	0.967	0.5	0.1	7.0	0.9	5.2	0.8
	0.05	0.9	495.5	438.0	0.067	1.222	0.939	0.4	0.2	7.3	1.0	5.2	0.5
	0.5	0.5	281.0	240.6	0.047	1.203	0.902	0.4	0.2	7.2	1.0	5.1	0.4
	0.1	0.5	964.0	699.2	0.047	1.245	0.905	0.3	0.1	7.5	1.0	4.6	0.7
	0.05	0.5	1518.0	948.4	0.045	1.251	0.911	0.3	0.1	7.5	1.0	4.7	0.6
Batch BP	0.5	0.9	107.0	40.0	1.674	2.636	2.445	16.4	11.2	21.2	10.4	19.9	9.9
	0.1	0.9	181.0	47.2	0.147	1.306	1.031	1.3	3.0	8.2	3.2	6.0	3.1
	0.05	0.9	229.0	44.1	0.039	1.267	0.927	0.3	0.1	7.5	1.0	5.4	0.7
	0.5	0.5	397.5	84.7	0.033	1.242	0.914	0.3	0.1	7.3	1.2	4.9	0.5
	0.1	0.5	533.0	126.6	0.037	1.241	0.910	0.3	0.1	7.4	1.1	4.7	0.6
	0.05	0.5	714.0	129.0	0.037	1.243	0.907	0.3	0.1	7.4	1.1	4.8	0.6
SCG	–	–	24.0	23.4	9.803	9.805	9.805	89.4	1.7	89.4	1.5	89.2	2.0
Chan-Fallside	0.5	0.5, 0.9	72.5	29.8	2.031	3.005	2.908	19.8	7.2	24.6	6.1	24.0	6.1
	0.1	0.5, 0.9	89.0	22.3	1.382	2.494	2.312	13.6	4.3	19.1	3.3	17.6	4.1
	0.05	0.5, 0.9	103.0	82.1	1.028	2.116	1.967	9.9	5.3	15.0	5.4	14.4	4.9
Silva-Almeida	0.05	0.9	84.0	16.9	0.556	1.723	1.627	5.5	4.9	11.7	4.4	11.1	3.9
	0.01	0.9	88.0	27.8	0.451	1.674	1.532	3.8	4.5	10.4	4.3	9.5	4.1
	0.005	0.9	78.5	26.9	0.254	1.525	1.417	1.8	3.2	8.8	3.3	8.2	2.6
	0.05	0.5	72.5	10.3	0.230	1.456	1.312	0.9	0.4	8.0	0.7	6.8	0.7
	0.01	0.5	80.0	11.4	0.190	1.437	1.313	0.6	0.4	7.2	0.8	7.0	0.9
	0.005	0.5	99.5	27.2	0.190	1.426	1.300	0.7	0.5	7.3	0.8	6.8	0.8
RPROP	–	–	60.5	13.5	0.236	1.707	1.981	0.9	0.7	9.4	1.6	10.2	1.1
Salomon-van Hemmen	0.5	0.9	194.0	60.9	0.038	1.185	0.954	0.3	0.1	7.1	1.1	5.2	0.7
	0.1	0.9	189.0	44.1	0.128	1.282	0.997	1.3	3.0	8.1	3.2	5.8	3.0
	0.05	0.9	214.0	48.5	0.046	1.205	0.917	0.4	0.2	7.2	1.1	4.9	0.7
	0.5	0.5	198.0	58.3	0.037	1.228	0.931	0.3	0.2	7.3	1.2	5.0	0.7
	0.1	0.5	192.5	60.0	0.065	1.212	0.921	0.3	0.2	7.3	1.3	5.0	0.6
	0.05	0.5	181.0	51.6	0.054	1.197	0.907	0.3	0.2	7.1	1.3	5.0	0.6

Table 13: Results from the simulations with the **Digit** benchmark.

Method	η_0	α_0	Number of Epochs		Square Error Percentage (Mean)			Percentage of Misclassification					
								Training		Validation		Test	
			Mean	σ	Train.	Valid.	Test	Mean	σ	Mean	σ	Mean	σ
On-Line BP	0.5	0.9	76.5	119.1	0.045	1.732	4.287	0.0	0.0	3.2	1.1	9.1	2.1
	0.1	0.9	55.0	65.4	0.289	1.875	3.639	0.0	0.0	2.7	0.9	8.4	1.9
	0.05	0.9	219.0	106.6	0.062	1.896	4.282	0.0	0.0	2.7	0.9	10.7	0.9
	0.5	0.5	91.0	28.0	0.069	1.871	4.078	0.0	0.0	2.7	0.9	9.6	1.0
	0.1	0.5	541.5	218.5	0.060	1.887	4.348	0.0	0.0	3.0	1.0	10.7	0.9
	0.05	0.5	1160.5	436.6	0.051	1.888	4.394	0.0	0.0	3.0	1.0	10.7	0.9
Batch BP	0.5	0.9	35.5	14.4	0.048	1.676	4.491	0.0	0.0	2.5	0.7	8.9	2.0
	0.1	0.9	58.0	8.1	0.040	1.755	4.392	0.0	0.0	3.0	1.0	9.3	0.9
	0.05	0.9	79.5	4.7	0.043	1.815	4.330	0.0	0.0	2.7	0.9	9.6	1.0
	0.5	0.5	101.0	14.3	0.100	1.726	3.884	0.0	0.0	2.3	0.0	8.9	1.0
	0.1	0.5	155.5	13.9	0.046	1.834	4.319	0.0	0.0	2.5	0.7	10.2	1.1
	0.05	0.5	254.5	12.9	0.040	1.870	4.464	0.0	0.0	3.0	1.0	10.7	0.9
SCG	–	–	65.5	57.0	0.171	1.949	4.568	0.1	0.3	3.9	1.0	8.9	2.0
Chan-Fallside	0.5	0.5, 0.9	76.0	69.4	0.150	1.633	3.951	0.1	0.3	2.5	0.7	7.8	1.8
	0.1	0.5, 0.9	148.5	103.8	0.066	1.722	4.219	0.1	0.3	3.2	1.1	8.0	2.3
	0.05	0.5, 0.9	120.5	107.0	0.153	1.662	3.995	0.2	0.4	2.5	0.7	7.8	2.5
Silva-Almeida	0.05	0.9	50.0	27.3	0.367	1.685	3.713	0.2	0.4	3.0	1.0	6.2	3.1
	0.01	0.9	65.0	47.6	0.189	1.657	4.329	0.2	0.4	3.6	1.5	7.6	1.1
	0.005	0.9	49.5	14.6	0.333	1.990	4.077	0.3	0.5	4.8	3.0	7.3	2.0
	0.05	0.5	60.0	31.8	0.509	2.026	3.631	0.3	0.5	3.4	1.5	6.7	2.4
	0.01	0.5	74.0	41.5	0.182	1.729	3.601	0.1	0.3	3.0	1.0	6.4	1.2
	0.005	0.5	77.0	33.3	0.200	1.796	3.476	0.0	0.0	3.4	1.1	6.2	1.3
RPROP	–	–	60.5	37.2	0.426	1.437	3.776	0.3	0.5	2.3	1.4	7.8	2.0
Salomon-van Hemmen	0.5	0.9	51.5	15.2	0.056	1.640	4.222	0.0	0.0	3.2	1.1	9.3	1.9
	0.1	0.9	40.5	11.7	0.126	1.707	3.959	0.0	0.0	3.0	1.5	8.0	1.8
	0.05	0.9	43.0	18.5	0.134	1.683	3.881	0.1	0.3	3.4	1.1	8.0	2.3
	0.5	0.5	38.5	13.8	0.202	1.592	3.533	0.0	0.0	2.3	1.0	6.7	2.2
	0.1	0.5	40.0	21.1	0.234	1.765	4.047	0.3	0.5	3.2	1.5	8.0	2.5
	0.05	0.5	45.5	14.2	0.096	1.703	4.251	0.1	0.3	2.7	0.9	9.1	1.6

Table 14: Results from the simulations with the **Wine** benchmark.